Embedding Learning in Behaviour-Based Architectures: A Conceptual Approach*

João Sequeira**, **José del R. Millán, M. Isabel Ribeiro, João G. M. Gonçalves**

Instituto Superior Técnico/Instituto de Sistemas e Robótica
Av. Rovisco Pais 1, 1096 Lisboa CODEX, Portugal Tel.: +351-1-8418057, Fax.: +351-1-8418291
e-mails: jseq@isr.ist.utl.pt mir@isr.ist.utl.pt
European Commission Joint Research Centre (JRC), TP 270, 21020 Ispra (VA), Italy
Tel.: +39-332-789416, Fax.: +39-332-789185
e-mails: jose.millan@jrc.it joao.goncalves@jrc.it

Abstract: This paper addresses the identification of the key elements to be present in a generic, *behaviour*-based architecture for robot control. These elements include the definition of *behaviours* aiming at rough task execution and the use of learning to improve the performance of the primitive *behaviour* set. Experimental results, with real and simulated systems, to demonstrate cooperative behaviour between a mobile platform and a manipulator are presented.

Keywords: Behaviour, Learning, Cooperation.

1 Introduction

Although there are several successful application examples of *behaviour*-based architectures, [2], [3], [4], [7], [9], there still is a lack of methodologies for representation, synthesis and analysis of this approach. This paper proposes a representation with properties suitable to formal analysis, namely, in what concerns controllability.

The paper is organized as follows. Section 2 defines the concepts used in the remaining of the paper. Section 3 proposes an architecture supported on the concepts previously defined. Section 4 introduces the use of learning techniques into the architecture, as a framework related with controllability concepts. Section 5 presents an experimental setup used to validate the concepts defined in the paper, together with results of several experiments using robots with different kinematic constraints acting cooperatively in two kinds of missions. Finally, section 6 concludes the paper.

2 Agent fundamentals

The concept of agent, with roots in the Artificial Intelligence community, encapsulates most of the aspects of the *behaviour* approaches, allowing problems with multiple heterodox robotic mechanisms to be treated under the same terminology. We consider an agent as being defined by the information pipeline *Environment* $\mapsto$ *Sensors* $\mapsto$ *Behaviours* $\mapsto$ *Actuators* $\mapsto$ *Robotic device.*

The information on the agent state and the task to be executed is contained in the following sets:

S	any internal state variables, e.g., battery level, sensing data,
Q	robot configuration space, or C-space,
A	set of available motion strategies or actions,
$T = Q \times A$	agent state set.

The sensor and/or state information circulates between an agent and the environment in bursts representing the system events. At each event the agent takes an action, defined as a mapping $a : Q \mapsto Q$[3]. The domain set contains the initial conditions of the action. The range set contains the trajectory in the C-space generated, in time, by the action[4].

The configuration, q, and the action, a, taken by the agent at an event, completely determine the trajectory of the robot, until the next event is detected. Therefore, an appropriate state variable for the agent is defined by a tuple $(q, a) \in T$. This definition contains the information about the current motion strategy of the agent and also some memory of the close past, embedded on the initial condition for the action. If an agent is fed back

*Partially sponsored by the programme PRAXIS XXI, project COOPERA - 2/2.1/TPAR/2087/95.

**Corresponding author, working under NATO grant 9/A/95/PO.

[3]We use the term *action* with the meaning of a trajectory generator, without semantic concerns, and the term *behaviour* if some semantic content is implicit.

[4]The time parameter can be assumed implicit.

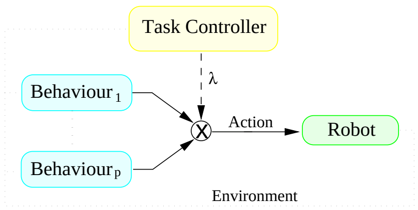

Figure 1: Conceptual *behaviour* architecture.

with the complete state of other agent it will know exactly what kind of motion strategy the other is taking, and may use that information when making its own decisions.

3 A conceptual *behaviour*-based architecture

The basic structure that allows the generation of motion in the state set consists of a closed operation, $\circ_T : T \times T \mapsto T$, which gives to the set the structure of an algebraic group. Consider an agent in the state $t_i = (q_i, a_i)$. At an event some decision mechanism requires the agent to switch to the state $t_j = (q_j, a_j)$. The operation $\circ_T$ can be interpreted as the mechanism that generates a path in Q connecting t_i with t_j. This interpretation is compatible with the well known group operation properties, requiring the existence of: **(i)** a null element of the group, meaning that at each event it must be always possible *to do nothing*, or *to do something that does not compromise future decisions*; **(ii)** an inverse for each element of the group, thus enabling the possibility of coming back to any state, e.g., for a replanning operation; **(iii)** the associativity property, meaning that the group operation does not make any special distinctions between states.

Fig. 1 shows the conceptual architecture based in the group structure. The *Task Controller* is responsible for the combination of the basic *behaviours*, $b_1, \ldots, b_p$, belonging to A, into more complex *behaviours*. The operation $\circ_T$ is expressed in the $\otimes$ block. λ represents the control variable for the combination of the basic *behaviours*, b_i.

The righmost levels of the agent information pipeline are represented by a continuous dynamical system, $\dot{q} = f(q, u)$, with $q \in Q$ and u being the control variable. Under the hypothesis of weak controllability of this system, the inner level of the agent can be expressed by the blocks $\{K, L_1, \ldots, L_m, \oplus, \int\}$ for some finite m, [8]. The L_i represent vector fields able to generate the trajectories in Q and K is a controller able to drive the robot through the reference configurations q_{ref}. The weak controllability hypothesis is not too restrictive. In fact, the *behaviour* paradigm is meaningful only if it exists a K able to drive the robot between any two points of its configuration space. Fig. 2 shows

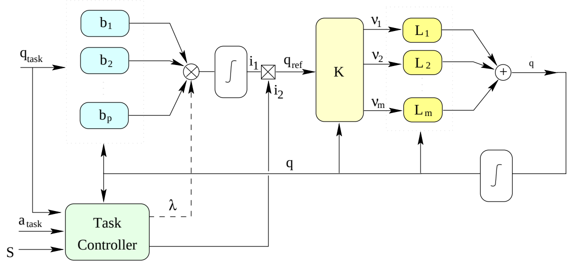

Figure 2: The complete *Behaviour*-based architecture.

a graphical representation of this model, merged with the abstract model of Fig. 1. The merging of the *Behaviour Structure*, defined by the blocks $\{b_1, \ldots, b_p, \textit{Task Controller}, \int, \otimes\}$, with the *Robot Structure*, defined by $\{K, L_1, \ldots, L_m, \oplus, \int\}$ is made through an *Interface Block*, represented by the symbol $\boxtimes$. This block is controlled by the *Task Controller* and can also be used to bypass the *Behaviour Structure*. Any configuration imposed at input i_2 is directly transferred to the output, overriding the contents of i_1.

The *Task Controller* block uses specific information about the class of tasks the agent is able to perform, e.g., the criteria that define which action to apply under the range of environmental conditions considered, to define the control variable λ sent to the $\otimes$ block, and, when necessary, point-to-point actions, sent directly to the $\boxtimes$ block.

4 Task Controller design, agent controllability and learning

The *Task Controller* design relies on the principle that the agent is equipped with a set of *behaviours* powerful enough to handle all the events that may arise in the environment where it operates. It is the responsability of the programmer to provide this set of *behaviours*. This principle represents the controllability concept used in the theory of Discrete Event Systems (DES), [6]. Agent definition using DES theory considers an event as a symbol of some alphabet, and a *behaviour* as a trajectory (i.e., a string of symbols) in that alphabet [1], [3]. In DES based frameworks, an agent is controllable if it satisfies a specified *behaviour* (in the DES sense), independently of the events that may occur.

The DES controllability concept provides also the methodology for the design of the *Task Controller*. It identifies all the environment situations that can be classified as *events*, and, for each *event*, provides a *behaviour* able to handle it. This results in a Finite State Machine (FSM) based design for the *Task Controller*, making the agent controllable in the sense used in DES[5], i.e., the agent

[5]And therefore in the usual control sense.

is able to handle the set of situations that were explicitly considered in the initial design.

A one-to-one correspondence between a *behaviour* and a coordinate of R^p (the space of the λs) can be assumed[6]. It is possible to define a number, m, of subspaces such that $\bigcup_{i=1}^{m} R^{p_i} = R^p$ and $R^{p_i} \bigcap R^{p_j} = \emptyset, \quad p_i \neq p_j$ and a one-to-one correspondence between events and subspaces R^{p_i} is created.

The initial map *events* $\mapsto R^p$ is defined by the *Task Controller* FSM, which, by the previous discussion about DES, assures the agent controllability. In general, this map may have as its range only some constrained subspace of R^p. This is the case, for example, in a "potential field" based obstacle avoidance, where the λ represent weights that depend on some distance between the robot and the obstacles. The next logical step is to incorporate the means to improve the performance with experience, without destroying the controllability property. This leads to the use of learning techniques.

The division of the space R^p into smaller, unconstrained, subspaces enables the use of Incremental Topology Preserving Maps (ITPM), together with Reinforcement Learning techniques [4], [5][7]. These maps preserve the topology knowledge being acquired in units, and the time history in links between the units. Knowledge acquired through time, and preserved in the ITPM, can be used to optimize the decision, when facing a situation similar to one already encountered in the past. ITPMs can be used in two different ways in the presented architecture, defined bellow.

Local learning: To improve the basic performance of the agent, expressed in local (i.e., in each of the subspaces R^{p_i}) searches for optimal actions. The ITPM either defines the current best λ to each given sensorial situation, or indicates that an *a priori* fixed λ must be used.

Global learning: To improve the global performance, different subspaces R^{p_i} can be searched. The initial encoding, defined by the above relations, represents the *a priori* knowledge about the map *Perception* $\mapsto$ *Behaviours*. Often, this initial encoding will not be optimal. Optimization can be carried out at three levels: **1:** by allowing different combinations for the construction of each subspace, **2:** by allowing the dimensions, p_i, of the subspaces to be time-varying, and **3:** by accepting some overlap between the subspaces. The first level is defined by all the permutations of the m indices that define the coordinates of the space R^p. The search space that represents the two last levels is composed by all of the m-tuples $(p_1, p_2, \ldots, p_m)$ that verify $\sum_{k=1}^{m} p_k = m + \delta \quad \wedge \quad p_i \leq m, \ i = 1, \ldots, m$ where δ represents the total amount of subspace overlaps (initially $\delta = 0$). ITPMs can be used to generate: **(i)** sequences of m-tuples, **(ii)** local variations (in the sense that they respect the above relation) of each m-tuple.

In this work one ITPM is used to learn, in local learning mode, a trajectory for a mobile platform with a constraint in the motion in one of the coordinates (the orientation) of the C-space. Thus, the ITPM works only in the remaining two coordinates (the position).

5 Cooperation using *behaviours*

The architecture of Fig. 2 was tested, without ITPM stages, in the control of a real mobile manipulator (see Fig. 6), to evaluate the performance of the basic *behaviour* architecture. Simulation studies, using one ITPM stage (for the mobile platform), were also performed. Ongoing work includes the test of the complete architecture in the real experimental setup shown at Fig. 6.

Two missions were considered: **(1)** traversal of a corridor to execute a task, **(2)** task execution in a close neighbourhood of an obstacle.

The experimental setup is organized in three agents: **(Agent 1)** the mobile platform, **(Agent 2)** the first 3 joints of the manipulator, in charge of the positioning in (x, y, z), **(Agent 3)** the last 3 joints of the manipulator, in charge of the orientation (α, β, γ). The partitioning of the manipulator into two agents simplifies the *behaviour* design. It keeps the gripper out of operation until the task neighbourhood is reached, thus minimizing potential conflicts between agents during mission execution. The system may approach the task (x, y, z) position using a trajectory that will make impossible for the gripper to achieve the task orientation. In this case some manouevering, defined by a specific *behaviour*, will be needed.

The *Behaviour Structures* for the three agents were implemented in a Sun SPARC 10 workstation, with a 40 MHz processor, while the *Robot Structures* for the mobile platform and the manipulator were implemented using 68020 processors at 16MHz, in the real setup, and in the workstation refered in the simulation cases.

The *Task Controller* is a FSM in which the output is of the form $\lambda = (\ldots, 0, 1, 0, \ldots)$, i.e., only one *behaviour* is active at a time. For all the three agents the corresponding FSMs are fully interconnected networks, where each state corresponds to a single *behaviour*.

[6] In some sense each coordinate switches on/off and/or weights a basic *behaviour*.

[7] Although the ITPMs work in constrained spaces (surfaces), one of the advantages of this optimization technique lies in the simplicity of the implementation when dealing with unconstrained subspaces.

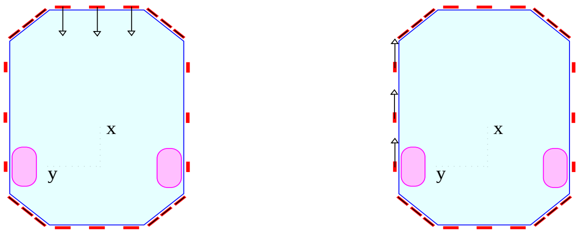

Figure 3: Obstacle avoidance *behaviours*.

5.1 Vehicle *behaviours*

The vehicle used in the experiments is of non holonomic type, with a non-slide kinematic constraint. The following classes of *behaviours* were used (the numbers code the *behaviours* in the graphics of Fig. 5):

Obstacle avoidance (3-4) This class of *behaviours* is divided into two minor subclasses: far from task navigation (3), used whenever the vehicle is out of a neighbourhood, of radius 1.5m, centered in the task, and near to task navigation (4), on the contrary. This subdivision is justified by a particular task execution philosophy: if an obstacle is placed in the trajectory of the vehicle, while far from the end of the task goal, obstacle circunvention is executed. When the vehicle is in the proximity of the task the environment must be scanned in greater detail than before and obstacle circunvention might not be possible. These classes of *behaviours* were chosen because they match the data available from the ring of 24 ultrasonic sensors.

The first class of obstacle avoidance *behaviours* contains two *behaviours*, used respectively for forward and backward motion. If an obstacle is placed near the left or the right side of the vehicle then it moves away in the forward/backward direction, this strategy being compatible with the non-slide constraint. It also avoids collisions generated by the span of a large area, around the vehicle rotation axis. The second class of obstacle avoidance *behaviours* contains three classes of vector fields, to deal with front/back walls and side walls.

Fig. 3 illustrates some of the vector fields used to define the *behaviours*, respectively for forward lateral obstacle avoidance. Each vector is placed over the ultrasonic sensor that generates it.

Null action (10) This *behaviour* is used whenever the vehicle receives information from the arm indicating that it can complete the task without further help from it.

Go straight to goal (11) and single step to goal (7) These *behaviours* determine an error vector between the present and the goal cartesian positions, and use a low level controller that generates appropriate controls for the mobile platform aiming at correcting this error vector. If the vehicle is the task leader, then a single command is sent to the robot (thus saving bandwith).

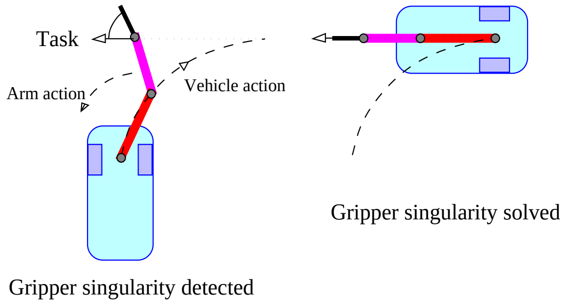

Figure 4: *Escape from gripper singularity* local vector field.

Mission 2 needs an **Escape from gripper singularity** *behaviour*, triggered whenever the gripper signals the vehicle that it can not execute its task, for instance due to physical limitations in the joint values. The vehicle moves according to an algorithm specified *a priori*. Fig. 4 shows a typical pattern for the resulting trajectory.

5.2 Arm *behaviours*

Go straight to goal (2) This *behaviour* is similar to the one described for the vehicle. If the arm becomes stretched above a pre-specified limit it disables the translation motion, keeping just the rotational component.

Approach to goal (1) This *behaviour* is similar to the *go straight to goal*, except that the stretching limit is higher to allow the task completion in situations where it is not physicaly necessary to move the vehicle anymore.

Get out of type 1 singularity region (5) This *behaviour* takes the arm out of the singular configurations characterized by the arm being almost completely stretched. The *behaviour* is specified through the weighted sum of two vector fields. The first one points to the manipulator base position while the second specifies the rotational component to have the arm heading the task.

Get out of type 2 singularity region (6) This singularity region is defined as a cylinder with a symmetry axis along the vertical axis of the manipulator base. This region includes those configurations in which the two manipulator links are close to each other. This *behaviour* is also specified through a weighted sum of two vector fields, specifying a rotational component until the arm is heading to the task, and a translational component, pointing to the task.

Get out of double singularity region (4) This singularity region is defined by a small volume, generated by the intersection of the volumes of the singularity regions 1 and 2. In practical terms, the arm is in this region when it is in the upward position. The vector field associated with the *behaviour* specifies a fixed direction of motion

Task	Task leader	x	y	z	θ	γ
Initial configuration	-	3.93	2.07	0.92	0	0
Approach corridor	Vehicle	3.38	3.06	-	-	-
Corridor traversal	Gripper	0.97	3.06	0.3	-	1.57

Table 1: Mission 1 specification ($\alpha = \beta = 0$ rad).

that drives the arm toward a region of higher maneuverability.

5.3 Gripper *behaviours*

This agent has a single *behaviour*: **Go straight to goal**. The agent is always trying to reach the orientation specified by the task. It generates a singularity warning to the other two agents whenever the whole system is in a 5cm neighbourhood of the task location (x, y, z).

5.4 Mission 1 - Corridor traversal

The approach and corridor traversal mission is defined in Table 1. This mission uses a concept of *Task Leader*, meaning that each task of the mission will have an agent relative to which the task is defined. Figures 5 display the typical results of two runs. The leftmost graphics correspond to the mission of Table 1. In the rightmost graphics the system starts in a random configuration, near the one of the first run. The lowest graphics display the activation sequence of the *behaviours*, each being identified by the integer codes previously specified. The doted lines in the xy plane indicate the position of the corridor structure. Fig. 6 shows the typical views at the beginning (1), in the vicinity of the approach task (2-3) and at the final of the mission (4).

In the first test the vehicle starts heading directly to the first task. When completed, it heads to the second task until the corridor entrance is detected. Obstacle avoidance *behaviours* are used to traverse the corridor until a point is reached such that the arm is able to execute the task, thus activating the null *behaviour*. The arm starts in the upward configuration and therefore it activates the *behaviours* that will get it out from the singularity regions. When out of the singularities, the arm approaches a point in the vertical of the (x, y) point that corresponds to the vehicle task. This means that the arm tries to complete a task that has no significance, from the point of view of the mission, since the leader for this task was defined to be the vehicle. This strategy cause the arm to enter the singularity region 2, thus activating the corresponding *behaviour*. By the time the

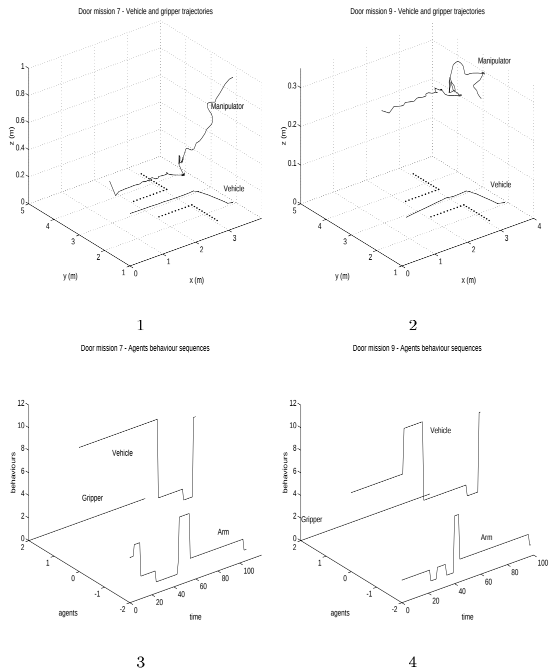

Figure 5: Experimental results - Trajectories and *behaviour* activation sequences.

vehicle is trying to enter the corridor, the arm is already trying to go straight to its goal.

Two forms of traveling to the goal were used for the vehicle: (i) the *single step to goal* (used in the second test) sends a direct position to the position controller of the vehicle, while (ii) the *go straight to goal* (used in the first test) computes local vectors that direct the vehicle to the task. No significant difference in the performance was found. The vehicle single steps to the first task, and the *behaviour* activation pattern for the second task is similar as the one in the first test. As for the arm, it starts by going straight to the goal and, as the vehicle approaches the first task, it switches to the approaching *behaviour*. The remaining activation pattern is similar to the first test.

5.5 Mission 2 - Testing learning

The learning stage of the architecture is composed of one ITPM, [5] that, if necessary, "adapts" the actions suggested by the *behaviours* of the mobile platform[8]. The results presented were obtained with a simulated mobile manipulator, having the same characteristics of the experimental setup. The ITPM was merged with the *behaviour* architecture using a client-server software architecture. Whenever the client (the platform) is not able to communicate with the server (the ITPM) it uses the basic *behaviours* previously established. Otherwise, the action suggested by the ITPM is used.

[8] Note that, in some sense, the ITPM action in this mission is similar to an online adaptation of the λ control.

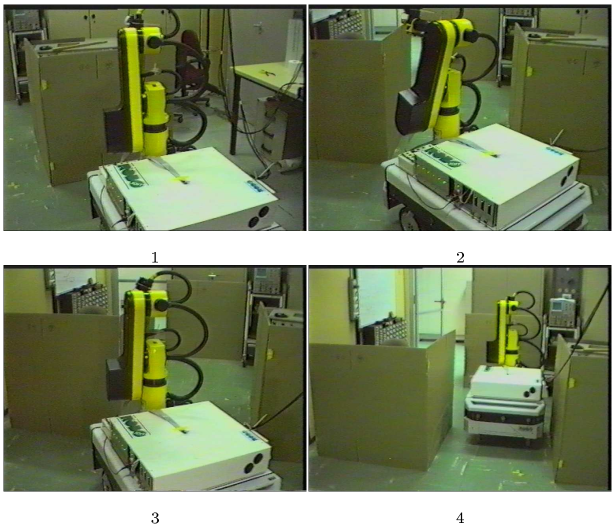

Figure 6: Four views of the door traversal experiment.

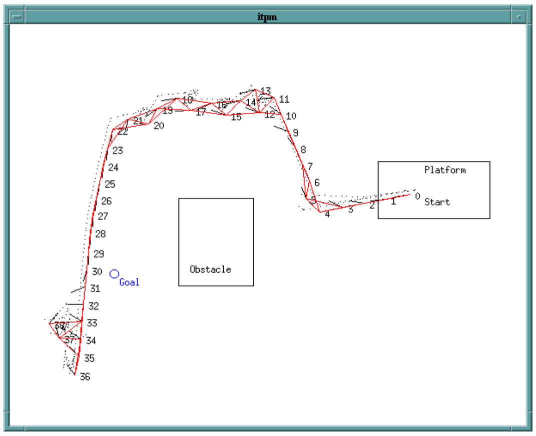

Figure 7: Final ITPM after two runs of mission 2.

Fig. 7 illustrates the ITPM after three mission runs (the mission was successfully completed in the third run). Units are numbered sequentialy, with lines connecting the related ones (see [5] for the possible relations between units). The short line associated with each ITPM unit indicates the motion direction advised by that unit. The first and second runs ended at unit 5, due to the short distance between the mobile manipulator and the obstacle. This was caused by a low obstacle detection threshold. In the third run the system was able to use the previously acquired knowledge to soon start avoiding the obstacle. This is expressed in the interconnections between units 3 to 6. The interconnections in the topmost part of the image express the potential of the ITPM to adapt the actions it proposes, in a unique run. The simple basic *behaviours* used, together with the non-holonomic characteristic of the mobile platform, cause these maneuvers. By tunning the obstacle detection threshold, it is possible to execute the mission using only the basic set of *behaviours*. However, the use of the ITPM enables the use of a lower threshold.

6 Conclusions

The experimental results show consistency in the performance of the system. Despite the sensory limitations of the vehicle, it was possible to traverse a corridor using, in practice, blind navigation. The important aspect is that the obstacle avoidance *behaviours* were able to position the vehicle at the corridor entrance in such a way that blind navigation was an acceptable technique in what concerns the mission execution.

The work presented in this paper is still ongoing. The learning stages described are now being carried on the real system. The *behaviours* used for the corridor traversal example may cause some overshooting in the vehicle orientation at the corridor entrance. This is another example where the use of the ITPM stage (already implemented) to modify the actions proposed by the basic *behaviours* is expected to increase the performance of the system.

References

[1] Antsaklis P., Lemmon M., Stiver J. "Modeling and Design of Hybrid Control Systems," Proceedings of the *2nd IEEE Mediterranean Symposium On New Directions In Control and Automation*, Maleme-Chania, Crete, Greece, June, 1994.

[2] Brooks R. "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of R&A*, RA-2, 1, pp. 14, 1986.

[3] Kosecká J. "Supervisory Control Theory For Autonomous Mobile Agents," PhD Dissertation Proposal, University of Pennsylvania, 1995.

[4] Millan J. "Rapid, Safe, and Incremental Learning of Navigation Strategies," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 26, Part B, No. 3, June 1996.

[5] Millan J. "Incremental Acquisition of Local Networks for the Control of Autonomous Robots," Working paper, December 1996.

[6] Ramadge P., Wonham W. "Supervisory Control Of A Class Of Discrete Event Processes," *SIAM Journal of Control and Optimization*, Vol. 25, No. 1, January, 1987.

[7] Sequeira J., Ribeiro M. I., Gonçalves J. M. G. "Cooperation Among Multiple Robotic Mechanisms," Proceedings of the SIRS'96, Lisbon, 1996.

[8] Sequeira J., Ribeiro M. I. "A Behaviour Based Kernel Architecture For Robot Control," Accepted for presentation at the SYROCO'97, France, September 1997.

[9] Ward K., Arkin R. "Reactive Control of a Mobile Manipulator using Pseudo-joint Damping," Technical Report, Georgia Institute of Technology, 1994.