# Plan execution based on Active Perception: Adding Hints to Plans

**Miguel Schneider-Fontán**

Instituto de Automática Industrial (IAI)

Consejo Superior de Investigaciones Científicas (CSIC)

La Poveda, Arganda del Rey

Madrid 28500

schneider@iai.csic.es

**Abstract**

In this paper we present a complete control architecture for a mobile robot which enables it to achieve a set of proposed goals with a high degree of autonomy and to react to the changing environment in real time. Autonomy and robustness are achieved through careful selection and incremental implementation of a set of basic Motor-Behaviors that interpret the sensor readings (sonar, vision and odometric sensors) and actuate the motors. The plan is provided by a user, and is expressed as a sequence of goals and a series of hints on how to achieve them. These hints are based on the user's knowledge of the environment and of the robot's behavioral and perceptual abilities. A new set of behaviors, called Conductor-Behaviors, which inspect and modify Motor-Behaviors' attributes, have been introduced in order to link the robot's Motor-Behaviors to the user's plan. Finally, a canonical set of symbols, attached to the Motor-Behaviors, serves as well grounded symbols that the user can utilize to express the plans. We also report experimental results with a real robot that demonstrate how plans expressed as goals and hints to achieve them improve the robot's performance.

## I. Introduction

The mobile robotics community has pushed his efforts towards building systems that are robust, with a high degree of autonomy and capable of achieving a set of proposed goals, and reacting to the changing environment in real time.

In order to cope with these premises, *traditional AI* based robots connected sensors to a reasoning-engine (symbolic planner) used by symbolic programs (Fikes and Nilsson 1971, Thomson 1977, Elfes 1986). The classical AI methodology applied to real world robotics pushed all the efforts towards the efficient elaboration of a plan based on the information provided by the sensors (Lozano-Pérez 1983). According to this doctrine, a hierarchy of functional modules was believed to be the soul of the system: higher order modules (symbolic reasoners) demanded symbolic information from lower level modules. These modules would *always* provide the required information appropriately, and different sensor sources should ideally be combined and fused into a complete Cartesian world model, so that the higher level reasoner would get an appropriate symbolic world to plan in (Moravec 1977). Problems with this methodology arose when real time performance could not be achieved, as the elaboration of a world model became the bottleneck of the whole process.

In contrast with this approach, *reactive architectures* (Brooks 1986) proposed no top-down processing and no representation at all, making the system reactive to the changing world. Functional modularization was replaced by behavior generating modules, where a hierarchy of processes was incrementally developed by the designer, each of the processing being extremely simple, and the short path from sensors to actuators provided a real-time interaction between the robot and the world.

However, although robustness and real time performance were achieved by these robots (Brooks 1990b, Connell 1989, Horswill and Brooks 1988), they lacked sufficient flexibility to adapt their goals to a user's pre-defined plan, due to the "hard-wired" concept of this architecture, where the control policy and the system organization is based on a pre-set hierarchy of control.

Trying to overcome this problem, *hybrid architectures* offer a compromise between purely reactive and planner-based approaches by employing a reactive system for low-level control, and a planner for higher-level decision making (Connell 1991, Malcolm and Smithers 1990), using Schemas (Arkin 1989b), or Internalized Plans (Payton 1990). However, a clear and reliable interface between the reactive layer and the symbolic planner in hybrid systems is far from being achieved and well understood. Besides, most hybrid architectures (Saffiotti, Konolige, Myers and Ruspini 1997, Arkin 1989a) still depend on a world model built up from sensor information to plan in.

In this paper we address the following questions: 1) how to scale-up the abilities of a behavior based system, 2) what is the link between the user's plan and the underlying behaviors and 3) how to provide a flexible plan to the robot.

### *A. Plan as "Hints"*

To achieve the main goal of building a behavior-based system that is capable of accomplishing the tasks instructed in a plan, we first develop, implement and debug all the basic behavioral and perceptual abilities of the robot, based on a behavior-based control philosophy. These basic abilities, which we call Motor-Behaviors, form in themselves an "immediate planning" system (Schneider-Fontán 1996).

According to our approach, a control structure is needed that selects the appropriate actions, and tunes the agent's behavior depending on the goals of a high level plan. We believe that this control structure should not be a traditional planning system, like in hybrid architectures, nor need the use of a symbolic model of the environment to select the appropriate actions to be accomplished by the robot. The reason for this is that the controlling structure must be tightly linked to the real world via the Motor-Behaviors, and should serve as a well grounded link between the user's plan and the underlying robot's perceptual and motor skills.

Regarding the high level plan, it is generally acknowledged that it is extremely difficult, if not impossible, for a user or system to produce a complete detailed plan in domains of realistic complexity (Agre and Chapman 1990, Kaebling and Rosenschein 1990). Like Agree and Chapman (Agre and Chapman 1990), we claim that it is not necessary to explicitly tell the robot what to do next in every situation.

Hence, in our system, the plan the user provides the robot is based on a set of goals and hints for reaching them. The communication between the user and the robot is performed in a similar way to explaining how to go from a determined starting point to a precise location, by giving a set of hints like: *"look to your right to find the bookshelf"*.

As we will demonstrate empirically, "hints" are useful for completing the task and are grounded in symbols attached to the behavioral and perceptual abilities of the robot, such as *Avoid*, *Follow Wall* or *Detect*.

The outline of the paper is as follows: Section II describes the robot used for experimentation, and briefly overviews the basic low level Motor-Behaviors we implemented in our system. Section III shows how Motor-Behaviors and their attached attributes will serve as well grounded symbols, and how to select a canonical set of symbols that the user is able to utilize to express its plans based on goals and hints to reach them. We also introduce a new control structure called Conductor-Behaviors, that will serve as the controlling link between the user's plan and Motor-Behaviors. Section IV describes how the plan must be provided by the user, and section V shows the experimental results and the dynamic performance of the robot. Section VI describes related work, and section VII concludes the paper.

## II. System Overview

The mobile's platform control (see Figure 1) is based on differential steering by using relative velocities of two independent wheels. Computation is performed off-board, where

Fig. 1. **Robuter**: the robot is equipped with a sonar sensor array and a stereo vision system. The sonar sensor array consists of a ring of 12 Polaroid ultrasonic transducers. The stereo vision camera system is mounted on a pan-tilt head which gets its control actions through the same serial modem as the mobile platform. Odometric sensors attached to wheels measure the robot's position in the environment.

the robot sends and receives ASCII messages through a serial modem from a host machine. The host computer is a Unix work-station (Sparc10), that is connected via Ethernet to other Unix platforms.

Motor-Behaviors (see Figure 2) get their sensory input from the robot's sensors (sonar array and cameras), and send their control outputs to the robot's actuators (wheels' and turret's pan-tilt controls). Motor-Behaviors are classified in two main sub-systems: Vision and Navigation Motor-Behaviors. Vision Motor-Behaviors (**VMB**s) get images from the cameras, and their motor actions control the movement of a pan-tilt head. Navigation Motor-Behaviors (**NMB**s) actuate on the wheels' controls of the robot, and get their inputs from the sonar sensors, the turret's encoders and 3-D locations from the stereo vision system.

### *A. Navigation Motor-Behaviors*

The navigation architecture is based on a hierarchic structure of behaviors acting in parallel. **NMB**s have more state and knowledge than purely reactive systems; they compute the next action to perform based on the sensor readings and the nearby past, and do not act as mere lookup tables.

Sensor readings are updated using a certainty grid representation (Moravec 1988). How-

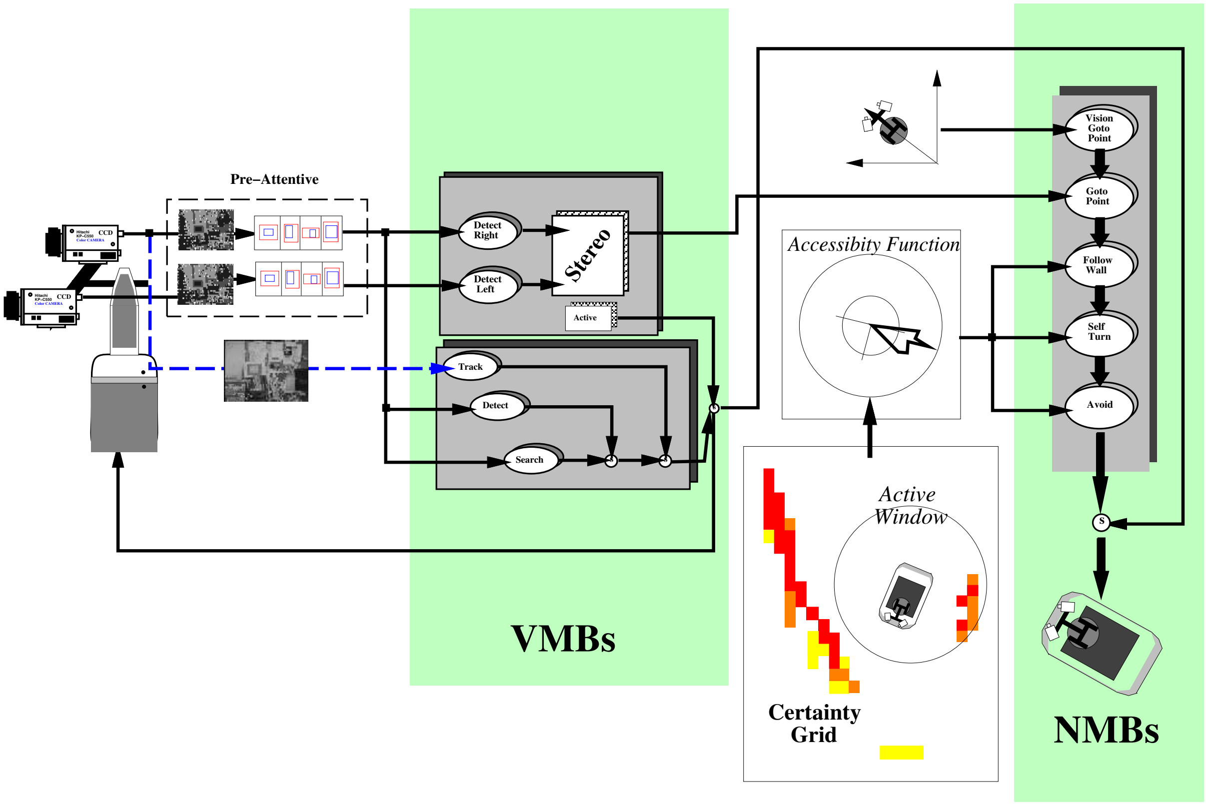



Fig. 2. Control structure of the Motor-Behaviors. Arrows indicate control and information flow.

ever, knowledge of regions located far from the robot is poor due to uncertainties in the measurements provided by sonar and odometric sensors.

In order to cope with these uncertainties, an *Accessibility Function* ($A_F$) is computed, which provides a quantitative representation of the local robot's environment in real time, permitting **NMB**s to take robust and reliable control actions. The $A_F$ provides a quantitative representation of the robot's surrounding space, and has many similarities in its formalism and design with Bohrenstein and Khoren's *Polar Histogram* (Bohrenstein and Koren 1991).

Formally, the $A_F$ is a function that takes the form:

$$A_F(\theta_i) = d_i \tag{1}$$

is referenced in polar coordinates and computed in a local area, defined as the *active window*, centered about the robot's geometric center.

The $A_F$ is defined in the $\theta_i \in [0, 360]$ domain, and $d_i$ takes values in the $[0, 1.1]$ range.

Value "0" indicates the absence of obstacles in the "active window" area, and "1.1" indicates the presence of an obstacle in $\theta_i$ direction inside the security area, defined as the area that the robot covers in omni-directional turns. Values comprised in the $[0, 1]$ are inversely proportional to the distance from the robot's geometric center to an obstacle in a $\theta_i$ direction.

The final goal of the $A_F$ is to provide a real–time and compact representation of the local environment to the Navigation Motor-Behaviors, so that they can output a robust and reliable control action based on this data.

### A.1 Control Structure of **NMB**s

The architecture is implemented as a set of ordered behaviors $B = \{B_1, B_2, \ldots, B_i\}$; each of the behaviors, $B_j$, gets as input a control action $(d_i)$ from a higher level behavior and sends its output $(d_o)$ to a lower level behavior:

$$d_o^j = B_j(d_i^j)$$

The control action that receives $(d_i^j)$ and outputs $(d_o^j)$ each of the behaviors $(B_j)$ is a vector $\vec{v}$:

$$\vec{v} = (s, \omega),$$

where $s$ is the translation speed and $\omega$ is the steering speed.

Figure 2 shows the priority order of **NMB**s. For the sake of clarity, we describe the implemented behaviors in descending priority:

- The task of the **Avoid** behavior is to avoid the obstacles that are in the active window area. It inspects the $A_F$ and generates a control action that is as close as possible to the input control action.
- **Self Turn** behavior was especially designed to avoid dead-locks between the *Avoid* and the *Goto Point* behaviors. It becomes active when it detects a dead-lock situation between the *Avoid* and *Goto Point* behaviors, and aligns the robot with the obstacle.
- The **Follow Wall** behavior inspects the $A_F$ and tries to align the robot to a nearby wall while moving forward.
- **Goto Point** behavior tries to go to an $(x, y)$ environment location given by the Stereo Vision system, utilizing odometric sensors.

• **Vision Goto Point** behavior is active when the vision system is tracking an object, and tries to minimize the angle between the camera's pan position and the robot.

### *B. Vision Motor-Behaviors*

Purposive and qualitative algorithms for artificial vision systems have demonstrated a high degree of robustness and real time response (Aloimonos 1990, Ballard 1991, Horswill 1992). Therefore, Vision Motor-Behaviors (**VMB**s) are specially designed to search, detect, track, and compute the 3-D location of a set of beacons, each consisting of two different colored squares (red, blue or green), one enclosing the other, and their centroids located in the same position. These beacons are attached to real-world landmarks like doors, tables, bookshelfs, etc.

On the left side in Figure 2 the control structure and sensory data flow of **VMB**s is shown. The color images grabbed by the cameras feed the *Pre-Attentive* stage. In this stage, visual features indicating the potential presence of a beacon are extracted. As we are only interested in detecting a fixed set of colored beacons, the color image is segmented into regions of 4-connected pixels, all of them of the same color (Haralick and Shapiro 1992). The result of this stage is a symbolic list containing all the connected colored regions in the image that enclose other regions. This process is performed on 63x47 pixels sized images, at 24 images per second rate.

*Search* and *Detect* behaviors get their sensory inputs from the pre-attentive stage.

• The **Search** behavior stops the turret if there is an element in the pre-attentive stage list. This behavior also remembers the direction where the beacon was last seen, and will focus on that direction, if the beacon was lost by the *Track* behavior.

• The **Detect** behavior will process all the elements in the list provided by the pre-attentive stage. If one of the elements satisfies the geometric and temporal constraints it will be considered as the beacon.

The beacons have to satisfy a set of *geometric constraints*, including that the centroids of both squares are close, and that the aspect ratio of both squares is constrained to a fixed value.

As we are dealing with very small images, *temporal constraints* are applied to avoid digitization errors: if an element that satisfies the geometrical constraints of a beacon is present in the same image location during a set of consecutive frames, than it will be considered a beacon.

Once the beacon is detected the **Track** behavior takes control of the system and tracks it based on a grey-level correlation technique, using a template ($M$) of the beacon that is initially centered on the beacon's position given by the *Detect* behavior. This template is correlated in a constraint search space of the grey level image.

If a new image is acquired at step $t$, we denote $I(u, v, t)$ as the grey level value of a pixel at $(u, v)$, $M(u, v, t-1)$ as the gray level value of the template at position $(u, v)$ at time $t-1$, and $(x, y)$ is defined as the translation in a rectangular constraint area of $I(u, v, t)$: $x_i \leq x \leq x_f$, $y_i \leq y \leq y_f$.

The normalized sum of absolute differences (SAD) between $M(t-1)$ and $I(t)$ is defined as:

$$\sigma(x, y, t) = \frac{\sum_{L_{y_i}}^{L_{y_f}} \sum_{L_{x_i}}^{L_{x_f}} |I(u+x, v+y, t) - M(u, v, t-1)|}{\sum_{L_{y_i}}^{L_{y_f}} \sum_{L_{x_i}}^{L_{x_f}} \Delta u \Delta v} \tag{2}$$

where $(L_{x_i}, L_{y_i})$ and $(L_{x_f}, L_{y_f})$ define the template's upper-left and down-right corners, respectively, and $|I(u+x, v+y, t) - M(u, v, t-1)|$ represents the absolute value of grey-level differences.

The best SAD between the image and the template is determined by:

$$\lambda(t) = min(\sigma(\mathbf{x}, \mathbf{y}, \mathbf{t})); \quad x_i \leq x \leq x_f,\ y_i \leq y \leq y_f, \tag{3}$$

$(x, y)$ denoting the location, in image coordinates, of the best match between the template ($M$) and the Image ($I$).

If the best SAD factor ($\lambda(t)$) is greater than a given fixed value the beacon is considered as lost, and *Search* and *Detect* behaviors resume control of the system. Otherwise, a new template is extracted at the estimated beacon's position $(x, y)$ that is going to be used as correlation mask in step $t+1$.

Using this technique on grey level images enhances the tracking performance to a rate of 38 images per second.

### B.1 Stereo vision system

The *Stereo* vision system determines the 3-D location of the beacon using a calibrated stereo head (Brooks, de Agapito, Huynh and Baumela 1996, Faugeras 1993). Two *Detect* behaviors, each attached to the left and right color images, are used in order to find the corresponding projections of the beacon in both images.

Stereo measurements of the beacon location are gathered in fixed time intervals in which the control actions of the **NMB**s are inhibited, and therefore the robot is stopped. This is due to the limitations of the frame grabber we are using[1], which multiplexes between both cameras in order to get a complete color image. This process takes an average time of one second.

As small sized images are used (63x47 pixels), only beacon's stereo measurements below 4 meters are considered as good by the stereo vision system. The definitive beacon's position used by the *Goto Point* behavior is the mean value of all the computed beacon's positions.

## III. Behaviors and Attributes as Symbols: Symbol Grounding

The problem of attaching meaning to the symbols a physical system employs is often called the problem of symbol grounding (Harnard 1995). In an uncertain world there are difficult problems of maintaining a proper correspondence between the user's world model (and imagination) and the real world and how the robot perceives and interacts in it.

Symbolic systems, such as traditional and most hybrid AI systems, are not grounded in real world symbols (Pfeifer 1996). To ground symbols in the real world it is important to design and implement low level modules that capture the essence of real world acting and sensing, and to further develop higher level modules that use them as input/output resources to interact with the real world (Malcolm and Smithers 1990).

We define a set of symbols, based on the perceptual and action resources of the robot, as communication primitives that the user may use to express long term plans. The above described Motor-Behaviors will serve as basic units of communication, and their attached attributes, that improve and modify the robot's functionalities, will extend the number of

[1] Model S2200 of Data Cell Ltd.

symbols the user may use to communicate more complex plans to the robot.

### A. Attributes

*Attributes* can be defined as local variables that determine the agent's behavior. Not only sensor readings, history, or the present state of the agent will affect the output; attributes of the behavior play a vital role in its performance. Attributes act on sensor readings, how experience is gathered, and how and when state transitions are performed. By changing the behavior's attributes, the subsequent outputs (and behavior) will differ.

Attributes were initially considered as constants, like thresholds on sensor readings, fixed action patterns, or look-up tables, and no further dynamic modification was performed on them (Brooks 1986, Brooks 1989, Flynn, Brooks, Wells and Barret 1989, Connell 1989, Horswill and Brooks 1988). However, the hormone system of the *Behavior Language* (Brooks 1990a) permitted dynamic activation, and some systems where based on the dynamic evolution of internal parameters (Parker 1992). Our goal is to use attributes, and their dynamic modification, to obtain a new set of behavior capabilities that permit the creation of a simple interaction language.

Attributes can be classified according to their granularity or semantic content. *Scalar attributes* are the most basic kind of attributes, and are like registers in the subsumption architecture (Brooks 1986). Attributes of these kind can be thresholds on sensor readings, or, for example, the value in the $[0,1]$ range of the *Accessibility Function* where the *Avoid* behavior considers the obstacle close enough to start the avoidance maneuver. In a higher level we can classify those attributes to which a linguistic label can be attached. To change one of these attributes implies the change of a set of scalar attributes. Hence, the behavior is parametrized by its attributes, and the agent is characterized by the *general* behavior (eg., follow wall, avoid, track, etc.) it performs, and the set of attributes that modify this given behavior. For instance: the *Follow Wall* agent has an attribute designed as *Leave Wall On the "A" Side*. "A" can be either "left" or "right". The intrinsic behavior of the agent does not change (to follow a wall), but the attribute parametrises the behavior, by changing a whole set of different scalar attributes, like turning directions, angles inspected in the $[0,360]$ domain of the *Accessibility Function*, etc.

In the definition of a communication language with the robot it has to be decided the granularity of the message. Some systems (Noreils and Chatila 1995) opted to access and change low level parameters of the control system. This kind of communication is hard and cumbersome for the user, as he has to know all kind of parameters, variables and low level functions, and how the behavior of the system changes when these are modified. One of our main objectives was to define and create a simple interaction language to specify plans, and therefore we opted to use "high level" attributes in the definition of the language. These attributes have a clear meaning for the user, and their modification alter the behavior and goals of the system in a substantial and clear way.

The inspection and modification of behaviors' attributes play a vital role in the overall dynamic performance of the robot, and the ways in which the users can interact with it. Behaviors and their attributes serve as well grounded symbols with which the user communicates plans to the robot.

### *B. Symbols*

TABLE I

Symbols attached to Motor-Behaviors and their Attributes.

<table>
<tr><th>Navigation Motor-Behaviors</th><th>Vision Motor-Behaviors</th></tr>
<tr><td>
<table>
<tr><th>Behavior</th><th>Attribute(s)</th></tr>
<tr><td>Avoid</td><td></td></tr>
<tr><td rowspan="2">Follow Wall</td><td>Left</td></tr>
<tr><td>Right</td></tr>
<tr><td>Goto Point</td><td>Reached</td></tr>
</table>
</td><td>
<table>
<tr><th>Behavior</th><th>Attribute(s)</th></tr>
<tr><td>Track</td><td></td></tr>
<tr><td rowspan="6">Detect</td><td>Blue in Red</td></tr>
<tr><td>Blue in Green</td></tr>
<tr><td>Red in Blue</td></tr>
<tr><td>Red in Green</td></tr>
<tr><td>Green in Red</td></tr>
<tr><td>Green in Blue</td></tr>
<tr><td rowspan="4">Search</td><td>Forward</td></tr>
<tr><td>Left</td></tr>
<tr><td>Right</td></tr>
<tr><td>Everywhere</td></tr>
</table>
</td></tr>
</table>

The implemented Motor-Behaviors and their attributes serve not only as modular units of behavioral generating capability, but also act as abstraction devices enabling the existence of an idealized and tractable representation of the robot's world. It is by means of the effective and reliable capabilities of these behavioral modules that the communication between the robot and the user is securely grounded in the real world. Low level behaviors must first be implemented and then, based on the knowledge of how these work and interact with the environment, higher level aptitudes can be developed, as in purely bottom-up subsumption conception (Brooks 1986). Until it is determined how the system is behaving in the environment, and how the different behaviors work, it is troublesome to communicate with the system in a reliable fashion. The capabilities of the robot determine the terms in which the plans must be expressed, and these competences (behaviors and their attributes) give semantic significance to the user.

Table I shows the canonical set of symbols we have attached to the Motor-Behaviors and their attributes. These constitute the canonical base of actions the robot can perform, and serve as modular units of user–robot communication. In this set of symbols, Motor-Behaviors like *Self Turn*, *Vision Goto Point* and *Stereo* are not included, as they are only implemented to enhance the robot's performance and they do not contribute to the user-robot interaction.

### *C. C-Behaviors*

In order to permit the user to specify plans we propose a new computational structure, called the Conductor-Behaviors (C-Behaviors).

C-Behaviors do not get their inputs from sensors and do not output directly to the robot's motor actuators, but instead they inspect Motor-Behaviors' attributes and modify them[2].

A C-Behavior can be formally described as a simple rule ($CB_i$), with a set of antecedents ($\{A^i_n\}$) and a set of consequents ($\{C^i_m\}$):

[2]In our architecture all behaviors have boolean attributes like "enable" and "active", which are not listed in Table I. The "enable" attribute takes two values (True or False), and by modifying this attribute, C-Behaviors can enable and disable a given behavior. The "active" attribute is read-only; it can not be modified by C-Behaviors, but by inspecting it a C-Behavior can determine if a given behavior is active.

$$CB_i : A^i_0, A^i_1, \ldots, A^i_n \rightarrow C^i_0, C^i_1, \ldots, C^i_m \tag{4}$$

Antecedents are states, in terms of Motor-Behaviors' attributes. Consequents modify the attributes' values of the Motor-Behaviors.

C-Behaviors can be of two different types: **Perpetual** and **Contingent**. Perpetual C-Behaviors activate whenever their preconditions hold; Contingency C-Behaviors activate the first time their preconditions hold, then remain inactive.

C-Behaviors are thus able to determine the "state" of the system by evaluating the behaviors' attributes, and can dynamically alter the goals and performance of the robot, modifying the appropriate attributes. C-Behaviors are not directly connected to the real world, but the symbols they are inspecting and modifying are. A user may define a plan based on hints given to the robot, in terms of the instantiation of sets of C-Behaviors, grounded to the real world.

## IV. User-Robot Interaction: The Plan

The user provides the robot with a Plan ($P$), as a set of different steps: $P = \{S_0, S_1, \ldots, S_j\}$, where a step ($S_i$) is defined as the activation of a set of C-Behaviors: $S_i = \{CB_0, CB_1, \ldots, CB_k\}$. Besides inspecting and modifying Motor-Behaviors' attributes, C-Behaviors can control the flow of the plan, and a given C-Behavior will trigger the next step of the plan.

C-Behaviors act as hints the user provides the robot to fulfill the step's goal. If a Contingency C-Behavior modifies the attribute of a Motor-Behavior this will affect its behavior. However, this attribute does not need to be fixed at that value permanently, and the dynamic performance of the behavior may change it. For instance, a C-Behavior may instruct the *Search* Visual Motor-Behavior to search left. If the target is detected and is being tracked, the *Search* behavior will modify its searching attribute dynamically. If the target is lost the *Search* behavior will try to find the target in the last seen direction.

### A. The Grammar

A very simple grammar (see Figure 3) permits the user to write plans the robot is able to understand. Based on this grammar, the plan ($\boxed{\texttt{<Plan>}}$) the user provides the robot is composed of a set steps (<Step>), each of them containing the description of the C-Behaviors that will monitor the underlying Motor-Behaviors.

```
<Plan>             = <Step>{<Step>}.
<Step>             = "BEGIN" "n" <C-Behavior>{<C-Behavior>} "END".
<C-Behavior>       = <C-Behavior Type> "{" "i" <Cond>{<Cond>}
                                          "j" <Cons>{<Cons>} "}".
<C-Behavior Type> = "ONCE"|"IF".
<Cond>             = <Flag>|<Attribute Quest>
<Flag>             = "TRUE" | "REACHED" | "TRACKING".
<Attribute Quest> = <Quest><Behavior>.
<Quest>            = "ENABLED" | "NOT_ENABLED".
<Cons>             = {<Mod><Behavior> "n" {<Attribute>}}
                    | {"NEXTSTATE =" "m"}.
<Mod>              = "ENABLE"| "DISABLE".
```

Fig. 3. Language grammar in BNF syntax.

Each step in the plan is delimited by the tokens BEGIN and END. The number n, on the right side of the BEGIN token indicates the step's number in the plan.

All the C-Behaviors(<C-Behavior>) of the step are either perpetual (IF) or contingency (ONCE).

In each C-Behavior the number of pre-conditions i and post-conditions j has to be explicitly indicated.

- The pre-condition (<Cond>) of a C-Behavior may be either TRUE, a <Flag>, or the inspection of a MB's attribute: (<Attribute Quest>).

– If the condition is TRUE it implies the absence of conditions to check in order to apply the post-conditions. This is useful to initiate Motor-Behaviors at the beginning of each step.

– "Flags" indicate the state of MBs and their attributes. Hence, if the REACHED flag is settled to true, it implies that the *Goto Point* behavior has reached its goal.

– TRACKING indicates if the **VMB**s are tracking the beacon.

– The inspection (<Quest>) of a behavior's attribute (<Behavior>) is constraint to determine if the behavior is ENABLED or NOT_ENABLED.

• The consequent (<Cons>) of a C-Behavior can either ENABLE or DISABLE a Motor-Behavior.

If a Motor-Behavior is activated by using the token ENABLE, the number of attributes n and the symbol <Attribute> associated to the Motor-Behavior must be indicated, according to the symbols shown in Table I.

One of the C-Behaviors of the step is in charge of verifying if the goal has been reached. The token NEXTSTATE is used to determine the next step (m) of the plan.

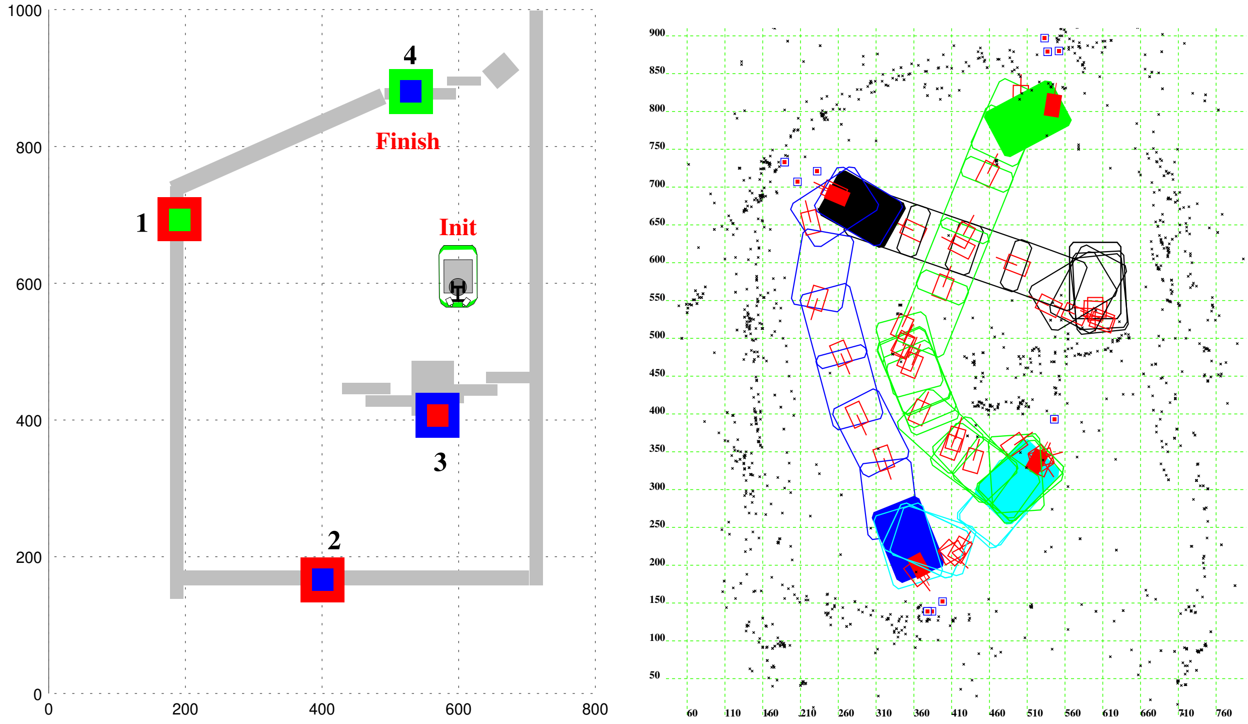
### *B. Achieving Common-sense through C-Behaviors*

Besides the knowledge about behaviors and their attributes that serve as symbols, the user should roughly know about the internals of how the system is built and how the different behaviors interact, in order to write a correct plan. For example, the way Motor-Behaviors are built force a clear distinction between the *Follow Wall* and *Avoid* behaviors: if both behaviors are active an emergent behavior appears that interferes with the performance of the *Follow Wall* behavior.

But we are striving for communication that is as independent of the internal control as possible. Therefore, we propose a computational structure that permits the user think that if the plan indicates the robot to go to the location of a certain beacon, its "common-sense" will be "responsible" for taking care of possible obstacles that are in its way to the target.

In order free the user from (1) writing plans that take care of the correct arbitration of behaviors and (2) to provide an explicit and complete set of actions (in terms of behaviors and their attributes) to be performed once certain conditions are met, we have included a set of static C-Behaviors called Common-sense C-Behaviors.

Common-sense C-Behaviors are built-in Perpetual C-Behaviors that permit the user assumption that the robot has a certain common-sense. Convenient rules, like obstacle avoidance if the robot is moving towards a goal, or the disabling of the *Follow Wall* behavior if the *Avoid* behavior is active, are controlled by the robot's common-sense, and therefore Common-sense C-Behaviors permit a more flexible interaction between the




(a) Scenario (b) Complete Path

Fig. 4. Scenario and performance. Numbers in the X and Y axis denote centimeters.

user and the robot. Hence, the user is no longer responsible for instantiating complex C-Behaviors that monitor all the aspects involved in the robot's architecture.

## V. Experimental Results

In this section we describe the experiments that were carried out to analyze the dynamic performance of the robot. In particular, we are interested in assessing how the performance of the robot is enhanced when some hints are added to the plan provided to the robot to perform a certain task, based on the user's knowledge of the environment and on the robot's perceptual capacities. Many experiments envolving different plans and different types of environments were carried out, one of which is described here.

Figure 4(a) shows an outline of the laboratory where the experiment was performed. The objects present in this space (robot, walls and obstacles) have been drawn to scale.

The numbers show the location of the goal for each step of the plan provided to the robot and coincide with the location of the different visual beacons. The plan the robot had to achieve was to reach the location of the four different visual beacons in a sequential order. A complete transcription of the plan provided by the user can be found in Figure 5.

As it may be observed, the plan consisted of five steps, with the last step being empty, as the robot was required to stop at the location of the last beacon. Each step also includes a number of hints provided by the user to improve the performance of the robot in achieving each partial goal.

Let us focus now on the first step. The user's hint was to search for the beacon composed of a green square inside a red one to the right hand side of the straight ahead direction:

```
ONCE { 1 TRUE 2
  ENABLE Detect 1 Green in Red
  ENABLE Search 1 RIGHT
}
```

Once the beacon was detected, the Common-sense C-Behaviors activated the *Goto Point*, *Avoid*, *Self Turn* and *Vision Goto Point* behaviors, implicit and necessary to achieve the goal. In its way to the beacon, the robot stopped at fixed intervals of time to compute the 3D location of the beacon using the stereo vision system. If the computed distance was found to be under four meters, the measurement was considered to be sufficiently precise by the *Goto Point* behavior, and the robot advanced towards that position. If, on the contrary, the beacon was detected to be lying further than four meters away, the *Vision Goto Point* behavior would continue in control of the system, aligning the robot's body with the direction of the turret.

Once the robot reached the beacon (a distance less than 110 cm from the rear axis), step number two took control of the robot. In this occasion, the hint to accomplish the goal of reaching the red in blue beacon was to follow the wall on the right side and look for the beacon in the forward direction, as can be observed in the outline of the plan shown in Figure 5.

Figure 4(b) shows the complete trajectory followed by the robot to achieve all four steps of the plan. The position and orientation of the robot and cameras, the sensor measurements (small crosses) and the stereo measurements (small boxes) considered as good (distance below 4 meters), are shown. Shaded robots indicate the robot's final

**Step 1**

```
BEGIN 1
ONCE { 1 TRUE 2
  ENABLE Detect 1 Green in Red
  ENABLE Search 1 RIGHT
}

IF { 1 REACHED 1
  NEXTSTATE 2
}
END
```

**Step 2**

```
BEGIN 2
ONCE { 1 TRUE 3
  ENABLE FollowWall 1 RIGHT
  ENABLE Detect 1 Red in Blue
  ENABLE Search 1 FORWARD
}

IF { 1 REACHED 1
  NEXTSTATE 3
}
END
```

**Step 3**

```
BEGIN 3
ONCE { 1 TRUE 3
  ENABLE FollowWall 1 RIGHT
  ENABLE Detect 1 Blue in Red
  ENABLE Search 1 LEFT
}

IF { 1 REACHED 1
  NEXTSTATE 4
}
END
```

**Step 4**

```
BEGIN 4
ONCE {1 TRUE 3
  ENABLE FollowWall 1 RIGHT
  ENABLE Detect 1 Red in Green
  ENABLE Search 1 RIGHT
}

IF { 1 REACHED 1
  NEXTSTATE 5
}
END
```

**Step 5**

```
BEGIN 5
END
```

Fig. 5. The Plan.

position and orientation once the step was accomplished.

Forty experiments were performed in the same scenario and the robot starting in the same position. In all the experiments the robot always completed successfully the plan. The robot finished the plan in almost the same time, and we computed a standard deviation of 15 seconds (see Table II for a more detailed exposition of time performance in one of the trials). We are convinced of the fact that the success rate was due to the careful and systematic implementation of Motor-Behaviors and the effectiveness of the user provided hints.

### *A. Effectiveness of user provided hints*

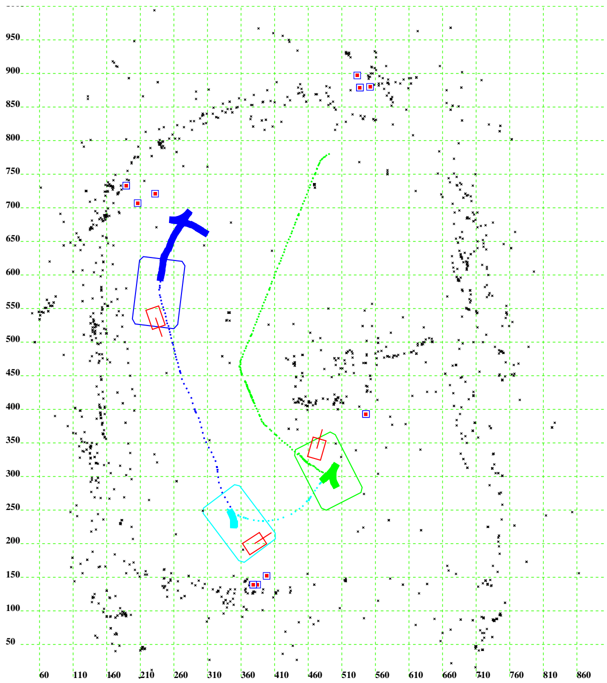



Fig. 6. Effectiveness of hints.

In this subsection we will show how the hints provided by the user were effectively used by the robot in its trajectory, consequently enhancing its behavior. The behavior of the robot following the hints is shown in Figure 6.

All the hints given to the robot in the plan (see Figure 5) consisted on following the wall to its right or left side, and on searching for the beacon to the right or to the left of the straight ahead direction.

Bold lines in Figure 6 show the path followed by the robot when the *Follow Wall* behavior was active while the vision *Search* behavior was looking for the beacon in the direction provided by the user's hint.

The position and orientation of the robot and the camera at the moment when the beacon is detected and the tracking begins has been depicted in Figure 6. At that stage, the user's hint to follow the wall is no longer useful/applicable to the robot. Therefore, a Common-sense C-Behavior monitored this transition enabling the *Avoid*, *Goto Point* and *Vision Goto Point* behaviors, and disabling the *Follow Wall* behavior:

```
IF { 1 TRACKING 4
   ENABLE Avoid 0
   ENABLE GotoPoint 0
   ENABLE VisionGotoPoint 0
   DISABLE FollowWall 0 }
```

Dotted lines show the path from the location where the system recognized and started to track the beacon, to the beacon's location. Small crosses show the sonar sensor measurements, and small boxes the stereo vision computed distances of each of the beacons.

### *B. Comparative study of performance with and without hints*

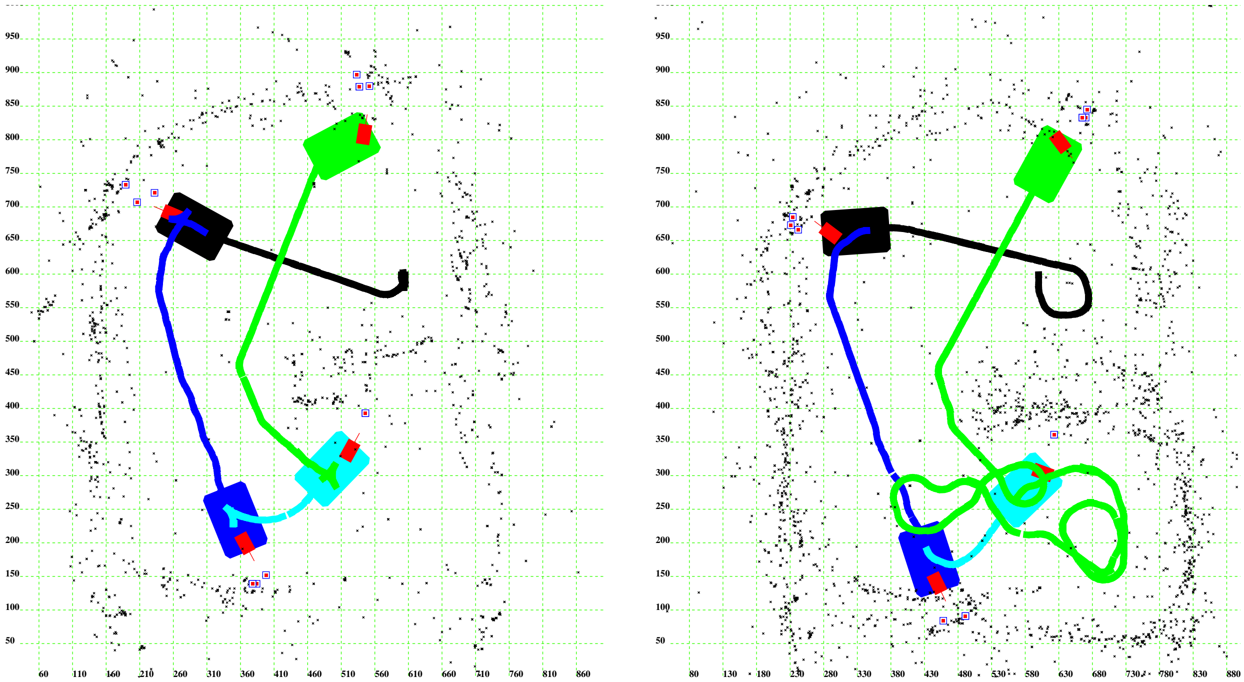
TABLE II

SECONDS SPENT FOR EACH OF THE STEPS.

| Step | Plan with hints | Plan without hints |
|---|---|---|
| 1 | 80 sec. | 100 sec. |
| 2 | 66 sec. | 68 sec. |
| 3 | 51 sec. | 45 sec. |
| 4 | 173 sec. | 255 sec. |
| TOTAL | 370 sec. | 467 sec. |

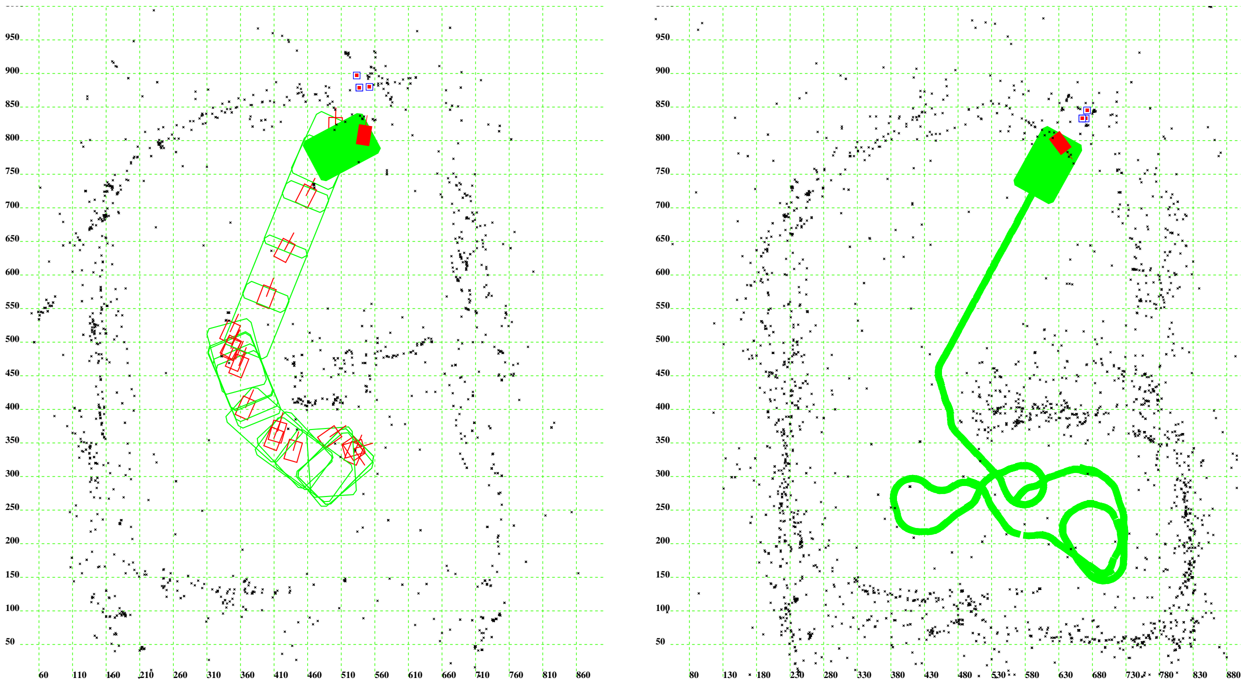
In order to study how hints improve the performance of the robot, the following experiment was carried out. The robot was provided with two different plans to achieve the same goals in the same scenario: one included several hints provided by the user and the

**Complete Path**



(a) Performance **with** hints. (b) Performance **without** hints.

**Step 4**



(c) Performance **with** hints. (d) Performance **without** hints.

Fig. 7. Comparison between plans with and without hints.

other did not include any hints.

Each of the steps in the plan without hints took a form of the kind:

```
BEGIN <step number>
ONCE { 1 TRUE 1
  ENABLE Detect 1 <beacon type>
}
IF { 1 REACHED 1
  NEXTSTATE <next step>
}
END
```

where each step (`<step number>`) only contained the nature of the visual beacon to be detected (`<beacon type>`), and the number of the next step in the plan (`<next step>`).

Table II shows the time performance for each of the steps in both plans. Surprisingly, similar times were expended in achieving the first three steps in both cases. However, it may be readily observed that a much longer time was needed to reach the location of the fourth beacon when no hints were provided in the plan.

The success in achieving the first three steps in almost the same time with both plans may be due to random luck and to the particular characteristics of the environment as there appeared to be a clear path from the robot's position to the next target. However, in the last step, the obstacles present in the room prevented the cameras from visually detecting the beacon with ease, causing the robot to move about the environment to look for the beacon.

Figure 7 shows a complete comparison of the dynamic performance of the robot with both types of plans. Figure 7(a) shows the complete path followed by the robot when the plan **with** hints was provided, while Figure 7(b) shows the trajectory achieved with the plan **without** hints. Figure 7(c) depicts the succesive positions and orientations of the robot and the cameras in step four, with the first plan, and similarly Figure 7(d) shows the path followed in this step with the plan without hints. The smooth path followed by the robot using hints highly contrasts with the path followed by the robot when no hints were provided.

## VI. Related Work

Behavior selection and plan execution has been actively analyzed short after reactive architectures were proposed by Brooks (Arkin 1989a, Maes 1989, Connell 1991, Malcolm and Smithers 1990, Saffioti, Ruspini and Konolige 1993, Payton 1990, Hayes-Roth 1993). From the large amount of research performed on this subject, our review in this section focuses on actual robots whose architecture is based on the hybrid paradigm, proposed algorithms for action selection, and robots that use visual perception like our robot.

A clear example of a hybrid architecture is the SSS system implemented on the TJ robot (Connell 1991). This hybrid system combines a servo-control layer, a "subsumption" layer, and a symbolic layer, and *event detectors* link the subsumption and the symbolic layers. Like our approach, this system also has the ability to turn each behavior on or off selectively, and has the ability to parametrize certain modules. However, the symbolic and reactive layers are clearly separable, and they are linked by using "contingency tables".

In the same way we ground symbols through behaviors, the assembly system SOMASS (Malcolm and Smithers 1990) grounds its automated planner through a single hierarchy of behavioral modules instead of a twin hierarchy of sensing and action. This system requires little more information than the shape of the desired assembly and the shape of the parts from which to construct an assembly plan, using a PROLOG assembly planning system. However, the SOMASS system does not plan with the help of a set of hints but, instead, the final shape of the structure to be built is given as symbolic input. Then, the planner decides the best plan to achieve the goal of building the structure using behaviors as symbols.

Related to our approach, Firby (1987) presents a system that makes use of higher level primitives that take care of, and hide, low level control details. More specifically, the 3T architecture (Bonasso, Firby, Gat, Kortenkamp, Miller and Slack 1997) decomposes the problem of robot control into three interacting functional modules such as planning, sensor processing and execution. The architecture is composed of: a dynamically reprogrammable set of reactive skills, which represent the architecture's connection with the world. These skills, like the Motor Behaviors in our architecture, are networked by having the outputs of one skill connected to the outputs of another skill. Similar to our

C-Behaviors, the 3T architecture uses Reactive Action Packages (RAPs) that activate and deactivate sets of skills. A RAP is a description of how to accomplish a task in the world using discrete steps. One of the main differences with our system consists in that instead of providing a set of goals and hints, that permit the robot a certain freedom in accomplishing its tasks, the wait-for clauses, the RAP interpreter uses, block the task execution until a *reply* is received by the skill manager. These *replies* are produced by special skills called *events*. The sequencer in the 3T architecture uses events to determine when a particular set of skills has completed its work and when the world has changed, whereby our system inspects the state of the system and the world via the attributes of the Motor-Behaviors. Finally, a human supervisor assigns the robot a set of tasks that a deliberative planner synthesizes into a partially ordered list of tasks.

The 3T architecture has been successfully applied to different embodied robots. At the MITRE Autonomous System's Laboratory this architecture has been used to program a robot (equipped with sonar sensors and a laser range finder with bar-coded tags for landmark recognition) to navigate in an office environment.

Like in our architecture, the upper layers of the layered control architecture of the robot Xavier (Simmons, Goodwin, Haigh, Koenig and Sullivan 1997) provide guidance to the lower layers, while the latter handle all the details and events that the upper layers abstract away. However, this system uses a task scheduling based on a symbolic planning architecture (Haigh and Veloso 1997), and deliberative and reactive control is provided by a general purpose software framework called *Task Control Architecture* (TCA) (Simmons 1994), which is more like a robot operating system. The TCA generates a task net, where each node in the net can be either decomposed into other nets or, like a Motor-Behavior, is an interface to the robot. Finally, the communication interface of this robot hides all the low level control details and uses the World Wide Web, which allows to specify one of 42 different tasks, like delivering a restricted set of messages and monitoring offices.

Instead of using behaviors and their attributes as symbols the robot *Flakey* (Saffiotti et al. 1997) uses complex behaviors (Saffioti et al. 1993) at the task level which are sequenced and their progress is monitored through events in the *local perceptual space (LPS)*, an egocentric symbolic representation of the local space where information from different sensor modalities, and different levels of abstraction and complexity, freely mix.

Noreils and Chatila (1995) integrate vision and navigation and propose an architecture that consists of three basic levels: planning, control and functional. The control system controls the functional level, enabling the robot to execute missions (plans) expressed in a command language. More than a high–level interaction language, this language is based on the precise and detailed instantiation of low level attributes, like the range of the ultrasound sensors, turning angles, etc.

An algorithm for action selection among agents with different goals has been studied by Maes (1989). She proposes an algorithm, tested on simulators, that provides global parameters, which can be used to tune the action selection behavior of an agent along several criteria, such as goal orientedness versus situation orientedness.

Integration of visual perception and navigation is also addressed by Košecká, Bajcsy and Mintz (1994). They propose a hybrid system, but they investigate the control-theoretic properties of the system. Their formalism has been tested through experiments with one agent performing the task of avoiding an obstacle while tracking a target.

Horswill (1993) integrated visual processing, navigation and action selection of low level behaviors, in the robot Polly. This robot provides visitors with tours of a floor in an office building, using a digital camera as its main sensory source. Polly waits for a visitor asking for a tour in the indoor environment; During the tour, the robot explains the visitor where different locations are, as they pass along them (e.g, “This is Ian’s office”), but the goals of the robot can not be modified by a user. However, the emphasis of his work is to demonstrate that a sequence of transformations derive a specialized system from a more general one (Horswill 1995).

Developers of symbolic planning systems are also seeking to incorporate “advice-taking” mechanisms into their systems. The *Advisable Planner* (AP) project (Myers 1996) accepts a variety of instructions and advise from a user, and employ them to guide the generation of a plan. In a very different approach as in our system, the AP is a classical top-down system, where the user advice is translated into a normalized advice that the system is able to understand, which is compiled into plan constraints. In clear contrast with this approach, the contruction of our system was based on a bottom-up philosophy, where advise (in terms of hints) is based directly on the previously developed perceptual and behavioral abilities of the robot. In a similar symbolic and top-down implementation fashion, the ACT (Wilkins and Myers 1995) formalism is a language that tries to get planning and execution systems to speak a common language by generating complex plans and the reactive execution of these plans in dynamic environments. In clear contrast with our system, ACT addresses problems in domains such as military operations, which require that the planner exercises greater influence over the execution system.

## VII. Conclusions

In order the robot reliably accomplishes the goals stated in the plan, we propose to ground the the user's plan in the robot's motor and perceptual abilities. However, in contrast to traditional hybrid systems (Connell 1991, Malcolm and Smithers 1990, Simmons et al. 1997, Bonasso et al. 1997), we do not propose a twin hierarchy of a symbolic planer and a reactive sub-system. Therefore three main issues have been addressed in this paper: 1) scalability of behavior based systems, 2) smooth transition between the reactive layer and the plan, and 3) how to provide a flexible plan to the robot.

To solve the scalability problem we insist on preserving the philosophy of incremental bottom–up implementation of perception/action capabilities of the robot (Brooks 1991), and have proposed the use of immediate planning behaviors as a substrate to generate a canonical set of well grounded symbols the user may utilize to express plans. The process of long term planning is left to the user, but the smooth transition between low-level modules and the structures that control them (C-Behaviors) mean that the plan is based on the perception and action resources of the robot.

A smooth transition is obtained, and as Mataric (1992), the plan acts as a set of reactive behaviors that control basic motor behaviors. However, in our system, C-Behaviors simply act as flexible control rules that monitor Motor-Behaviors' attributes. By using the behaviors' attributes, we are able to define plans based on symbols attached to them, and augment the basic functionalities of the robot by simply modifying its behavior through the modification of the attributes.

It is extremely difficult for a user to produce a complete detailed plan, but hints enhance the robot's abilities, and give a certain kind of freedom to the robot once they are no longer useful. There are tasks and environments where a symbolic planner is not necessary, and in our system we only provide the robot with a plan that consists of a set of goals in terms of locations in the environment the robot has to reach in a given order. However, and as Figure 7(b) demonstrates, by only instructing the robot what goals to reach does not guarantee the success of the mission. Therefore, we have added hints to the plan. These hints are based on the perceptual and behavioral capabilities of the robot, like looking for the beacon in a given direction, or aligning to a nearby wall. Hence, hints are based on perceptual references of the environment that the robot and the user can reliably and precisely perceive. References like "turn around after reaching the goal" might be ambiguous and not reliable, as it is hard to determine beforehand the relative position of the robot when it reaches the goal. However, walls in indoor environments are stable in time (they do not change very often), can be reliably detected and are therefore not ambiguous for the user and the robot.

Hints are also intended to tune the system's behavior for the time they are useful. Figure 6 shows the path followed by the robot when the hints were active. The inner dynamics of the system, like tracking and moving towards the beacon once it has been detected, establish when hints are not useful anymore. For instance, if the beacon is lost by the **VMB**s the hint of looking for it on a given direction does not apply anymore. Instead, the *Search* **VMB** tries to find the beacon in the direction where it was last seen.

Experimental results show how hints increase the robot's performance, in terms of time spent pursuing each of the goals stated in the user's plan, versus a completely reactive plan. The careful and incremental implementation of Motor-Behaviors permitted the achievement of a high degree of robustness and real time response to changes in the environment.

Future work focuses on creating new types of C-Behaviors that deal both with time and combinations of symbols. For instance, we may attach a symbol to a "go through the door" behavior, that modifies a set of different attributes attached to several behaviors. We are also interested in error recovery, due to errors produced by the internal dynamics of the system, or by a bad user's plan. Either the case, the robot should be able to detect "error" situations, report the error and ask the user for advice.

## Acknowledgments

The author is grateful to Lourdes de Agapito, for discussions about the ideas and implementation details of this work, as well as for suggestions and corrections on earlier versions of this paper.

Many thanks to Maja Matarić, for providing patient, expert and useful advise, and for comments on earlier drafts of this paper which have helped to improve it. The author also wishes to thank Mike Brooks, Jorge Gasós and the anonymous reviewers for useful comments.

The work described in this paper was partially supported by the Spanish Ministry of Science under project I.Ares: CICYT, TAP 816193-C03-2.

## References

Agre, P. E. and Chapman, D.: 1990, What are plans for?, *Robotics and Autonomous Systems* **6**, 17–34.

Aloimonos, J.: 1990, Purposive and qualitative active vision, *DARPA Image Understanding Workshop*, pp. 816–828.

Arkin, R. C.: 1989a, Integrating behavioral, perceptual and world knowledge in reactive navigation, *in* P. Maes (ed.), *Designing Autonomous Agents*.

Arkin, R. C.: 1989b, Towards the unification of navigational planning and reactive control, *AAAI Spring Symposium on Robot Navigation*, pp. 1–5.

Ballard, D. H.: 1991, Animate vision, *Artificial Intelligence* **48**(1), 57–86.

Bohrenstein, J. and Koren, Y.: 1991, The vector field histogram-fast obstacle avoidance for mobile robots, *IEEE Transactions on Robotics and Automation* **7**(3), 278–288.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. and Slack, M.: 1997, Experiences with an architecture for intelligent, reactive agents, *Journal of Experimental and Theoretical Artificial Intelligence* **9**(1).

Brooks, M. J., de Agapito, L., Huynh, D. Q. and Baumela, L.: 1996, Direct Methods for Self-Calibration of a Moving Stereo Head, *Proceedings ECCV'96*, Vol. 2, Cambridge, UK, pp. 415–426.

Brooks, R. A.: 1986, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **RA-2**(1), 14–23.

Brooks, R. A.: 1989, A robot that walks; emergent behavior from a carefully evolved network, *Neural Computation* **1**(2), 253–262.

Brooks, R. A.: 1990a, The behavior language; user's guide, *A.I. Memo 1227*, M.I.T.

Brooks, R. A.: 1990b, Elephants don't play chess, *Robotics and Autonomous Systems* **6**, 3–15.

Brooks, R. A.: 1991, Intelligence without representation, *Artificial Intelligence* **47**, 139–159.

Connell, J. H.: 1989, A colony architecture for an artificial creature, *M.I.T. A.I. Lab Tech Report 1151*, M. I. T. Artificial Intelligence Laboratory.

Connell, J. H.: 1991, SSS: A hybrid architecture applied to robot navigation, *IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2719–2724.

Elfes, A.: 1986, A distributed control architecture for an autonomous mobile robot, *Artificial Intelligence* **1**(2), 135–144.

Faugeras, O.: 1993, *Three-dimensional Computer Vision: A Geometric Viewpoint*, MIT Press.

Fikes, R. E. and Nilsson, N. J.: 1971, STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**, 189–208.

Firby, R. J.: 1987, An investigation into reactive planning in complex domains, *Proceedings, Sixth National Conference on Artificial Intelligence*, Seattle, pp. 202–206.

Flynn, A. M., Brooks, R. A., Wells, W. M. and Barret, D. S.: 1989, The world's largest one cubic inch robot, *Proceedings IEEE Micro Electro Mechanical Systems*, IEEE, pp. 98–101.

Haigh, K. and Veloso, M.: 1997, High-level planning and low level-level execution: Towards a complete robotic agent, *Proceedings of the International conference on Autonomous Agents (AA)*.

Haralick, R. M. and Shapiro, L. G.: 1992, *Computer and Robot Vision*, Vol. 1, Addison Wesley.

Harnard, S.: 1995, Grounding symbolic capacity in robotic capacity, *in* L. Steels and R. Brooks (eds), *The "artificial life" route to "artificial intelligence"*, New Haven: Lawrence Erlbaum.

Hayes-Roth, B.: 1993, Opportunistic control of action in intelligent agents, *IEEE Transactions on Systems, Man, and Cybernetic* **23**(6), 1575–1587.

Horswill, I. D.: 1992, Characterizing adaptation by constraint, *Toward a practice of autonomous systems. Proceedings of the First European Conference on Artificial Life*, MIT Press/Bradford Books, pp. 58–63.

Horswill, I. D.: 1993, *Specialitation of Perceptual Processes*, PhD thesis, MIT Artificial Intelligence Laboratory.

Horswill, I. D.: 1995, Analysis of adaptation and environment, *Artificial Intelligence* **73**(1-2), 1–30.

Horswill, I. D. and Brooks, R. A.: 1988, Situated vision in a dynamic world: Chasing objects, *AAAI-88* pp. 796–800.

Kaebling, L. P. and Rosenschein, S. J.: 1990, Action and planning in embedded agents, *Robotics and Autonomous Systems* **6**, 35–48.

Košecká, J., Bajcsy, R. and Mintz, M.: 1994, Control of visually guided behaviors, *GRASP Lab. Technical Report 367*, Department of Computer and Infomation Science, University of Pennsylvania.

Lozano-Pérez, T.: 1983, Spatial planning: A configuration space approach, *IEEE Transactions on Computers* **C-32**(2), 395–407.

Maes, P.: 1989, How to do the right thing, *Connection Science Journal* **1**(3), 291–323.

Malcolm, C. and Smithers, T.: 1990, Symbol grounding via a hybrid architecture in an autonomous assembly system, *Robotics and Autonomous Systems* **6**, 123–144.

Mataric, M. J.: 1992, Integration of representation into goal-driven behavior-based robots, *IEEE Transactions on Robotics and Automation* **8**(3), 304–312.

Moravec, H. P.: 1977, Towards automatic visual obstacle avoidance, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)* p. 584.

Moravec, H. P.: 1988, Sensor fusion in certainty grids for mobile robots, *AI Mag.* pp. 61–74.

Myers, K. L.: 1996, Advisable planning systems, *in* A. Tate (ed.), *Advanced Planning Technology*, AAAI Press, Menlo Park, CA.

Noreils, F. R. and Chatila, R. G.: 1995, Plan execution monitoring and control architecture for mobile robots, *IEEE Journal of Robotics and Automatition* **11**(2), 255–266.

Parker, L. E.: 1992, Adaptive action selection for cooperative agent teams, *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pp. 442–450.

Payton, D. W.: 1990, Internalized plans: A representation for action resources, *Robotics and Autonomous Systems* **6**, 89–103.

Pfeifer, R.: 1996, Symbols, patterns, and behavior: towards a new understanding of intelligence, *Proc. Japanese Conference on Artificial Intelligence*, Tokyo, pp. 1–15.

Saffioti, A., Ruspini, E. H. and Konolige, K.: 1993, A fuzzy controller for flakey, an autonomous mobile robot, *Technical Note 529*, Artificial Intelligence Center. SRI International, 333 Ravenswood Ave. Menlo Park, CA 94025, USA.

Saffiotti, A., Konolige, K., Myers, K. and Ruspini, E. H.: 1997, The saphira architecture: A design for autonomy, *Journal of Experimental and Theoretical Artificial Intelligence (JETAI). Special issue on Architecures for Physical Agents* **9**(1), 215–235.

Schneider-Fontán, M.: 1996, *Planning based on active perception*, PhD thesis, Universidad Complutense Madrid, Facultad CC. Fisicas.

Simmons, R.: 1994, Structured control for autonomous robots, *IEEE Transactions on robotics and Automation* **10**(1), 34–43.

Simmons, R., Goodwin, R., Haigh, K., Koenig, S. and Sullivan, J.: 1997, A layered architecture for office delivery robots, *First International Conference on Autonomous Agents*, Marina del Rey, CA, pp. 245–242.

Thomson, A. M.: 1977, The navigation system of the JPL robot, *Proceedings of the International Joint Conference on Artificial Inteligence*, pp. 749–757.

Wilkins, D. E. and Myers, K. L.: 1995, A common knowledge representation for plan generation and reactive execution, *Journal of Logic and Computation* **5**, 731–761.