A Complete Control Structure for a Visually Driven Autonomous Robot

Miguel Schneider-Fontán
Volen Center for Complex Systems
Computer Science Department
Brandeis University
Waltham, Ma 02254
miguel@cs.brandeis.edu

Abstract

In this paper we present a complete control architecture for a mobile robot that will work in an indoor semi-structured environment. The goals the robot has to reach are a set of visual colored beacons attached to real world objects. The robot is able to detect and track these objects, compute the distance to them, and move towards them while coping with unexpected events that may occur in the environment.

Our main goals have been to construct a complete control systems that (1) is able to react in real time to events and (2) accomplishes the steps of a plan provided by a user. This user plan is based on the perceptual and behavioral abilities of the robot and is flexible in the sense that it provides a set of hints on how to get to the goals instead of a rigid and unyielding list of commands to be carried out.

Keywords

Autonomous robots, sensor processing, behavior based systems, plan execution, user-robot communication.

I. Introduction

The concept of "autonomous robot" is usually applied to any mobile controlled system that is able to perceive the environment with its sensors and to interact with it without any external control. We have to admit that complete autonomy is far from being reached, and that a general-purpose and robust autonomous robot has yet to be built. However, certain environments, like indoor environments, have desirable visual properties that can be exploited to enable robust robot performance. These properties can be exploited via low level control primitives [1] or, as in our case, by providing the robot with a plan consisting on a set of visual goals and some hints on how to reach them, based on the user's knowledge about both the environment and the robot's capabilities.

To achieve the main goal of building a system that is capable of accomplishing the tasks instructed in a plan, we first developed, implemented and debugged all the basic behavioral and perceptual abilities of the robot, based on a behavior-based control philosophy [2]. These basic abilities, which we call Motor-Behaviors, form in themselves an "immediate planning system" [3].

Behavior based systems are based on the paradigm of "divide and conquer"; the control structure being a hierarchical set of interdependent modules which are executed in parallel and which only use the information that is relevant to their specific task. For instance, an obstacle avoidance module does not need to know the color of the object the vision system is tracking (see Figure 1 for a graphical explanation based on actual data recorded from our robot performing the task of approaching a beacon while avoiding obstacles). This should only concern the object-tracking module. Behaviors take control of the system when they perceive that they should. Hence, when a collision is imminent, all other activities of the robot are more or less irrelevant, and the most important event the robot has to face is the obstacle avoidance.

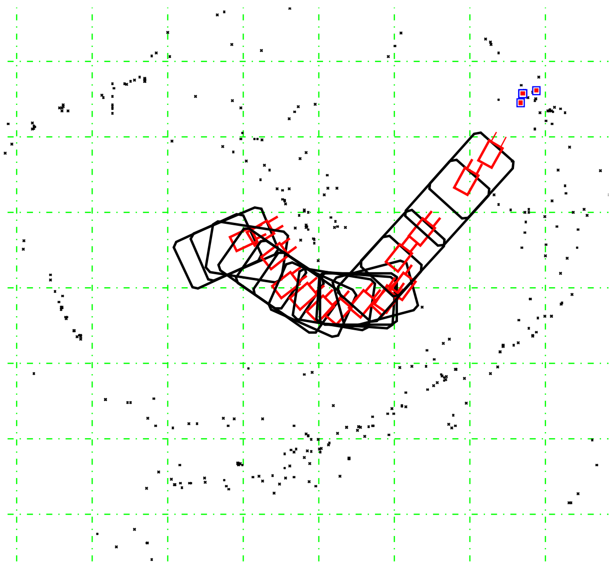


Fig. 1. **Interaction among behaviors:** Initially, the robot moves towards the beacon guided by the *Vision Goto Point* behavior. When an obstacle obstructs its path to the goal, the *Avoid* behavior takes control of the system and generates the control actions that avoid the obstacle, while, in parallel, the vision system is still tracking the beacon and performs 3-D stereo measurements of the beacon's location. Once the obstacle is fully avoided, the robot moves in a straight line to the goal, guided by the *Goto Point* behavior.

Behavior based vision systems are mainly purposive and are built specifically to cope with the task they are intended for [4], [5]. Our system is able to reliably search, detect, track and determine the 3-D location of simple colored beacons attached to real world objects. We believe that building a specific vision system for an autonomous robot embedded in a given environment, permits the successful completion of the directives instructed in a plan provided by the user.

Hints

When people encounter a foreign environment they usually ask for advice on how to get to their destination. Based on the fact that Motor-Behaviors form in themselves an "immediate planning system", in our system we have tried to simulate the way people advise others on how to get to a certain location: People use hints like, "once you pass the grocery store turn to the right". You do not need to tell anybody how to walk to get to the store, or how to avoid other people in their way. Therefore, in our system, the plan the user provides the robot is based on a set of goals and hints on how to reach them. The communication between the user and the robot is similar to the way we explain 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"*. These "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*.

II. System Overview

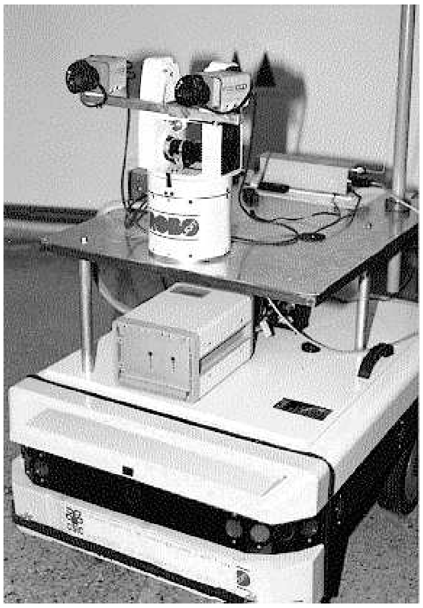


Fig. 2. **Robuter**: the robot is equipped with a sonar sensor array and a stereo vision system. The sonar sensor array consists of a ring of 12 Polaroid ultrasonic transducers. The stereo vision camera system is mounted on a pan-tilt head which gets its control actions through the same serial modem as the mobile robot's platform. Odometric sensors attached to wheels measure the robot's position in the environment.

The mobile robot's platform control (see Figure 2) is based on differential steering by using relative velocities of two independent wheels. Computation is performed off-board; 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 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 (Figure 5) get images from the cameras, and their motor actions control the movement of a pan-tilt head. Navigation Motor-Behaviors (Figure 3) get their inputs from the sonar sensors, and actuate the wheel controls of the robot.

A. Navigation Motor-Behaviors

The navigation architecture is based on a hierarchic structure of behaviors acting in parallel. Navigation Motor-Behaviors have more state and knowledge than purely reactive systems; they compute the next action to perform based on the sensor readings and the recent past.

The large dimensions of our robot, and the limited number of sonar sensors with which it is equipped,

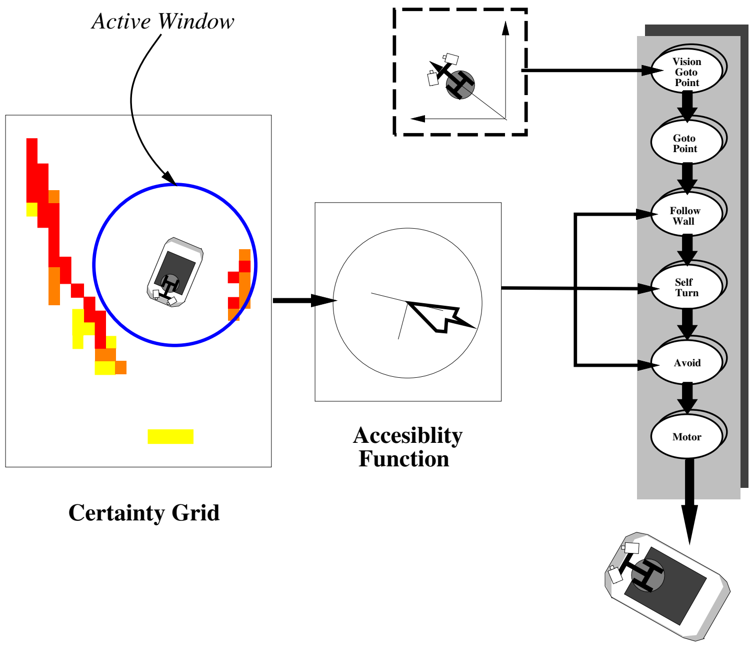


Fig. 3. Control structure of the Navigation Motor-Behaviors. Arrows indicate control and information flow.

makes it very hazardous and dangerous to navigate in the environment based on the last sensor readings. An obstacle that might have been visible by the sensors could be not visible a moment later (e.g., because of the relative angle between the robot and obstacle). Therefore we use a certainty grid to maintain a representation of the robot's local environment.

Certainty grids have been proven as an efficient method to represent empty and occupied space based on sonar sensor measurements [6]. They are based on the construction of a two-dimensional cartesian map, divided in a set of MxN square cells. Each cell contains a value that indicates the level of certainty that an obstacle is present in that location. A distance measurement is updated on the map by increasing the certainty that an object is present in the area that covers the cell. However, knowledge of regions located far from the robot is poor due to uncertainties in the measurements provided by sonar (like false reflections, crosstalk, false echoes, etc.) and odometric sensors (due to uneven and slippery floors).

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 Navigation Motor-Behaviors to take robust and reliable control actions. We are only interested in whether Navigation Motor-Behaviors react to a local and constrained area around the robot; "long-term" actions are taken into account by the vision system and the hints the user provides the robot. The only 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* [7].

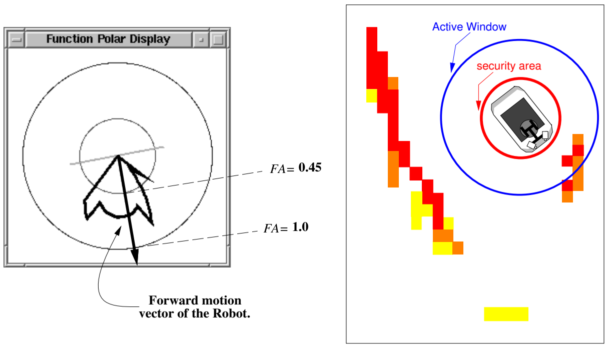


Fig. 4. The right image shows the location of the robot in the certainty grid, which represents the current status of the environment perceived by the robot. The left image shows the computed A_F in the *Active Window* area that surrounds the robot.

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

$$A_F(\theta_i) = d_i$$

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 θ_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 θ_i direction. (See Figure 4 for a graphical explanation).

A.1 Control Structure of Navigation Motor-Behaviors

The control architecture is implemented as a set of ordered behaviors, where each of the behaviors gets as input a control action from a higher level behavior and sends its output to a lower level behavior.

In our system, behaviors try to output a control action that depends on the sensor information they are processing, and the control output of the higher level behavior they are connected to. Instead of generating a control action that has nothing to do with the objectives of higher level behaviors (as for instance in reactive systems [2]), we try to ensure that each behavior outputs a control action that is as close as possible to the control outputs of other behaviors. For instance, if the **Avoidance** behavior detects a nearby obstacle, it will not just avoid it in a pre-determined way (by turning left or right in a reactive fashion); instead, it will take into account the output control "desire" of the **Goto Point** behavior, and try to direct the robot to a safe location that does not separate the robot too much from the path to the goal.

Figure 3 shows the priority order of Navigation Motor-Behaviors. 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, and directs the robot to a safe location.
- **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

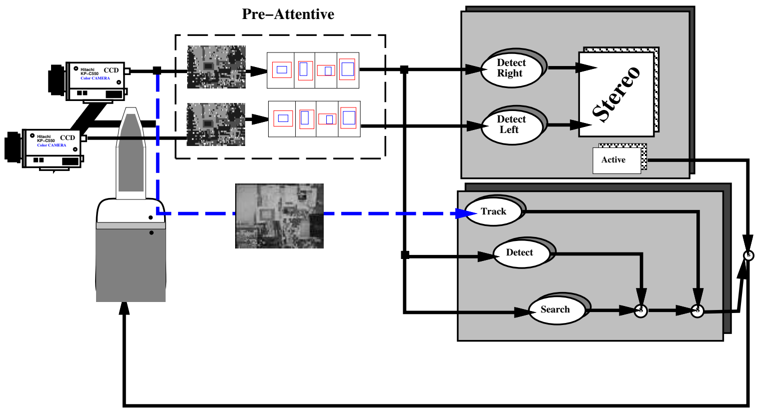


Fig. 5. Control structure of the Vision Motor-Behaviors.

Vision Motor-Behaviors 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 5 the control structure and sensory data flow of Vision Motor-Behaviors is shown. The color images grabbed by the cameras feed into the *Pre-Attentive* stage. In this stage, visual features indicating the potential presence of a beacon are extracted.

B.1 Pre-Attentive Stage

TABLE I

CODING IN THE PRE-ATTENTIVE STAGE.

Pixel	Coding
$R > G, R > B$	Red
$G > R, G > B$	Green
$B > R, B > G$	Blue
else	Unknown

Color information provided by the cameras is stored in an array of RGB pixels. As the goal of the vision system is to detect a fixed set of colored beacons, each pixel in the image array is transformed according to the rules expressed in Table I. This segmentation technique has the advantage of being very fast, and by means of its simplicity, it provides a robust way of sketching out the salient features. As we already mentioned, we are only interested in detecting combinations of colors that are usually not present in indoor environments. Therefore, the only piece of information relevant to the recognition algorithm is if a pixel has more of one color than the other two, for example, more "red" than "blue" and "green". We do not really care if the pixel is pink or brown instead of red, which might be the case due to different light conditions. What we are looking for is that the combination of "reddish", "greenish" and "blueish" regions in the image satisfy a set of *geometrical* and *temporal constraints*.

To do so, we join 4-connected pixels into regions of the same color [8]. The final result of the whole process is a symbolic list containing all the connected colored regions in the image that enclose other regions.

The process, carried out by the pre-attentive stage, is performed on 63x47 pixels sized images, at 24 images per second rate.

B.2 Behaviors

We divided the process of **search**ing, **detect**ing and **track**ing a beacon into three, almost independent, set of hierarchically structured visual behaviors. Depending on the state of the system (beacon lost or not found already, imminent detection, and tracking of the beacon), only one of the behaviors will be actuating on the motors of turret, inhibiting the outputs of other behaviors.

These behaviors obtain their sensory inputs from different sources: *Search* and *Detect* behaviors get their sensory inputs from the pre-attentive stage, while the *Track* behavior inspects the gray level image.

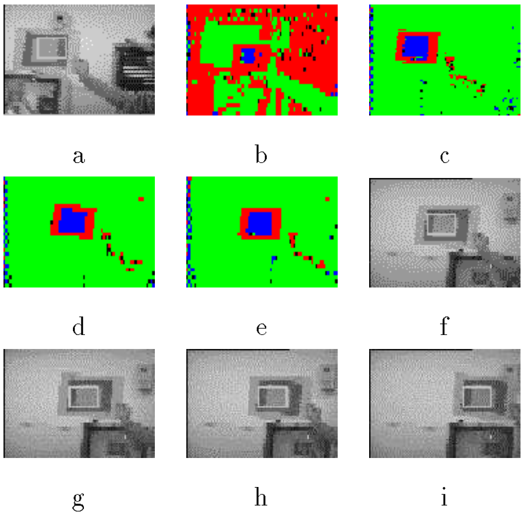


Fig. 6. **Example of the behaviors control flow:** A beacon (blue rectangle inside a red one) was moved from right to left. Image a) shows a grey level image, indicating that the *Track* behavior was tracking the beacon. Images from b) to e) show segmented images (according to the coding given in Table I), indicating that the beacon was lost by the tracking behavior and the *Search* and *Detect* behaviors where controlling the system. As it can be observed in figure a), the beacon's centroid was last seen on the left of the center of the image. Therefore, the *Search* behavior moved the turret to that direction (images c, d and f). Images f) to i) show the performance of the tracking behavior once the beacon was found again by the *Detect* behavior. The small white rectangles in the grey-level images indicate the position of the correlation mask used by the *Track* behavior.

Hence, if the system tracking the beacon, the pre-attentive stage process is not working, as there is no need to waste precessing time in computing image features that are not relevant to the task at hand (to track the beacon).

The **Search** behavior stops the turret if there is an element in the pre-attentive stage symbolic list. This behavior also remembers the direction where the beacon was last seen (left or right of the center of the image), 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 gray-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 gray level image.

If a new image is acquired at step t, we denote $I(u,v,t)$ as the gray 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 $I(u,v,t)$: $x_i \le x \le x_f$, $y_i \le y \le y_f$.

The correlation factor between $M(t-1)$ and $I(t)$ is defined as:

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

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 gray-level differences.

The best correlation factor between the image and the template is determined by:

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

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

If the best correlation 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 $t+1$.

The main reason of using a correlation technique on gray level images is that it enhances the tracking performance to a rate of 38 images per second.

B.3 Stereo vision system

The *Stereo* vision system determines the 3-D location of the beacon using a parallel camera model stereo rig. The goal of the system is only to compute the 3-D location of the beacon the vision system is tracking. Therefore, we use two *Detect* behaviors, each attached to the left and right color images, to find the corresponding projections of the beacon in both images. The cumbersome matching process gets thereby greatly reduced to a matching between two symbolic lists.

The result of the matching process is the location, in both images, of the centroids of the outer squares that constitute the beacons. The distance (z) from the beacon to the camera system can be easily computed by the following equation:

$$z = \frac{d_{12} f}{(v_2 - v_1) p}$$

where, d_{12} is the distance between the optical centers of the cameras, f is the focal length of the cameras (both are exactly the same), p is the width of the pixels in the camera array, and v_2, v_1 are the x coordinates of the matched points.

Stereo measurements of the beacon location are gathered in fixed time intervals in which the control actions of the Navigation Motor-Behaviors 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.

It is clear that measurements made with stereo vision systems are less precise when the elements are located far from the cameras due to the reduction in the differential angle between the cameras subtended at the object. Because of this, and the low resolution of the images used (63x47 pixels), stereo measurements over 4 meters are discarded, and not communicated to the *Goto Point* behavior. Finally, the definitive beacon's position used by the *Goto Point* behavior is the mean value of all the computed beacon's positions.

III. Expressing Plans

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. Part of the Motor-Behaviors described above will serve as basic units of communication, and their attached attributes, that improve and modify the robot's functionalities, will extend the number of symbols the user may use to communicate more complex plans to the robot.

Attributes are 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. 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.

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.

A. Symbols

The implemented Motor-Behaviors and their attributes serve not only as modular units of behavior generating capability, but also act as abstraction devices enabling the existence of an idealized and tractable

[1]Model S2200 of Data Cell Ltd.

TABLE II

SYMBOLS ATTACHED TO MOTOR-BEHAVIORS AND THEIR ATTRIBUTES.

Navigation Motor-Behaviors

Behavior	**Attribute(s)**
Avoid	
Follow Wall	Left
	Right
Goto Point	Reached

Vision Motor-Behaviors

Behavior	**Attribute(s)**
Track	
Detect	Blue in Red
	Blue in Green
	Red in Blue
	Red in Green
	Green in Red
	Green in Blue
Search	Forward
	Left
	Right
	Everywhere

representation of the robot's world.

Table II 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.

B. 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 are simple rules that 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].

C-Behaviors can be of two different types: **Perpetual** and **Contingent**. Perpetual C-Behaviors will

[2] In our architecture all behaviors have boolean attributes like "enable" and "active", which are not listed in Table II. 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.

always be active, monitoring motor behaviors and changing their attributes when certain conditions are satisfied. Contingency C-Behaviors change behaviors' attributes **once** the conditions they are monitoring are satisfied, and are not active any longer.

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.

C. The Plan

A very simple grammar (see Figure 7) permits the user to write plans the robot is able to understand. Based on this grammar, the plan (<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. 7. 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).

Each C-Behavior may have i pre-conditions and j post-conditions:

- 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 Vision Motor-Behaviors 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 II.

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.

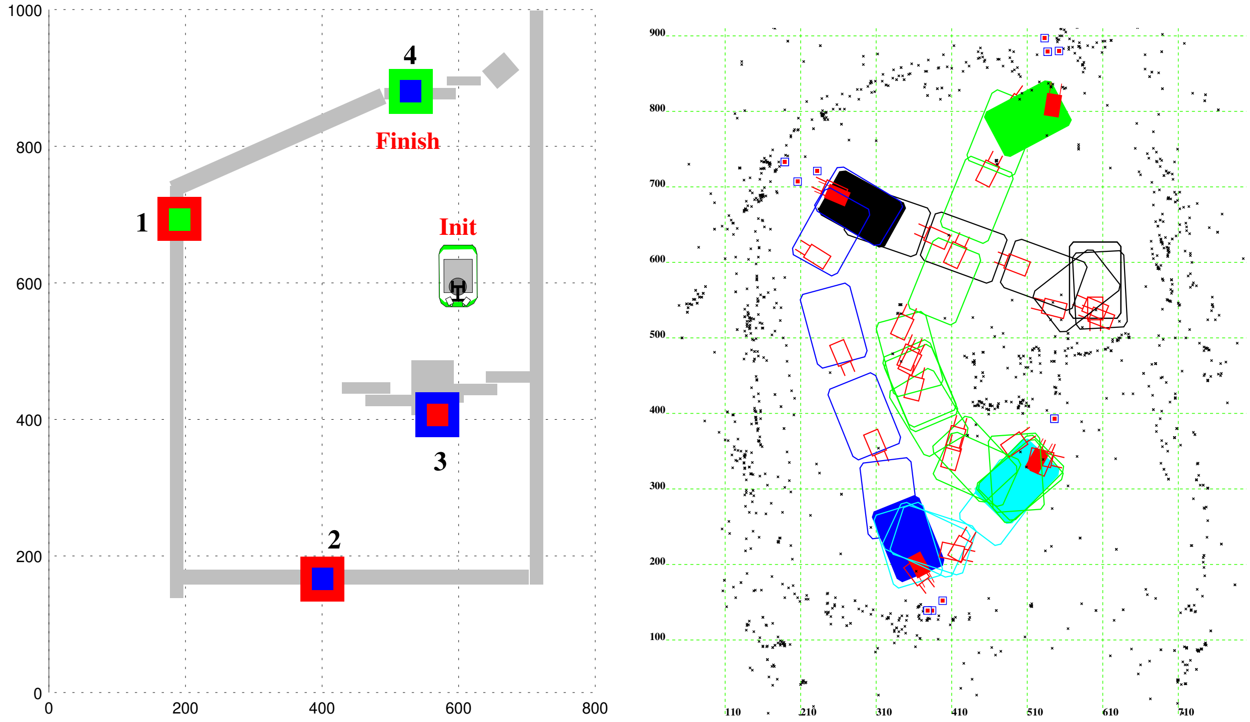
D. Embedded C-Behaviors

In order to (1) assist the user in writing plans that take care of the correct arbitration of behaviors, and (2) in providing 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. These behaviors are built-in Perpetual C-Behaviors that permit the user assumption that the robot has a certain common-sense. Obvious facts, 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 built in repertoire of behaviors, and therefore they permit a more flexible interaction between the 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.

IV. Experimental Results

Figure 8(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. See Figure 9 for a complete transcription of the plan provided by the user.

Figure 8(b) shows the complete trajectory followed by the robot to achieve all four steps of the plan.



(a) Scenario (b) Complete Path

Fig. 8. Scenario and performance. Numbers in the X and Y axis denote centimeters.

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 position and orientation once the step was accomplished.

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 built-in 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

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

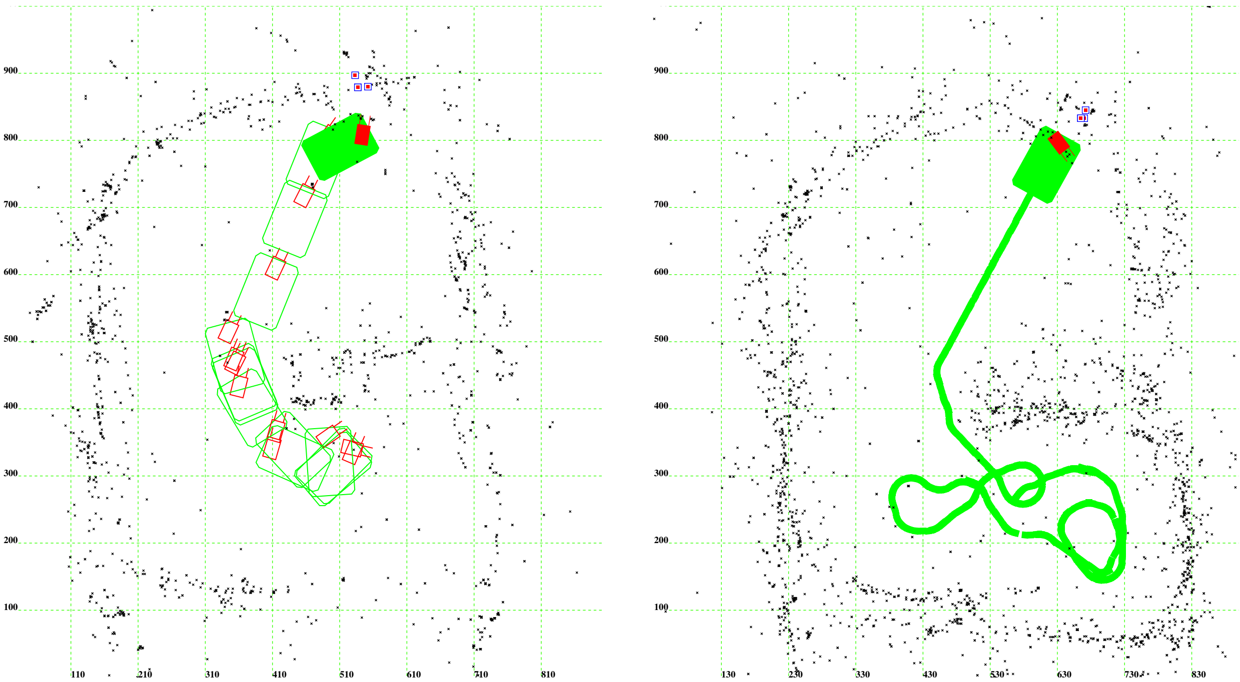
```
BEGIN 4
ONCE {1 TRUE 3
  ENABLE FollowWall 1 RIGHT
  ENABLE Detect 1 Red in Green
  ENABLE Search 1 RIGHT
}

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

Step 5

```
BEGIN 5
END
```

Fig. 9. The Plan.



(a) Performance **with** hints.

(b) Performance **without** hints.

Fig. 10. Comparison between performances with and without hints.

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. Following this, 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 9.

Step 4 is of special interest, as the obstacles present in the room prevented the cameras from visually detecting the beacon from the initial position of the step. The usefulness of the hint, of aligning to the right of the wall while looking for the beacon to the right side of the robot, is clearly show in Figure 10. Figure 10(a) shows the complete path followed by the robot when hints were provided, while Figure 10(b) shows the path when no hints were provided. In the latter case only the kind of the beacon to be detected (a red square inscribed inside a green one) was provided. This caused the robot to move about the environment to look for the beacon, and the smooth path followed by the robot using hints highly contrasts with the path followed by the robot when no hints were provided.

V. Conclusions

In this paper we have presented a complete control architecture for a mobile robot which is able to execute a plan given by a user in terms of goals and hints. We have integrated several sensor devices (sonar and vision) in an independent fashion: Instead of fusing the sensor information into a common symbolic structure [6], [9], we have adopted a parallel and decentralized approach at all levels, from basic motor behaviors to Conductor-behaviors.

Experimental results show the accuracy of the system in terms of perception and plan execution. Stereo measurements have an average of a 10% error margin, although the focus of this work was not precision in the robot's final location in front of the beacon (in the same way we do not explain someone the exact spot where to stop if we tell him how to get to the kitchen to clean the dishes).

Future work will mainly focus on creating a module of self-awareness, so that the robot knows when it is lost and cannot complete the plan. In our first prototypes we have worked with time thresholds, as for example, to detect if it takes too long to complete a given step. In this case, and just as usually people do, the robot should ask 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 Richard Watson, Mike Brooks and Jorge Gasós 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

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

[2] Rodney A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automatition*, vol. RA-2, no. 1, pp. 14–23, March 1986.

[3] Michael Schneider-Fontán, *Planning based on active perception*, PhD thesis, Universidad Complutense Madrid, Facultad CC. Fisicas, 1996.

[4] John Aloimonos, "Purposive and qualitative active vision", in *DARPA Image Understanding Workshop*, 1990, pp. 816–828.

[5] D. H. Ballard, "Animate vision", *Artificial Intelligence*, vol. 48, no. 1, pp. 57–86, Feb 1991.

[6] Hans P. Moravec, "Sensor fusion in certainty grids for mobile robots", *AI Mag.*, pp. 61–74, Summer 1988.

[7] Johann Bohrenstein and Yoram Koren, "The vector field histogram-fast obstacle avoidance for mobile robots", *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, June 1991.

[8] Robert M. Haralick and Linda G. Shapiro, *Computer and Robot Vision*, vol. 1, Addison Wesley, 1992.

[9] Richard E. Fikes and Nils J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.