Control of Mobile Robot Actions

Fabrice R. Noreils and Raja G. Chatila
L.A.A.S- CNRS
Robotics And Artificial Intelligence Group
7, Avenue du Colonel Roche
31077 Toulouse-Cedex FRANCE

Abstract

This paper deals with the control architecture of mobile robots. It is based on three types of entities: *Modules* that accomplish basic computations on data from sensors (resp. commands to effectors); *Processes*, in charge to establish *dynamic links* between perception and action to achieve closed-loop behaviors, and *Functionnal Units* providing specific functionalities. These modules are organized hierarchically, and are considered to be the robot's resources. We introduce and discuss a generic control system structure, based on a decomposition of its functions mainly into an *executive* managing robot resources, and a *surveillance manager* for detecting and reacting to asynchronous events. The control system enables the robot to execute missions (plans) expressed in a command language. Several running examples are given.

1 Introduction

In order to achieve a high degree of robustness and autonomy, a mobile robot should be able to plan and execute actions, and adapt its behavior to environment changes. The variety of environment conditions emphasizes the importance of multisensory perception on the one hand, and of robot reactivity at the various representation levels of its internal knowledge on the other hand.

We present in this paper an approach to the problem of action control for mobile robots, and the underlying software architecture. The hardware architecture on which this system is implemented is a general purpose multiprocessor system.

The control system manages the robot's resources (modules, sensors, effectors, processes and functional units) to enable it to execute a task given by a planner or a user, while providing reactivity to external stimuli. This includes the detection of, and reaction to, asynchronous events by adapting the robot's behavior so that the final goals are reached, and if not, the known reasons of failure reported. The software architecture should be robust and enable dynamic interactions between the robot's modules. It should also be flexible in order to integrate new functions, and be easy to implement and debbug [5].

Our approach to build a generic control structure meeting these requirements is to consider that the robot possesses functionalities achieved by related modules that can be defined and extended as the robot system grows more complex (section 3). The control structure does not merely integrate them "statically", but also provides for a mechanism for achieving programmable dynamic links between some of them to produce given behaviors. This structure (section 4) is composed of a *supervisor* that handles the global task sending each action to an *executive* that is in a way similar to an operating system, a *surveillance manager* for detecting some given events or situations and adapting the robot's behavior accordingly, and a *diagnostic* module for assessing the situations when an error occurs, and mending the on-going plan. Examples (section 6) showing the behavior of this system will be detailed. A command language for describing tasks (section 5), and corresponding to the robot's possible actions, is also introduced. Before detailing this architecture, we briefly overview some existing systems in the following section.

2 Related Approaches

For the A.L.V project [26], an architecture called the "Driving Pipeline" [14] that integrates multiperception and provides continuous motion was developed. The implementation is based on a "WhiteBoard" [23], derived from the BlackBoard paradigm, but wherein modules can asynchronously access the database by a token exchange mechanism. There are dedicated modules (Pilot, Helm, Obstacle Avoidance and Color Vision) with intrinsic synchronisation through messages exchange. Elfes [11] also presents an architecture based on a Blackboard paradigm and proposes a plan formalism (which handles parallelism). These are centralized systems (information contained in a central database); modules must have low granularity due to the centralized communication, and low communication bandwidth. Other authors propose a distributed approach. Brooks [5,13] claims for a behavioral decomposition and builds an architecture in layers of control corresponding to levels of behaviors to ensure reactivity. A given layer includes a number of modules with low bandwidth communication. A low level can be "subsumed" by a higher one. This is done by inhibition of the layer's inputs/outputs and timeout. The control is entirely wired, so it is irrevocable. An important issue is conflict resolution between behaviors [3,8]. This is a research area which has not been sufficiently studied and a necessary step in the development of the behavioral theory. Payton [22] introduces *Virtual sensors* to recognize features in the environment and send regularly their data to *reflexive behaviors* which in turn send commands (currently these are commands to motors) on a blackboard. A control module, using a priority arbitration, selects one message which is fed into vehicle actuators. Arkin [2] develops *motor and perceptual schemas*. A schema can be viewed as a computing agent. For each motor schema activated, perceptual schemas are created to detect events. If such an event occurs, a new motor schema is instanciated and enters in concurrency with the others. To ensure a high degree of concurrency, schemas communicate through a blackboard. Kaelbling [15] also uses behaviors but there are no connections between them. Behavior management is done by *mediators* which may inhibit behavior outputs according to internal conditions. Firby [12] also builds a reactive system but he uses a more traditional Artificial Intelligence approach to do it. He introduces *reactive action packages* (R.A.P) which are programs running until either the goal is reached or the R.A.P fails. Activation and conflict resolution are done by the *R.A.P interpreter*. Meystel [19] develops a strongly hierarchical architecture including rule-based systems. Kanayama [16] uses a decomposition of tasks in processes and develops an environment named RCS based on a multitasking paradigm.

3 Robot System Architecture

We describe now the robot architecture that the control system as such will manage. The atomic component of the robot system is called *module*. Figure 1 shows the modules integrated to date. There exist several kinds of modules.

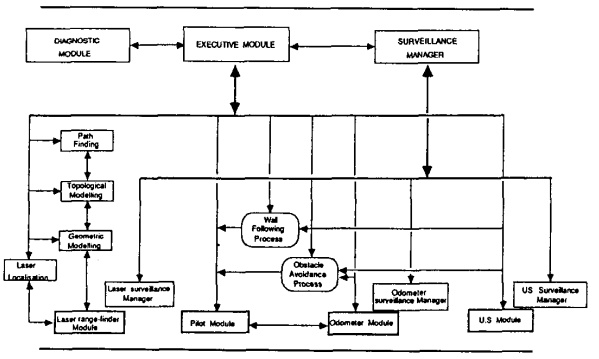

Figure 1: General Organization of The System

a) *Sensor* and *effector* modules that process sensor outputs or effector commands. These modules are layered according to a hierarchical internal representation of knowledge corresponding to levels of interpretation and abstraction, and therefore to levels of action of the robot.

b) Some modules, called *functional units*, provide a given specific function (*e.g.*, robot localization selecting and/or combining the available means to do it: vision, odometry, laser ranging, *etc.*)

c) A third kind of module implements the notion of *process* that provides, in general, for a dynamic closed-loop between perception and action, possibly using functional units.

3.1 Sensor and Effector Modules

The data acquired by a given sensor is processed by a sensor module and made available to the following higher-level ones for their own processings and interpretations. Some modules in this chain may have their input from different sensors. A similar hierarchy applies for the effectors, but here, the commands are interpreted downward. A sensor module achieves two main functions:

- Data processing: A series of processings are performed on incoming data to obtain different levels of interpretation that will then be used by the higher levels of our architecture.
- Reactivity to events: a level of interpretation corresponds to the extraction of a numerical or symbolic feature defined by a collection of parameters. The control system can instanciate a set of conditions which are applied to these parameters. Whenever the conditions are verified the control system is informed. These conditions are the left-hand-side member of what will be referred to as *surveillance monitors* discussed later in the paper (section 4.1.1).

As an example, let us consider a laser range finder on a pan scanning platform. First, each measurement ρ is associated with the sensor's orientation θ. Depth information is then extracted (from sensor calibration) providing a local position (x, y) in the frame related to the robot. A local map is built by linear approximations of depth points. This constitutes a chain of laser sensor modules. Successive local models can be integrated in a global map suited for navigation [6].
Other sensor modules are the Odometer and Ultrasonic Modules. The Pilot Module is an example of effector module. It provides a variety of trajectories that the robot can follow (such as lines, clothoids ...) or a tracking mode. In this mode the robot tracks a target (in its local frame) supplied by another module.

3.2 Functional Units

The information provided by a given level of interpretation can be used for a specific purpose by a dedicated module called functional unit. It is activated upon an explicit call and execute some computations, possibly making use of other modules. The algorithms included in a functional unit may be very complex. For example, robot localization is a functional unit. This unit may use different resources to provide robot position, for example a laser range-finder, or vision, by matching a local perception with a global model, or odometry, or a combination thereof. It also may compute robot position by deducing it from other knowledge (*e.g.*, the robot is docked to a known workstation). Robot localization can be given as a set of coordinates (and associated uncertainties), as well as a symbolic expression (*e.g.*, (IN_ROOM ROBOT R3)).

3.3 Processes

When we want to realize a *dynamic link* between perception and action, an entity called *process* is activated.

A process can be represented by a finite state automaton. Its inputs are data from sensor modules (regardless of the interpretation level), and its outputs are commands toward an effector. It has a control line required for:

- activating or halting the process by the control system. In our definition, a process is never generated dynamically but is continuously present in the system. Along with its activation, the control system can send some information (parameters) needed by the process for local decisions;
- Reporting process failures. If the input data requested from a sensor module is not sufficient to enable the process to accomplish its task, and if this situation occurs repeatedly, the process reports to the control system. Besides this case, a process does not stop by itself.

Instances of processes are Docking (to a workstation for example), and Local Obstacle Avoidance. An application of such processes with a detailed description of sensor/effector modules is presented in [21].

4 The Control System

We presented the elements that the robot control system will make use of for actually operating the robot. We shall now describe in detail the control system (or controller) itself. In our approach, its structure has both centralized and distributed components.
The controller has four components: **Supervisor (SV), Executive Module (EM), Surveillance Manager (SM)**, and **Diagnostic Module (DM)** (figure 1). Currently, the Supervisor and Diagnostic Module are not implemented, and their part is assumed by the Executive or by default decisions, as we shall see later. We will thus focus an the SM and EM, and firstly, we present the surveillance system that the SM controls. In former papers [6,25], this structure was presented in a first implementation wherein the functions of all these modules were achieved by a single one, called *decisional kernel*, that sets distributed surveillance monitors and is informed when they are triggered to decide for further actions after the first reflex reaction. We have refined this structure considerably.

4.1 The Surveillance Monitoring System

4.1.1 Surveillance monitors

Surveillance monitors play a key role in mission execution, and in robot operation. Their purpose is to set conditions on sensed data or computed results, and to trigger some immediate (reflex) reaction when these conditions are met. Formally, a surveillance monitor is equivalent to a classical rule [25]:

MNTR <conditions>⇒ <actions>

Where <conditions>contains conditions on sensors or robot state variables, and <actions>contains the set of reactive actions to carry when the left-hand side conditions are true. For example:

$$(\text{S\_zone IN } <\text{prmts}>) \Rightarrow (\text{set-speed } v)$$

is a surveillance monitor of type zone; if the robot enters in an area defined by a sequence of points (prmts), its linear speed is changed to v. In section 5, a more complete definition of surveillance monitors is given.

We define two main kinds of surveillance monitors: P-surveillances related to the on-going plan or mission, and provided along with it to detect some events or situation and cope with them, and O-surveillances related to the operation of the robot, even if no task is being executed (*e.g.*, battery power level).

All monitors are set by the Surveillance Manager that is informed when they are fired. However, for each sensor module, there is also a local surveillance manager.

4.1.2 Sensors and Surveillance Monitors

Conditions can be set at any level of data processing in a chain of sensor modules on their results in order to guarantee an immediate reaction when the events that produced them occur. On the same sensor module, several surveillance monitors can be active at the same time and are controlled by a *Sensor Surveillance Manager* (SSM). Its role is shown in the following example.

Several conditions may be monitored for instance by the ultrasound module (Figure 2) that performs a number of computation at different levels of representation. Conditions that can be monitored are, for example, related to the detection of a given pattern in the local environment model (such as <corner>or <edge>), specified with the appropriate parameters. The pattern may be defined by a logical condition on several ultrasonic sensors such as (AND (sensor$_i$ S_i)(sensor$_j$ S_j)), where S_i and S_j are values of sensor measurements. This actually corresponds to as many surveillance monitors. The SSM detects the simultaneous triggering of all the monitors in order to validate the global one, and then sends the corresponding message to the SM.

4.1.3 The Surveillance Manager

At the top level of the Surveillance system is the *Surveillance Manager*. Its function is twofold:

- It receives surveillance monitors from the Executive and decomposes the left-hand side member, sending the various components to the adequate SSMs.
- It receives the messages from the SSMs indicating that a surveillance has been triggered. If the left-hand side of a given monitor included a logical condition on different sensors, the SM verifies the logical formula. Then the reflex actions corresponding to the triggered monitors are executed through the Executive.

  If different surveillance monitors are triggered (almost) simultaneously, a *priority order* is used to decide what action to perform first. This solution is rather limited, and we are developing a new approach to this problem based on a precomputed decision tree.

4.2 The Executive Module

This module is in fact similar to an *Operating System*. The Executive Module (EM) receives the commands (requests, missions) through the user interface. Missions may include macro-commands that correspond to predefined procedures, and are interpreted by the executive by decoding a stored script. This first function will be in further implementations carried out by a separate module (the Supervisor) that sends the tasks to the executive each step at a time, leaving to the executive itself the task of managing robot resources (Sensor/effector modules, Functional Units, processes) to achieve the mission.

If an action is executed correctly, the executive launches the next one. Otherwise, in case of failure, the system is in one of the following situations:

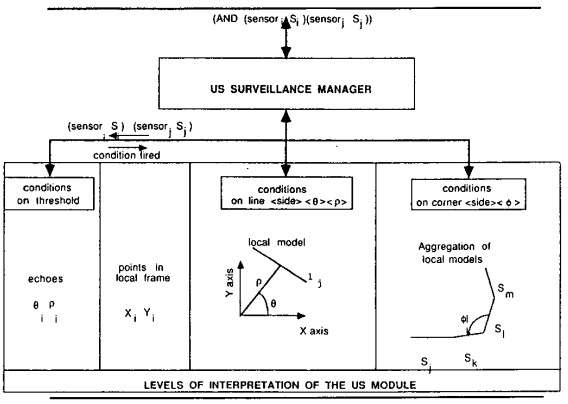

Figure 2: Levels of Interpretation and conditions of the U.S module

a) Process failure: A running process does not receive the data necessary to its computations because the sensor does not detect anything (*e.g.*, an obstacle avoidance has terminated, or the obstacle moved away), or because it is not adapted to changing external (environment) conditions (*e.g.*, insufficient lighting for vision).

b) Sensor/effector module failure: a module is unable to answer to its request, mainly because of physical malfunction (*e.g.*, battery power level, blocked wheel).

c) P-Surveillance triggering: In this case, an asynchronous event occured, and the right-hand-side of the surveillance monitor redirects the executive's action as programmed in the mission itself. The surveillance might be related to a running process, and in this case the process is halted.

Cases a) and b) might result in triggering related surveillance monitors (O-surveillances in general). All cases are detected because appropriate P-surveillances or O-surveillances were triggered. For instance, in case a), luminosity level is monitored, and if insufficient while a vision-based tracking process is active, the process is halted and a ultrasound-based process is activated instead [21]. After the execution of the reflex action, the Executive calls the Diagnostic module, providing it with the information from the failing module for further possible recovery action. The role of the diagnostic module is to assess a failure situation if necessary, after the first reflex reactions, and to provide a "local" plan to recover from it to resume the planned mission if possible. In order to be able to carry out its task, this module may require some data from sensor modules for example, through the executive. After diagnosis, error recovery may consist in executing a stored procedure. This module is not implemented yet as such, and will be composed of a compiled rule-based system. Currently, the executive system itself includes this function.

5 Mission Formalism

A mobile robot executes plans or missions provided by a planner or a human user. It is important to define the formalism in which the plan is expressed, and the information that it includes to enable the control system to execute it while complying with varying conditions. Our longer term goal is to define a command language.

The basic element in this formalism is a *command*. A command has two fields:

- The first field specifies the name and parameters of an *execution operator*. This is an order sent to sensor/effector modules or functional units, such as **read-pos** (read the position given by odometer) or **smooth-move()** (motion command to Pilot for excuting a path given as argument).
- The second field is optional. It specifies one or more surveillance monitors.

Before giving the formal command syntax, let us give some details on the role of surveillance monitors during plan execution. The monitors can fall in three different categories to this respect:

Terminal Surveillance Monitors : When triggered, the current action (execution operator) is stopped.

Non Terminal Surveillance Monitors : When triggered, they do not interfere with the current action, but start another one concurrently, provided it does not share the same resources. However, these monitors are linked to the current action and are inhibited or killed when it is achieved.

Static Surveillance Monitors : When a plan fails, the Executive cancels the current action and the related surveillance monitors, and calls the diagnostic module. However, it might be necessary that some monitors remain active, or be reactivated. We call them *static* monitors.

A command's syntax is then the following:

$< cmd >$ ::= $(< operator > \{< surv >\}^*)$
$< surv >$ ::= MNTR [STATIC] $< condition >$
⇒ <reflex-action>{<control-action>}

The reflex-action field is the immediate reaction to the event, launched by the Surveillance Manager. The last field {<control-action>} specifies the reaction through the Executive module after a surveillance is triggered and the reflex-action executed. The control-action may include a decision concerning the continuation or not of the mission -see examples below-).
A mission is then defined as follows:

<mission> ::= (MISSION <mission-name>(
[(PRE: <precondition-list>)]
[(ENV: <environment-list>)]
[(LSURV: <surveillance-list >)]
(MAIN: <body >)))
<body > ::= <mission>
| <cmd>

where:

- <precondition-list>is the set of conditions that must be true for executing the mission.
- <environment-list>sets the execution environment or model by specifying some indications to the controller for error recovery and local decisions.
- <surveillance-list>is the set of surveillance monitors that have to be active during mission execution.
- <body>is the sequence of commands to be executed. Sub-plans (or macros, or elementary tasks) can be included. Control structures (*e.g.,* loops) may also be used.

The control system will execute the mission step by step, but should be able to stop before the end of an action in some cases. For this purpose, *key-words* appear in the *control-action* field of the surveillance monitors. Currently these key-words may have two possible values:

NEXT : means that, the current action being stopped, its related monitors are killed and the next action executed.

FAIL : the whole mission is stopped and the diagnostic module is called.

A great flexibility is thus given to the control system to modify the programmed course of a plan according to new perceived environment or robot conditions. **The control system not only has the capacity of executing plans, but also provides with a programmable reactivity.** This is actually the central issue in our approach.
Other systems in the litterature include sensor perceptions in plans. Doyle [10] presents a system (GRIPE) which inserts perception requests as well as the expected results in order to check that an action has been accurately executed. For navigation in a known environment (defined as a set of labelled areas) Miller [20] develops a language which includes closed-loops on sensor data as well as actions on effectors. Chochon and Alami [7] describe an on-line system (NNS) for executing assembly tasks that include an execution model specifying alternatives based on sensor conditions. Gini and Smith [24] propose a system which expands a program also inserting conditions to check before executing the next action.

6 Application And Examples

Before presenting several examples to illustrate the control system operation, we describe the current implementation status on our mobile robot HILARE.

6.1 Implementation

HILARE has two independant driving wheels and a free wheel. Currently, its on-board sensors are:

- A ultrasonic sensor belt (16 sensors) [4],
- A Laser range-finder on a pan platform.
- An odometry system
- A trinocular stereovision system [9] (computations made on a ground workstation).

The Pilot Module [17] is the only effector module. Figure 1 shows the system's architecture, without detailing the sensor/effector modules (detailed in [21]).
The implementation is partly on-board, and partly on a SUN workstation (figure 3). The robot is radio-linked to the SUN.

6.1.1 On-Board Implementation

Sensor modules for the ultrasonics, odometer, and laser range-finder, as well as the effector module for motor control (Pilot module) are on-board. Two processes have also been implemented associating the ultrasonics and pilot modules: Docking (wall following) and Obstacle Avoidance.
The SSMs are not implemented yet at the sensor level, and the SM actually assumes their function. Therefore, the surveillance conditions can only concern the interpretation levels.
All programs are written in PASCAL or PLM86 and implemented on five INTEL 80286 and 8086s on MULTIBUS I (this configuration is soon to be changed to 68020s on VME).

6.1.2 Off-Board Implementation

The man/machine and graphics interface and part of the control system, which is currently in its development phase, are implemented on a SUN workstation. Eventually, the controller will be completely on-board.
The off-board system is constituted by:

- A parser (written with YACC/LEX) which parses the commands and generates a new structure used by the Executive Module.
- The Executive Module.
- The User and graphics interface.

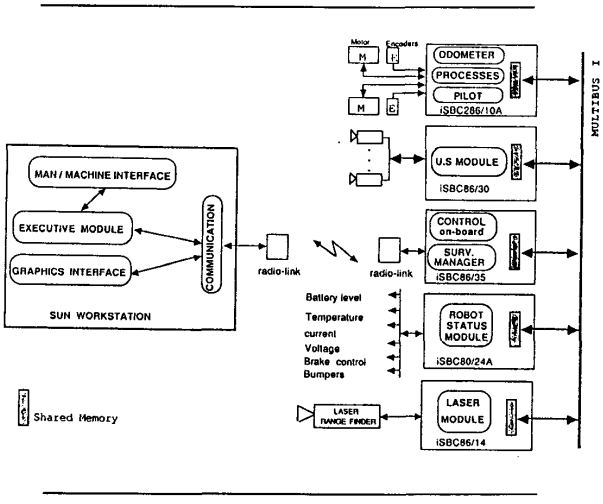

Figure 3: On-Board/ Off-Board Implementation

The basic function of the user interface is to send commands or requests to the Executive system and keap a history record of all previous commands. The graphics interface is used to display robot movements and sensor readings. These tasks communicate using a system developed at LAAS based on sockets, also enabling remote inter-process communication [1].

6.2 Examples of Mission Execution

We show on a first simple example the mission formalism and the system's behavior, and then an application of the approach in a more complex situation (example 2).

6.2.1 Example 1: Wall Following

Let us consider a robot moving in a room. We would like to reach a given location, near a wall, by navigating in the room to the wall and then change navigation mode automatically to wall following until the goal is reached. Goal coordinates are known (figure 4). This task is expressed as follows:

(MISSION mission1
(PRE: (eq motors ON))
(MAIN:
(smooth-move(...) MNTR(S_us-perception line (ρ, θ))
$\Rightarrow$ ((stop)){(NEXT)})
(RUN (wall-following (right-side 25))
MNTR(S_straight-line (x, y, ϕ)) $\Rightarrow$ ((stop)){(NEXT)})
)
)

Let us examin how the control system handles this mission. Figure 5 shows the exchanges between modules for the first command; messages are numbered as we reference them here for the sake of clarity. The executive module sends the first command (smooth-move) to the pilot module (1). Smooth-move is a command whose argument is a sequence of points (a broken line) along which the robot has to move, smoothing orientation changes by clothoid curves. When SM receives the surveillance monitor associated to the operator (2) it returns on the one hand an identifier to the executive (<n1>in (3)) and on the other hand sends the left member of the surveillance labelled by the identifier to the Ultrasonic SSM (4). The control-action field {(NEXT)} is memorized by the executive and referenced by the identifier sent by SM. During the movement, if the condition of the surveillance is triggered (5) (detection of a line (wall) with the given parameters -see figure 2), US-SSM emits a message to SM indicating that the condition identified by <n1>is true. SM executes the right member of the surveillance (6-1) and informs the executive (6-2). The Pilot module which receives the stop order, informs the executive that the movement has been interrupted (7). Then, the executive performs the control-action which is NEXT, meaning that the next command should be carried out. After destroying the previous monitor, this step is executed. It consists in running the *process* wall-following on the right side and at 25 cm distance from the wall, associated to a surveillance monitor for ending it when the robot crosses a straight-line defined by the parameters (x, y, ϕ) defining the goal. When this happens, the robot stops, and the NEXT control-action exits from the mission.

Figure 4 shows the robot executing this mission on the graphics interface. The command to provide this display is (show-robot on 50), specifying the refreshing period to the pilot for sending robot position. The ultrasonic echoes, along with the sensor identifier (between 1 and 16) are also shown.

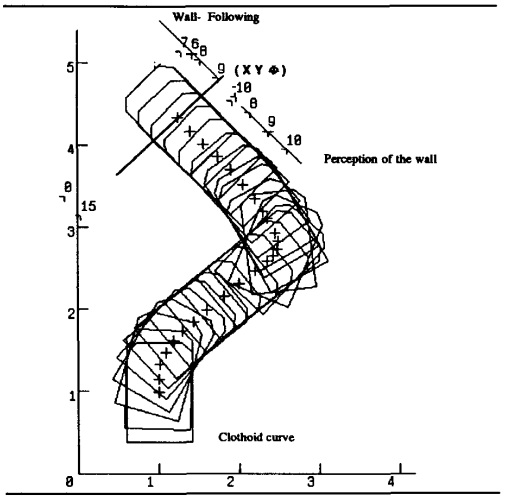

Figure 4: Wall Detection and Following

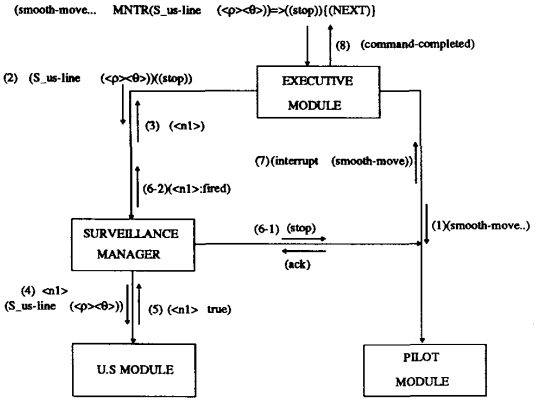

Figure 5: Flow of messages between modules

6.2.2 Exemple 2

In this example, the robot reports its ultrasonic sensor readings while passing by an object expected to be along its computed trajectory inside a specified zone. Furthermore, if an unexpected obstacle lies on the path in this zone, the mission terminates with a failure.

```
(define Tclist[4] ZONE109 ((150 500)(150 700)(350 700)(350 500)))
(define Tclist[4] sensors-list ((1 40)(2 50)(3 50)(4 40)))
(define Tpath PATH ((180 520)(350 520)(360 320 2700)))
...
...
(MISSION mission2
(PRE: (eq motor ON)(not (eq robot-in-zone $ZONE109)))
(ENV: (set LOA T))
(MAIN:
  (smooth-move(PATH)
              MNTR(S_zone IN ZONE109) ⇒ ((set-speed 20)){
                 (set robot-in-zone ZONE109)
                 (set LOA NIL)
                 (MNTR(S_zone OUT ZONE109) ⇒ ((set-speed 50)){
                                    (set robot-in-zone NIL)
                                    (set LOA T)
                 }
                 (WHILE (robot-in-zone)
                    (read-all-us-data)
                    (draw %G_pos%G_echoes r-position r-us-data)
                 )
              }
              MNTR STATIC (S_us-threshold sensors-list) ⇒ ((stop)) {
                 (IF (LOA)
                    ((avoid-obstacle)(resume-order))
                    ((FAIL))
                 )
              }
  )
)
```

In this example, the mission includes nested structures, variable definitions with type (*e.g.*, define Tclist ZONE109), a control loop, and a conditional statement. The first three lines define variables that will be used later in the mission. The precondition field specifies that the mission can only be executed if the motors are on, and the robot not already in the defined zone. The execution environment, or model, specifies that local obstacle avoidance (LOA) is active, *i.e.*, the robot is to go around unexpected obstacles lying on its path, and not just to stop when detecting them. Two surveillance monitors are active during the mission. The first one (S1: MNTR(S_zone IN ZONE109) ⇒ ...) is verified if the robot enters the specified zone, and the second (S2: MNTR STATIC (S_us-threshold ...) reacts to the detection of an obstacle within a given distance of the ultrasonic sensors. Once the robot is inside the zone, obstacle avoidance is prohibited. Two cases may occur in these conditions:

- Suppose that S1 is fired (and not S2). After the reflex action **set-speed** the EM executes the corresponding control-action: it sets the variable robot-in-zone to its new value, inhibits local obstacle avoidance, creates a new surveillance monitor (S1-2: (MNTR(S_zone OUT ZONE109) ...) to be fired when the robot exits the zone, and defines a loop which reads ultrasonics sensors and sends results as well as robot position to the graphic interface as long as the robot is in the zone. If S2 is then fired, the executive carries out its control-action (the IF statement), and since LOA is desactivated, FAIL is executed and the overall mission aborts.
- If S2 is fired before S1, the elementary-task avoid-obstacle, which includes the avoidance process, is executed because LOA is activated. During its execution, if the robot enters the zone, (S1) is fired, and LOA is set to NIL. The executive stops the avoidance task because its execution environment (LOA T) is not valid anymore. At this moment, S2 is fired again because it is *static*, and its conditions are still true (the obstacle is still there), but now LOA is NIL and the FAIL is executed thus aborting the mission.

Figure 6 shows the robot avoiding an obstacle and returning echoes to the graphics interface while in the specified zone.

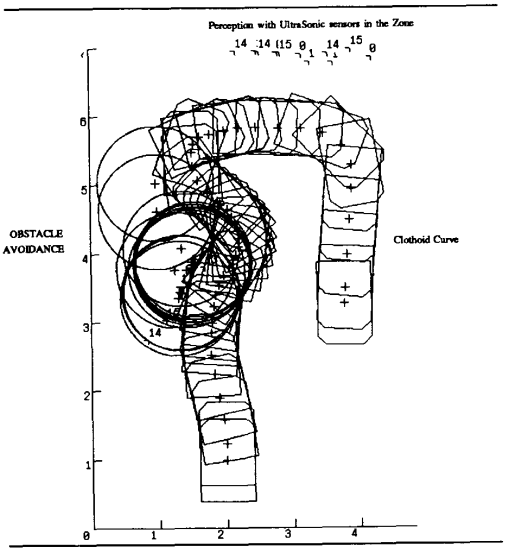

Figure 6: Obstacle Avoidance and U.S. data transmission in the specified zone

7 Future Work

We presented a control system architecture for mobile robots able to execute missions specified by a user, and having reactivity and error recovery capacities. This system uses the concept of process to achieve a controlled reactivity through the use of surveillance monitors providing a flexible control of these processes.
Future developements will concern mainly:

Integration of vision: This approach enables to easily integrate new sensors or modules to a mobile robot, the control system being able of using them according to the conditions specified in surveillance monitors for example. Vision for localization [18], object tracking and motion control in a structured environment (corridor) is currently being integrated.

Flexibility : Integration of vision will intensively make use of processes. Currently, their inputs and outputs are fixed. In order to reconfiger the various processes according to the context of use, their inputs/outputs will be typed, and a formalism to manipulate them developed.

Mission Formalism: Including control structures in the mission formalism will lead eventually to a language for programming mobile robots.

This approach provides the possibility for integrating new sensors and modules easily through the definition of their conditions of use. It will eventually make use of a language, of which elements were presented, useful to define "(re)programmable mobile robots" that could be reconfigured according to applications.

Acknowledgements: This work would not have been possible without the help of G. Bauzil, G. Vialaret and C. Lemaire for HILARE's hardware, and A. Khoumsi who implemented the Pilot Module.

References

[1] Rachid Alami. CPU Manuel d'Utilisation. Technical Note, L.A.A.S., Robotics Group, 1988.

[2] Ronald C. Arkin. Motor shema based navigation for a mobile robot: an approach to programming by behavior. In *IEEE, International Conference on Robotics and Automation, Raleigh*, pages 264 – 271, 1987.

[3] J. C. Aylett. Some speculations on the behavioural paradigm. Discussion Paper No 56, 1988. Department of Artificial Intelligence, University of Edinburgh.

[4] G. Bauzil, M. Briot, and P. Ribes. A navigation sub-system using ultrasonic sensors for the mobile robot HILARE. In *1st International Conference on Robot Vision and Sensory Control, Stratford-Upon-Avon*, April 1981.

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

[6] Raja G. Chatila. Mobile robot navigation : space modeling and decisional processes. In *3rd ISRR, Gouvieux, France*, 1985.

[7] H. Chochon and R. Alami. NNS, a knowledge-based on-line system for an assembly workcell. In *IEEE, International Conference on Robotics and Automation, San Francisco*, pages 603 – 609, 1986.

[8] Peter W. Cudhea and Rodney A. Brooks. Coordinating multiple goals for a mobile robot. In *Intelligent Autonomous Systems*, pages 168 – 174, 1986.

[9] A. Robert de St Vincent. A 3d perception system for the mobile robot HILARE. In *IEEE, International Conference on Robotics and Automation, San Francisco*, 1986.

[10] Richard J. Doyle, David J. Atkinson, and Rajkumar S. Doshi. Generating perception requests and expectations to verify the execution of plans. In *5th national conference on artificial intelligence*, pages 81 – 88, 1986.

[11] Alberto Elfes and Sarosh N. Talukdar. A distributed control system for the C.M.U rover. In *8th IJCAI, Karlsruhe (FRG)*, 1983.

[12] R. J. Firby. An investigation into reactive planning in complex domains. In *6th national conference on artificial intelligence*, pages 202–206, 1987.

[13] Anita M. Flynn and Rodney A. Brooks. MIT mobile robots: what's next? In *IEEE, International Conference on Robotics and Automation, Philadelphie*, pages 611 – 617, 1988.

[14] Yoshimasa Goto, Steven A. Shafer, and Anthony Stentz. *The Driving Pipeline: A Driving Control Scheme For Mobile Robots.* Technical Report CMU-RI-TR-88-8, Robotics Institute, Carnegie Mellon University, 1988.

[15] Leslie P. Kaelbling. *An Architecture for Intelligent Reactive Systems.* Technical Report, Artificial Intelligence Center SRI International, April 1986.

[16] Y. Kanayama. Concurrent programming of intelligent robots. In *8th IJCAI, Karlsruhe (FRG)*, pages 834 – 838, 1983.

[17] Ahmed Khoumsi. Pilotage, asservissement sensoriel et localisation d'un robot mobile autonome. Doctorat de l' Université Paul Sabatier, June 1988.

[18] Eric P. Krotkov. *Mobile Robot Localization using a Single Image.* Technical Report 88125, L.A.A.S., May 1988.

[19] A. Meystel. Planning in a hierarchical nested autonomous control system. In *SPIE vol. 727 Mobile Robots*, pages 42 – 76, 1986.

[20] David P. Miller. *Planning by Search Through Simulations.* PhD thesis, Yale University, Department of Computer Science, October 1985.

[21] Fabrice R. Noreils, A. Khoumsi, G. Bauzil, and R. Chatila. Reactive processes for mobile robot control. To appear in International Conference on Advanced Robotics (ICAR), 1989.

[22] David W. Payton. An architecture for reflexive autonomous vehicle control. In *IEEE, International Conference on Robotics and Automation, San Francisco*, pages 1838 – 1845, 1986.

[23] S. A. Shafer, A. Stentz, and C. E. Thorpe. *An Architecture for Sensor Fusion in a Mobile Robot.* Technical Report CMU-RI-TR-86-9, Robotics Institute, 1986.

[24] Richard E. Smith and Maria Gini. Reliable real-time robot operation employing intelligent forward recovery. *Journal of Robotic Systems*, 3(3):281– 300, 1986.

[25] Ralph P. Sobek and Raja G. Chatila. Integrated planning and execution control for an autonomous mobile robot. *The Int. Journal for Artificial Intelligence in Engineering*, 3, April 1988.

[26] Charles Thorpe and Takeo Kanade. *1987 Year End Report for Road Following at Carnegie Mellon.* Technical Report CMU-RI-TI-88-4, Robotics Institute, Carnegie Mellon University, 1988.