An Architecture for Task Interpretation and Execution Control for Intervention Robots: Preliminary Experiments

Raja Chatila Rachid Alami Bernard Degallaix
Victor Pérébaskine
Paul Gaborit Philippe Moutarlier
Laboratoire d'Automatique et d'Analyse des Systèmes (LAAS-CNRS)
7, Ave. du Colonel Roche 31077 Toulouse cedex - France

Abstract

This paper deals with the software system design of robots that can replace or assist humans in intervention tasks in hazardous environments. We consider missions that include navigation from a site to another, execution of specific tasks, and transmission of data. Communication difficulties with a remote ground station (delays and bandwidth), as well as time constraints on the tasks make it necessary to endow such robots with a high level of autonomy in the execution of a well specified mission. However, it is today unrealistic to build completely autonomous (and useful) robots. This paper presents an approach to answer these requirements; it describes an on-board robot system architecture that enables the robot to interpret its mission according to the context, and to control and adapt in real time the execution of its actions.

1 Introduction

Intervention robots are machines that have to perform tasks in difficult environments which are often remote or of difficult or dangerous access. In addition, according to the application context, specific constraints, related to the interaction with the human operator, to the task itself, and to the environment have to be taken into account:

- Communication constraints: time delays (for example one way light travel time between Mars and Earth is between 5 and 20 minutes); impossibility or loss of communication (a rover on Mars may communicate with Earth only at certain periods of the day because of visibility); low bandwidth.
- Task constraints: duration (e.g., motion only during daytime in planet exploration); non-repetitiveness: a human operator should be able to assign a variety of tasks to the robot, and to modify them according to returned information.

- Environment constraints: the environment is partially known and with little accuracy before the intervention, and may remain so for the human operator.

The question addressed in this paper is: "What robot architecture is adapted to cope with these requirements and constraints?".

These constraints forbid, in many applications, the use of classical teleoperation to control intervention robots, as well as telerobotics-like [15] approaches, wherein the human operator is still tightly in the control loop even if the robot has the capacity to execute some tasks automatically [9].

On the other hand, research on autonomous mobile robots is not (yet) able to produce an intelligent agent capable of accomplishing completely by itself a general objective such as "Collect *interesting* rock samples", non withstanding the fact that the specification of this objective itself cannot be really done beforehand (what is an *interesting* rock?).

In another stream of thinking, *collective intelligence* is considered to be more easy to accomplish by building small robots which have complete autonomy but limited behavior, and wherein *"the basic components that make up the system are designed in such a way that the desired functionality emerges as a global side-effect of their massively parallel, simple behavior"* [7]. Of course, the problem is now "how can global functionality emerge from local behavior"[5]. The shortcoming of such a system, even if successfull as defined, is that it is *not* programmable, i.e., it is necessary to fully specify the task at robot design stage.

In the *"task level" teleprogramming* approach presented in this paper, the robot is provided with *executable* missions that will be carried out autonomously. These missions (plans) are refined into actions according to specified execution modalities and to the actual execution context [13, 2]. The robot control structure includes the systems necessary for task interpretation and autonomous execution. The system is aimed to be generic and applicable to several domain instances of intervention robots: planetary rovers [6], underwater vehicles [11], disaster reaction [13], etc.

Section 2 presents the functional architecture of the overall system and and section 3 presents the operation of the on-board supervision and task refinement and execution control system.

In order to illustrate the approach we will consider a Mars Rover as an example of intervention robots. This case study is motivated by the french national project VAP[1][6].

Once safely on Mars, the rover is expected to be operational for about one year and to carry out eight hours per day activities (during daylight). Missions will have three main aspects:

- experiments within a worksite, such as sample collection, drilling, or various measurements,
- autonomous navigation between two worksites, within time and space (route) constraints given with the mission. Navigation includes natural landmark recognition, inertial localization, environment modelling for motion planning and execution on rough terrain ...
- worksite modelling and data transmission to Earth.

[1]VAP: Autonomous Planetary Vehicle, a project of the french national space agency CNES.

2 A Functional Architecture for Task-Level Teleprogramming and Action Execution Control

The proposed functional architecture (Figure 1) involves two distinct systems: the *Operator Station* and the on-board *Robot Control System* (RCS).

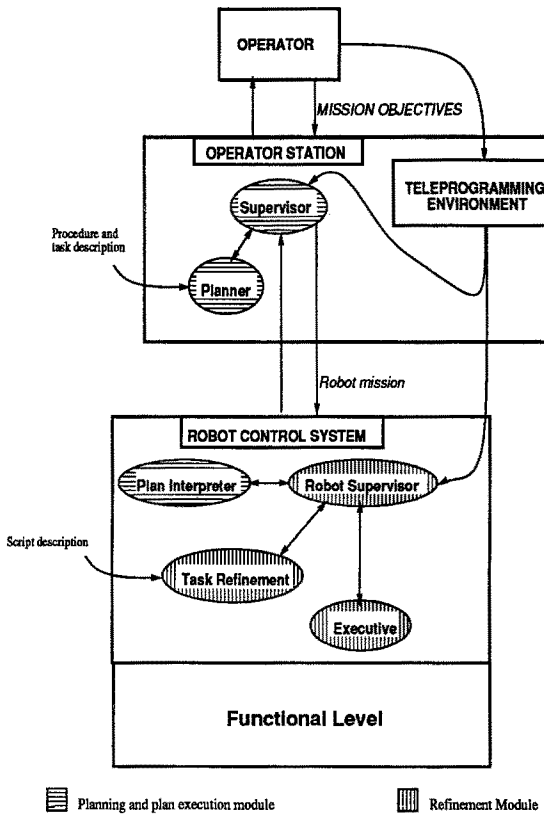

Figure 1: Global Functional Architecture

2.1 The Operator Station

The Operator Station includes the necessary functions to allow a human operator to build an *executable mission*, i.e., a mission that can be interpreted by the RCS, and to supervise its execution by the robot.

Its main components are a Mission Planner which determines the tasks to be achieved and their ordering and a Task-level Teleprogramming environment which provides information that will allow the Robot Control System to refine and execute the tasks.

2.1.1 Mission Planning

The mission planning phase is based on general action planning techniques, including temporal reasoning, since it is necessary to take into account time constraints in robots that have to act in the real world.

We have developed a temporal planning system called *IxTeT* (Indexed Time Table) [12] which can reason on symbolic and numeric temporal relations between time instants. It produces a set of partially ordered tasks with temporal constraints. This issue will not be developed further in this paper.

The main reason why this phase must be performed at the Operator Station stems from the fact that, in the case of an intervention robot, the determination of the goal itself is based on interpreting the current work environment. Such an activity is naturally performed by a human operator who is best able to decide what the robot should do, given the data acquired by the robot.

2.1.2 Task-level teleprogramming

The purpose of this phase is to transform a mission planned according to some objectives into an executable mission interpretable by the robot. It is also a planning phase. However, it relies on specialized planners (e.g. geometric planners, manipulation planners) that are able to take into account directly the interactions between the robot and its environment.

An executable mission is composed of a set of partially ordered tasks (the *mission plan*q, procedures that need no further refinement, numerical data and task-dependant data structures, and *execution modalities*. These comprise:

- information produced by the operator station which will guide the refinement of the mission by the robot;
- constraints and validity conditions on task execution;

One key aspect is that this "programming" phase must be performed using partial and inaccurate information about the robot's environment, and about the consequences of the robot's actions. It must therefore produce programs which are robust with respect to these uncertainties, and which include actions that allow the robot to verify without ambiguity that it is indeed executing the required task. Such verification cannot be performed in terms of absolute values (which cannot be precisely known at programming time), but in terms of relations between the robot and identifiable features in its environment.

This means that the resulting program must rely on *sensor-based actions* (e.g. feature tracking) to allow the robot to permanently adapt its behavior and take appropriate actions when it detects any discrepancy between the planned state and the actual state of the world. Note that these requirements may drastically influence the way in which a goal is refined into tasks. For example, the need for robust execution of a motion between two points may entail a choice of trajectory which is not necessarily optimal in length or duration, but which allows the robot to track some feature.

Finally, depending upon the difficulty of the task and the nature of the environment, the teleprogramming phase can sometimes rely on the capacity of the RCS to successfully interpret high-level commands. But in some other situations, the task must be

deeply detailed by the operator station, which normally requires previous acquisition and transmission of large amounts of data.

In the case of the planetary rover, the tele-programming of navigation tasks will be limited to the determination of itineraries and natural landmarks based on low-resolution images obtained from the orbiter. Further task refinement is performed autonomously by the RCS. Other tasks, like sample collection, must be programmed at the ground station in more details using "in situ" information acquired by the rover.

2.2 The Robot Control System

Once the executable mission is prepared at the Operator Station, it is sent to the on-board system called the *Robot Control System.*

Because the robot is working in a remote and a priori little known environment, significant differences between the planned (expected) state of the world and its true state are bound to appear. Therefore the robot is *fully autonomous* at **"task level"**. It receives tasks that it transforms into sequences of actions using its own interpretation and planning capacities, and executes these actions while being reactive to asynchronous events and environment conditions. However, it may decide (by itself according to the context, or because it was programmed explicitly to do so) to contact the Operator Station when necessary (and possible) in order to solve some difficulties. Conversely, the Operator Station may also supervise the execution if possible, and if necessary interfere with action execution.

Thus, the robot is endowed with the capacities of (i) interpreting the mission according to the actual situation, and (ii) autonomous execution of its tasks. The on-board **Robot Control System** (figure 1) is composed of two levels: a supervision and planning level, and a functional level.

2.2.1 The Supervisory Level

The supervision level comprises:

- The **Robot Supervisor**, which manages the overall robot system and interacts with the operator station. The supervisor makes use of two resources:
  - the **Plan Interpreter and monitor** that is in charge of requesting (to the supervisor) the execution of the different tasks in the plan, of verifying that their execution satisfies the different constraints included in the plan, and of maintaining the necessary world state description.
  - The **Task Refinement System** which refines tasks into actions, taking into account the execution modalities specified with the plan, and the actual situation of the robot.
- The **Executive**, which is in charge of managing and controlling the robot's fucntional modules in order to execute actions.

2.2.2 The Functional Level

In order to control the robot's actions, we think that it is useful to define its basic functions in a systematic and formal way so that they can be controlled according to their specific features, while being easy to combine, modify or redesign [2]. We therefore defined in [14] and extended in [10] *robot modules* and introduced *primitive function types* within them [1] [8].

Essentially, a module embeds primitive robot functions which share common data or resources. An internal control task called the "module manager" is responsible for receiving requests to perform these functions from the robot controller, and for otherwise managing the module. Each function being well defined, its activation or termination must respect certain conditions that the module manager verifies.

A module may read data exported by other modules, and may output its own processing results to exported data structures (EDS). At a given time, a module can be executing several functions. All of the functions of each module are pre-defined at the system design stage.

The robot primitive functions fall into four different *types* according to their functioning mode.

Servers. Functions executed upon request. The processing result is exported in a predefined data structure to be accessed by the requesting module, or sent directly (message).

Filters. Functions started upon request (or systematically) and then run continuously at a given frequency until stopped. Their results are output in a data structure that is updated at the mentioned frequency.

Servo-processes. These functions implement a closed-loop between a perception function (related to processing sensory data) and an "action" function (related to robot effectors).

Monitors. These functions are used to detect a given situation or event. They verify, at a given frequency or at reception of an asynchronous signal, the occurrence of an event and react by generating themselves a signal (sent to the supervisory level or another pre-specified destination).

Both filters and servo-processes may be stopped, but cannot stop by themselves, unless they detect an abnormal condition; in this case, an appropriate signal is sent to the supervisory level of robot control system.

3 Plan Interpretation, Task Refinement and Execution Control

3.1 Introduction

After the planning and programming phase, an *executable mission* is produced composed of tasks, and possibly for some parts, of more low-level actions produced by the programming phase. The mission plan has to be tranformed by the robot control system into

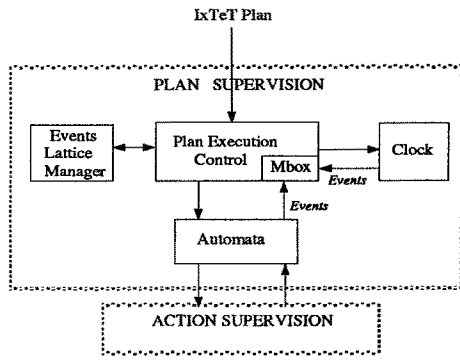

Figure 2: The Plan Interpreter

executable primitive actions, selected by taking into account the state of the environment and of the robot as the mission is executed. Mission execution will be globally managed by the robot supervisor, that makes use of the plan interpreter for scheduling the tasks, of the task "refiner" for precising their execution, of the robot executive that manages the robot functional modules for actually executing them.

3.2 Mission Interpretation

Plans, as produced by IxTeT, embed a set of partially ordered tasks, together with temporal constraints such as minimal and maximal expected durations, and synchronisation with expected external events or absolute dates.

The plan interpreter is in charge of monitoring the execution of the tasks involved in the plan. It has therefore to verify that the tasks produce indeed the expected effects, and must react in case of discrepancy. Some of the tasks need to be further refined according to the actual situation. They will be sent by the supervisor to the task refinement system.

3.2.1 Plan Execution Monitoring

Instants in the time lattice correspond to different event types: beginning or end of task execution, intermediate events produced by tasks during their execution, expected external events (which occur independently from robot actions). Besides, numerical bounds for dates and durations may be attached to some time-points or intervals.

The plan execution control process interacts with a clock by requiring the sending of messages at given absolute dates and with the robot supervisor by requiring task execution or cancellation (fig 2).

Messages received from the clock authorize the system to state the occurence of dated expected events and to monitor the task minimal and maximal expected durations.

Messages received from the robot supervisor are 'filtered' by the associated automata and transformed into events which correspond either to instants planned in the time lattice (in case of nominal execution) or to unexpected instants otherwise.

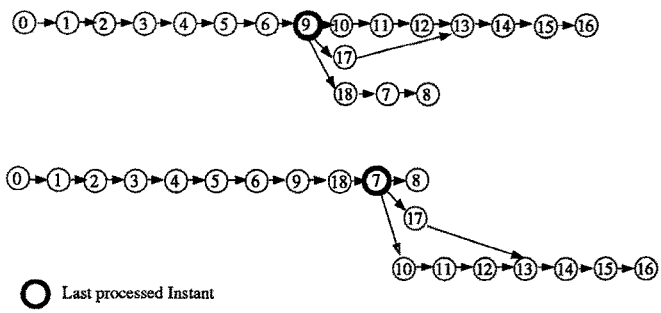

Figure 3: Plan interpretation

The plan interpreter starts from a given instant in the time lattice and 'executes' the time lattice by performing the following actions:

- At any moment, it considers only the time instants whose predecessors have been processed and whose planned occurence date is compatible with the current time.
- It requires the execution of a new task when it reaches the instant which corresponds to its beginning.
- While processing incoming events, the interpreter verifies that they correspond to planned instants and that they satisfy the planned ordering and numerical time constraints. If it is the case, the absolute date corresponding to the occurence of the events is considered, inducing a progressive 'linearisation' of the time lattice (see figure 3).
- In the current implementation, the only 'standard' reaction performed by the plan interpreter in case of non-nominal situations is to stop the current tasks, and update the world state according to events incoming from task execution until all tasks are stopped. This is performed by inserting new time instants in the time lattice corresponding to the transitions as defined by the automata.

Figure 3 shows the plan produced at two different stages of interpretation. The first time lattice represents a situation wherein the rover is navigating from *site-2* to *site-3* (indeed, instant 9 is in the past while instant 10 is still in the future). It has previously acquired a panoramic view of *site-2* (between instants 5 and 6), however it has to wait for instant 18 (visibility with the orbiter) in order to send the data (between instants 7 and 8).

In the second lattice, instant 18 has occured; the rover is performing two tasks in parallel (navigating and sending data to the orbiter).

3.2.2 Using Automata To Monitor Task Execution

In order to allow the plan interpreter to monitor task execution and to act on them while they are executed, each task is modelled by a finite state automaton (FSA).

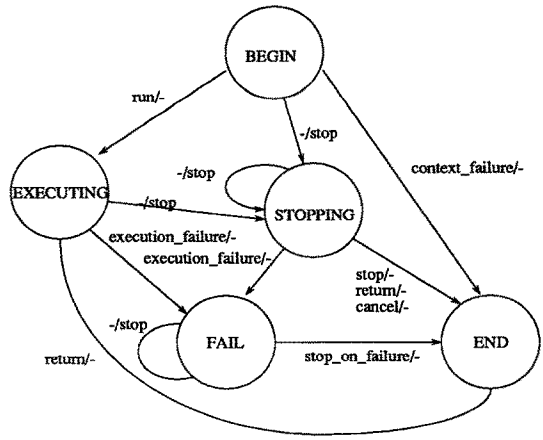

Figure 4: A Generic Automaton for task execution. Events are represented by a pair internal/external

The FSA associated to a task not only models the nominal task execution (as defined in the task description manipulated by the planner) but also non-nominal situations (exceptions), as well as the different actions which should be taken then by the plan interpreter.

In the finite state automata we use, state transitions can be caused by 'internal events' (i.e. events generated by the task execution process and transmitted to the plan interpreter) or by 'external events' (i.e. events due to an action of the plan interpreter on the task while it is executed).

Figure 4 shows a generic automaton for action execution.

Typical internal events are: RUN (beginning of execution), RETURN (nominal end of execution), STOP_ON_FAILURE (end due to a failure), STOPPED (end of execution caused by an external STOP request).

A Typical external event is STOP (corresponding to a stop request issued by the plan interpreter). Other external events are related to the task and force a state transition in the execution.

For each task, an automaton class is provided. An example is given in figure 5. Note that each automaton state change produces transitions in the world state as it is maintained by IxTeT. The example in figure 5 shows a decription of automaton associated to the task *TASK-NAVIGATE*.

Whenever a task execution is requested by the plan interpreter, an automaton of the associated class is instanciated and initialized. It will then represent the execution of the task as it is viewed by the plan interpreter.

```
(ixtet:Task TASK-NAVIGATE
     :args (?sitel ?site2)
     :effects ((:is-met VAP-Site ?sitel)
               (:meets VAP-Site ?site2)
               (:equal Moving))
     :duration-min (min-length ?sitel ?site2)
     :duration-max (max-length ?sitel ?site2))

(DEFINE-AUTOMATE-CLASS TASK-NAVIGATE (?sitel ?site2)
  (:STATE begin
   :INTERNAL-EVENTS ((:run
                      :NEXT-STATE :executing
                      :TRANSITIONS ((:ON MOVING)
                                    (:OFF VAP-SITE ?sitel))))
   :EXTERNAL-EVENTS ((:stop
                      :NEXT-STATE :stopping
                      :TRANSITIONS ())))
  (:STATE executing
   :INTERNAL-EVENTS ((:return
                      :NEXT-STATE :end
                      :TRANSITIONS ((:OFF MOVING)
                                    (:ON VAP-SITE ?site2)))
                     (:execution_failure
                      :NEXT-STATE :failure
                      :TRANSITIONS ((:ON NAVIGATION-FAILURE))))
   :EXTERNAL-EVENTS ((:stop
                      :NEXT-STATE :stopping
                      :TRANSITIONS ())))
  (:STATE failure
   :INTERNAL-EVENTS ((:stop_on_failure
                      :NEXT-STATE :end
                      :TRANSITIONS ((:OFF MOVING)
                                    (:ON VAP-SITE CURRENT)
                                    (:ON VALID-PATH CURRENT ?sitel)
                                    (:OFF VALID-PATH ?sitel ?site2))))
   :EXTERNAL-EVENTS ((:stop
                      :NEXT-STATE :stopping
                      :TRANSITIONS ())))
  ...)
```

Figure 5: Automaton description

3.3 Task Refinement

Task Refinement transforms a task into specific actions that are adapted to the actual context. As an example, a motion task may be executed in different ways:

- a displacement using only dead-reckoning systems to guide the movement.
- a closed-loop motion using a perceptual feature of the environment (landmark tracking, edge following of a large object, rim following, etc.).

These two modes correspond to the execution of different *scripts*, associated with the task "move". Script selection is based on testing conditions using the acquired data. Scripts have variables as arguments, that are instanciated at execution time. The execution of a script is similar to the execution of a *program.*

3.4 Task Execution

Action execution are represented by *activities.* The execution of a script corresponds to a global activity. An activity is thus equivalent to the execution of a program, and is analogous to the notion of process in an computer operating system. A simple activity is the execution of a function by a module. An activity may cause the emission of requests to other modules, starting *children activities.* The module in the parent activity is then a client of the module in the child activity. An activity may be the parent of many children, but may be the child of only one other activity. A set of rules and mechanisms were

developed to create and manage activities. Two basic mechanisms are activity creation and message transmission between two activities.

At a given moment, the set of activities represent the functions being executed in the robot system. The activity structure is a tree with a parent-child relationship. The tree evolves while the robot is executing. An activity communicates with its (single) parent activity via "up-signals" and with its (eventual) child activities via "down-signals". The activity hierarchy is not predefined and depends on the current task.

Regardless of the specific processing it performs, an activity must be able to react to signals sent by its parent or its children. In particular, it must be able to react to asynchronous signals within a pre-defined bounded time delay.

An activity is also represented by a finite automaton. Its state changes are dependent on external or internal signals. Control flow between a mother and child activities is implemented as typed messages that cause a state change. Specifc mechanisms permit the propagation of a state change along the activity tree.

Important features of the notion of activity are:

- Activities provide for the management of a hierarchy of actions without imposing a fixed number of layers.
- All activities, regardless of level, may be treated in the same way.
- No assumptions are made about the nature of the inter-connections between activities: the activity hierarchy is *not* pre-defined. We might require that a "high level" activity starts and manages a "low level" activity at any level. This knowledge must however exist in the modules.
- All mechanisms for managing activities (starting, terminating, etc.) and the communication between them do not depend upon any programming language constructs; in this sense, they resemble part of an operating system.
- The concept of activity permits reactivity at all levels: at any moment, each activity in the tree structure is able to respond to asynchronous events.

Figure 6 represents an example of activity tree at one stage of the execution of a *go-to(location)* task. The "root" box represent the mother activity of this task, within the executive. The "monitor-11" box is the mother activity of a main child-activity (monitor-12) and of an associated monitoring activity (timer-200: a time-out for this task). When the corresponding event occurs, the main activity is stopped. "Monitor-12" is a monitoring activity also: the traveled distance should not exceed a given maximum (here 60 meters). Sequence in the "sequence-6" box means that the following children are to be executed in sequence (this defines a sub-activity block). "go-to-xy" and "exec-traj" are the two sequential activities. The first is the "go to" coordinate location, and the second is the trajectory execution phase within it. Other possible phases are "build-model" (environment modelling) and "build-traj" (trajectory planning). The last box "move-5" is the primitive action move being executed.

Figure 7 shows the currently configuration of the implemented modules.

Figure 8 shows the execution of the *go-to(location)* task in a laboratory experiment, where the robot discovers its environment and builds a model of it while it navigates to reach the assigned location.

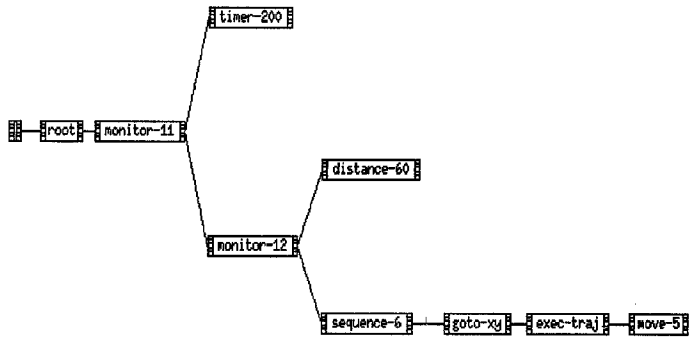

Figure 6: Activity tree during execution of a GOTO task

4 Conclusion

We presented in this paper a global approach to task planning and execution for intervention robots. Such robots are characterized by the fact that they have to be able to autonomously execute their actions in a partially and poorly known environment in order to accomplish missions and tasks specified and programmed by a human user. The robot have to interpret the tasks according to the context and its evolution, and to achieve autonomous execution.

The main interest of the plan interpretation as it is proposed in this paper is to maintain and update a complete history of the plan execution not only in nominal cases but also when a failure occurs. This is performed using automata which model tasks execution and their interaction with the plan interpreter. However other representations may be used. Indeed, we are working on an extension using on a rule-based system which should allow not only to model task execution but also combined effects due to the simultaneous execution of several tasks as well as domain dependent knowledge allowing to infer new transitions from starting from a set of observed events.

While it is still necessary to further deepen some aspects, the experimental results show that this task-level teleprogramming approach is sound and applicable.

References

[1] R. Alami, R. Chatila, M. Devy, and M. Vaisset, System architecture and processes for robot control. Technical report, Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), Toulouse (France), June 1990.

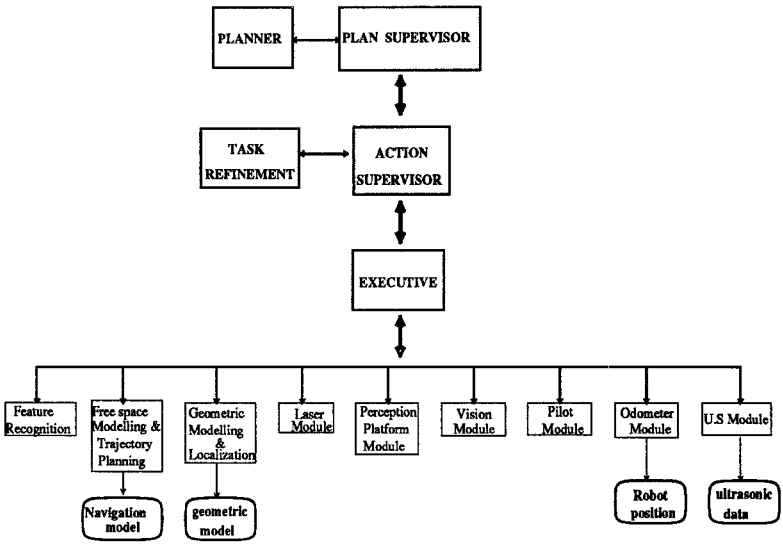

Figure 7: Implemented architecture

[2] R. Alami, R. Chatila, and P. Freedman. Task level programming for intervention robots. In *IARP 1st Workshop on Mobile Robots for Subsea Environments, Monterey, California (USA)*, pages 119–136, October 1990.

[3] Amine Mounir Alaoui. Raisonnement temporel pour la planification et la reconnaissance de situations. Thèse de l'Université Paul Sabatier, Toulouse (France), Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), October 1990.

[4] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[5] C. M. Angle and R. A. Brooks. Small planetary rovers. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 383–388, July 1990.

[6] L. Boissier and G. Giralt. Autonomous planetary rover (v.a.p.). In *IARP Workshop on Robotics in Space*, Pisa, Italy, June 1991.

[7] R. A. Brooks, P. Maes, and G. Moore. Lunar base construction robots. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 389–392, July 1990.

[8] R. Ferraz De Camargo, R. Chatila, and R. Alami. A distributed evolvable control architecture for mobile robots. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 1646–1649, 1991.

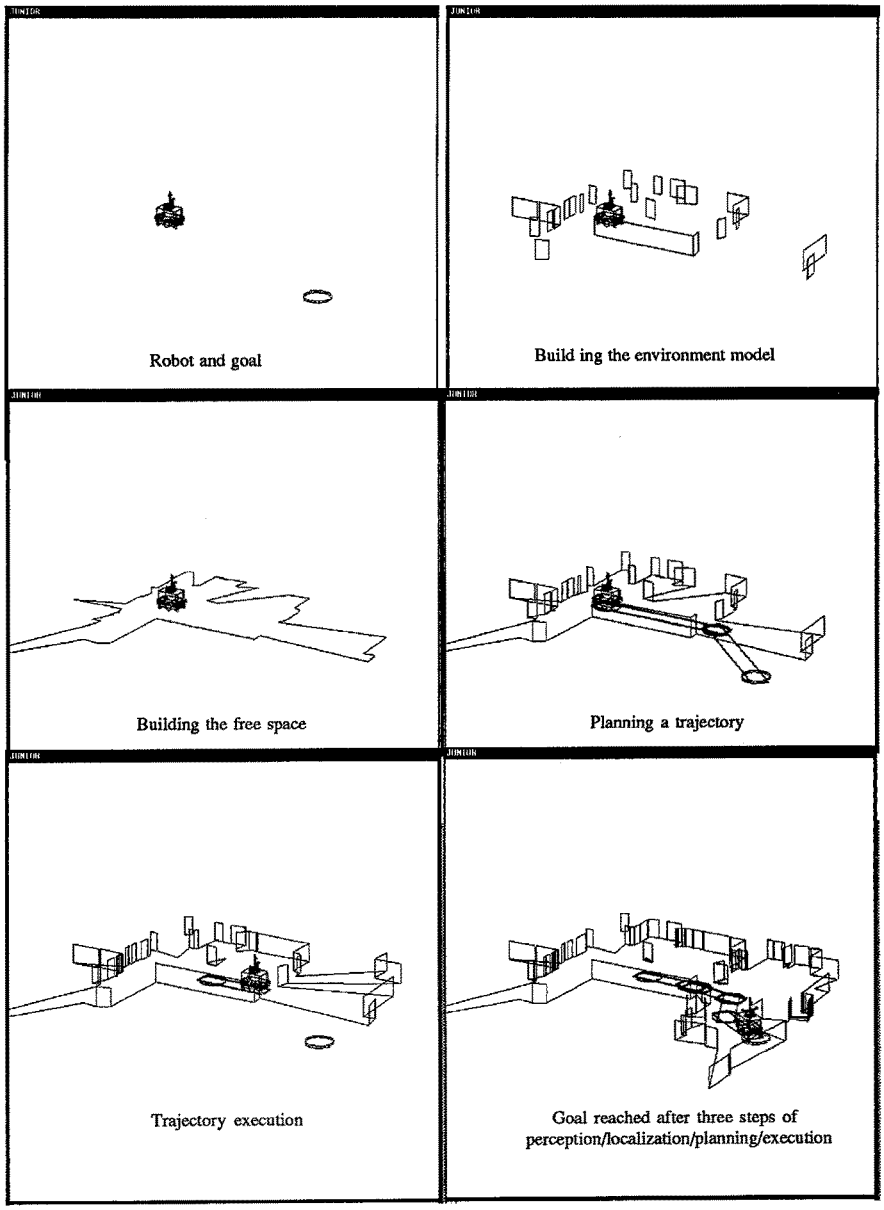

Figure 8: Experimental Execution of a *goto(location)* task.

[9] R. Chatila, R. Alami, and G. Giralt. Task-level programmable intervention autonomous robots. In *Mechatronics and Robotics I, P. A. MacConaill, P. Drews and K.-H. Robrock Eds, IOS Press*, pages 77 - 87, 1991.

[10] R. Chatila and R. Ferraz De Camargo. Open architecture design and inter-task/intermodule communication for an autonomous mobile robot. In *IEEE International Workshop On Intelligent Robots and Systems, Tsuchiura, Japan*, July 1990.

[11] L. Floury and R. Gable. The wireline reentry in deep ocean dsdp/odp holes. Technical Report DITI/ICA-91/172-LF/DB, IFREMER - Centre de Brest, July 1991.

[12] M. Ghallab, R. Alami, and R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles, editor, *Robotics Research: The Fourth International Symposium*. MIT Press, Mass., 1988.

[13] R. Laurette, A. de Saint Vincent, R. Alami, R. Chatila, and V. Pérébaskine. Supervision and control of the amr intervention robot. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 1057–1062, June 1991.

[14] F. R. Noreils, A. Khoumsi, G. Bauzil, and R. Chatila. Reactive processes for mobile robot control. In *International Conference on Advanced Robotics (ICAR)*, 1989.

[15] T. Sheridan. Telerobotics. In *IEEE Int. Conf. on Robotics and Automation (Workshop on Integration of AI and Robotic Systems*, 1989.