Integrated planning and execution control for an autonomous mobile robot

Ralph P. Sobek and Raja G. Chatila

Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S., 7, avenue du Colonel-Roche, F-31077 Toulouse Cedex, France

We present a distributed hierarchical planning and execution monitoring system and its implementation on an actual mobile robot. The planning system is a distributed hierarchical domain independent system called FPS for Flexible Planning System. It is a rule based plan generation system with planning specific and domain specific rules. A planning solution to the 'Boxes and Wedge' Problem[1] is presented.

The Robot Control System (RCS) operates and monitors the robot in the real world. In order to allow real-time responses to asynchronous events (both internal and external), RCS consists of a rule-based decision kernel and a distributed set of sensor/effector monitors. RCS contains an execution model and may authorize local corrective actions, e.g., unexpected obstacle avoidance during execution of a trajectory. RCS also generates status and failure reports through which the PMs inform the different decision subsystems as to the robot's state and current capacities. The failure reports help the RCS and planners in correcting/replanning a plan that has aborted. An illustrative example of system behaviour is to be presented.

Key Words: plan generation, execution control, mobile robots, distributed systems.

1. INTRODUCTION

Research in the field of robotics is centred on the concepts, methods, and systems which make up third generation robots, i.e., machines that exhibit an 'intelligent connection of perception and action'[2]. Furthermore, the direction of planning and problem solving research in the 80s has been towards distributed problem solving[3]. The issues that come up in a distributed environment involve communication, representation, and control. Communication comprises both knowledge sharing and a conciseness of information abstraction so as not to overload bandwidth constraints. Formerly the robot was the only agent and therefore the data base representation was assumed to be correct. Now, robot models and their belief systems approach more closely, though still simplified, what is called the 'real world'. In terms of control, there are the problems of coordination of asynchronous processors and dynamic reorganization of resources in order to carry out particular tasks.

Autonomous Mobile Robots represent an ideal paradigm for third generation robots. In a real-world environment, a mobile robot must have multiple and varied performance levels in perception, planning, and execution control. For robustness, efficiency, economy of operation, and timely system response, distributed decision-making is necessary[4].

A system does not have to be multi-agent in order to be considered distributed. A distributed multi-processor system allows research on distributed processes in cooperation. The notions of communication, coordination, and error recovery are the same as in the general case. However, there are no hostile entities on-board the robot, just possibly in the environment.

A distributed hierarchical domain independent planning system called **FPS**, for **F**lexible **P**lanning **S**ystem, and an extensible rule-based execution control system called **RCS**, for **R**obot **C**ontrol **S**ystem, are presented. FPS is a rule-based planning system. FPS contains planning specific and domain specific rules. It interfaces with other planners and decision modules (e.g., Navigation Planner and Route Planner) as well as RCS. The presentation of FPS is terminated with a plan generation example: the 'Boxes and Wedge' constraint problem of Fikes, Hart, and Nilsson[1].

RCS is a distributed control system that interprets directives from the user or FPS, and it reports on the state of its distributed net of monitors and processing modules. An illustrative execution protocol of RCS together with FPS is presented. The entire system undergoes tests in a real-world robotic environment (the HILARE project[5]).

2. DOMAIN-INDEPENDENT PLANNING

In general, a problem is a situation for which an organism (i.e., a program) does *not* have a ready response. Problem solving involves (1) the sensing and identification of a problem, (2) formulation of the problem in workable terms, (3) utilization of relevant information, and (4) generation and evaluation of hypotheses. A planner is a program that attempts to deal with points 2 through 4. In this section we present **FPS**, a rule-based plan generation system.

In a real-world environment, a multitude of error situations may arise. Besides correction of planning errors, a planner must try to determine when an error is recoverable or when replanning is necessary. Planners

Table 1. Planning node entries

1.	Goal Pattern
2.	Goal Instantiations
3.	Preconditions/Enablements
4.	Continuation Conditions
5.	Post-Conditions (including goal pattern)
6.	Constraint Conditions
7.	Parent
8.	Children associated by each decomposition
9.	Importance
10.	Allocated Cost Estimate
11.	Cost Expended
12.	Success Rate for each Post-Condition
13.	Error Recovery Handles:reason, source, locally recoverable
14.	Operator List
15.	Script

must be able to select processing strategies appropriate to each situation encountered and be able to handle complex goal descriptions. In such an environment, what is a plan? It has to be a flexible and extensible structure which permits quick adaptation to unexpected situations. It must permit goal-directed as well as data-directed processing. Goal simultaneity and interaction must be verified during planning and before attempted execution.

FPS is rule-based principally for the following reasons. Production System (PS) rules allow for a neat solution to the frame problem when we use the STRIPS assumption[6] in that all updates in the model are done explicitly through rules. Since rules can react in one PS cycle, PSs can adapt to new situations very rapidly, the rules acting like demons. In planning or execution monitoring this fact allows the system to deal with unexpected/ serendipitous situations. In addition, rule interactions may permit parallel searchs for a best solution or may allow rapid responses to recognized problem situations (e.g., planning goal conflicts). In a PS the addition of knowledge is incremental. Therefore, the evolution of our robotic environment will be easily characterizable to FPS. Also, in the future our use of a PS architecture will permit FPS to organize and generalize the plans that it has created in the form of new rules.

Some may say that PSs are inefficient. It has been shown that by the use of compilation strategies significant gains in execution speed are attainable[7]. FPS is implemented in the PS1 production system language[8,9]. It does not have to sacrifice efficiency for flexibility in its representation since PS1 is a compiled PS. Rule patterns are compiled into a parallel-match tree similar to but more general than OPS[10]. The advantages of PSs have been adequately described[11]. Some planners and expert systems have opted for a frame-based approach[12] and it should be noted that there is a similarity between PSs and frame-based systems.

2.1. Planning structure

We present a robot planner (FPS) which deals with the dynamics of a plan[13]. FPS has in its ancestry the planners STRIPS[14], NOAH[15], and especially JASON[16]. It generalizes these planners in representation and flexibility. FPS is used in a real-world mobile-robot paradigm. Superficially, FPS is similar to NOAH and its generalization JASON in that they are all conjunctive goal-oriented planners. Where plans for NOAH consist of a directed graph of procedures (procedural net), in FPS plans may be described as a directed graph of processes. Each process contains its state in a structure called a 'planning node' each with its associated goal (see Table 1). A process characterizes the dynamics of a plan step while the planning node represents the data aspects. A process gets its node's entries filled from three principal sources: (1) from the parent node, (2) when an operator is selected for a node, or (3) by the executive, critic, task communication, and scheduling rules. The processes are managed by a tasking executive which arranges the processes on a priority agenda taking into account for each process the importance, success, cost expended, and estimated allocated cost. The inter-process coordination and high-level conflict resolution knowledge are called planning specific and are represented in rules. For example:

If all sibling* children of a process have achieved its preconditions

then check if the process can be decomposed into sub-processes**.

Simple goals in FPS may consist of a relational predicate, e.g., **(INROOM BLOCK1 RM3)**, its negation, or the application of a specific operator to a goal. Compound goals may be conjunctions or sequences of goals. Compound goals let FPS search for possible conflicts whereas sequences specify an explicit required ordering of the goals involved.

Planning involves iteration of node expansion with plan criticism. Criticism may start as soon as a node is expanded, which eases a shortcoming of NOAH. NOAH could only apply its critics at the end of each expansion cycle. The critics in FPS are considered a major part of the planner's 'planning executive'. They contain knowledge that is relevant to the entire planning process, i.e., both planning specific and domain specific. The current critics are 'Eliminate Redundant Preconditions', 'Resolve Conflicts', 'Use Existing Objects', and 'Resolve Constraint Violations'†. Concomitant and overlapping with critics are heuristic rules. For example:

H1. **If** multiple choices are possible **then** select one which minimizes cost, effort, or distance

H2. **If** robot moves an object **then** it should not block a door

H1 is a general rule whereas H2 is domain specific.

The planner presents a model of the robot's possible actions within its environment; it currently does not model the robot's interactions. There is no representation for other purposive (goal oriented) organisms or causality other than the robot's. The possible actions are modelled by operators.

* The sibling processes are all children of a process which are conjoined by the same goal instantiation of the parent. The parent process might have mulitple instantiations for its goal description and then would have a disjunctive group of siblings for each instantiation.

** All rules presented in this article are English language simplifications of FPS rule-sets.

† The critics 'Eliminate Redundant Preconditions', 'Resolve Conflicts', and 'Use Existing Objects' come from Ref. 15

GOTOROOM:

argument:	?r
preconditions:	(INROOM ROBOT ?r2)[a]
	(CONNECTS ?d ?r ?r2)[b]
script:	(SEQ (GOTODOOR ?d)
	(GOTHRUDOOR ?d))
post-conditions:	(INROOM ROBOT ?r)
	(NOT (INROOM ROBOT ?r2))

[a]- primary result (goal condition)
[b]- static data

Fig. 1. GOTOROOM operator specification

2.1.1. Operators

Operators are dynamically selected for each planning node; no *a priori* connection between operators and goals exists. Associated with operators are preconditions (environmental context), continuation conditions, post-conditions, and constraints*. The operators are ordered in a specialization/generalization hierarchy. Operators may have scripts which specify how they should be reduced; the scripts may define conditionals, parallel paths, goals, constraints, and sub-operator applications (see Table 2). If they contain subgoals then a sub-process will be created for each subgoal. Otherwise, the scipt will be checked by 'critics' against the surrounding plan structure.

An example operator is **GOTOROOM** (see Fig. 1). Given a room **?r** as argument, if the robot is in an adjoining room then it will try to apply sequentially the two sub-scripts **GOTODOOR** and **GOTHRUDOOR**. These latter two scripts are subordinate to **GOTOROOM** *only* in the current node's dynamic context: a different call sequence would create another hierarchy of goals and operators. FPS would fan out from the goal state, e.g., (**INROOM ROBOT RM1**), using the connectivity graph represented in the predicate **CONNECTS**, provided by the Routing Planner, until it finds the current state.

After a route is found the two subordinate scripts are tried in order to assure that the robot can get to and through the door. If there are multiple doors to a room, each would cause two parallel descendant nodes to be created. The Route Planner is further described in § 4.1.

An operator's goal is specified to the system as the primary post-condition. The above operator can also be invoked to get the robot out of a particular room. Each time that an operator succeeds with respect to a goal its correspondent level of importance or competence is rewarded.

The fact that FPS is rule-based permits an interplay between the algorithms and heuristics, both instantiated as rules, not found in more traditional programming environments[11]. Search strategies allowed in FPS include both breadth-first and depth-first. Depending on the situation, the rule programmer may add to FPS heuristic rules which would, for example, allow FPS to remember efficient routes once found.

2.1.2. Constraints and planning variables

A relational predicate may also contain 'planning variables' similar to those developed by the first author[16] and those introduced in SIPE[17]. Their use is very similar to the constraint expressions introduced in MOLGEN[18,19]. These variables do not actually contain values; they may specify restrictions upon the allowed values (bindings) that their positions in a predicate may take, e.g., in the predicate (**INROOM ROBOT ?RM**) the variable **?RM** may specify a number of possible instantiations for the predicate. The planning variables serve three functions:

(1) an alternative to disjunctive goals with similar disjuncts,
(2) a method for the postponement of decisions, and
(3) constraint expressions.

Planning variables are inherited through the goal – subgoal hierarchy in the planning network: the PRECONDITIONs of goals. They are globally known throughout the planning process. A unique feature of the planning variables not found in the recent codesignation constraints* of Ref. 20 is that a plan can be considered complete while still having free, or partially specified, planning variables which will be resolved during the execution phase (see § 2.2). When a planning variable is free in a plan expression (see Table 2) the expression is considered *non-constrained.* If the planning variable has associated constraint expressions, then its participation in a plan expression *constrains* the said plan expression. Planning variables permit FPS to automatically generate macro plans (in the spirit of Ref. 21).

Constraint expressions may be attached to goals, planning nodes, and operators. Constraints are similar to goals except that they must be satisfied for the preconditions of a possible operator as well as during and after the operator's execution. They are taken into account in the node expansion procedure and can cause the insertion of additional plan steps *before* a node. For the moment, the following constraint expressions are allowed:

1. Equality: (equal ⟨var⟩ ⟨element⟩)
2. Inequality: (not-equal ⟨var⟩ ⟨element⟩)
3. Inclusion: (member ⟨var⟩ ⟨element⟩$^+$)
4. Exclusion: (not-member ⟨var⟩ ⟨element⟩$^+$)

where an ⟨element⟩ can be either another planning variable or a constant value.

* All the above are necessary for planning. These and other informations, such as default monitors, are associated also with RCS (cf. § 4.1).

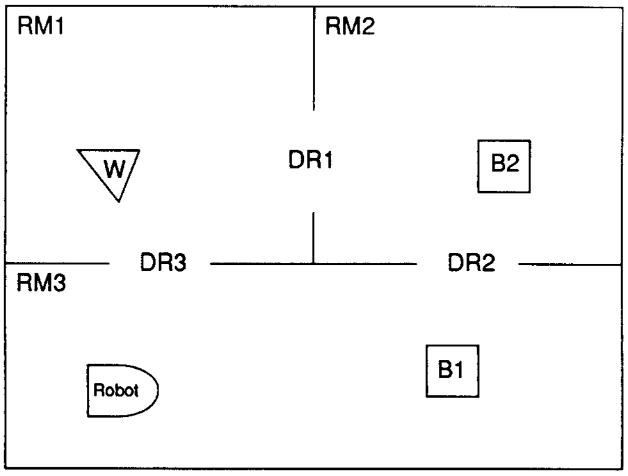
* Which they resemble very much.

Table 2. Planning script grammar

```
<script> ::= <step>+

<step> ::= (SEQ <step>+)
         | (AND <step>+)
         | (OR <step>+)
         | (IF <step> THEN <step> { ELSE <step> })
         | (WHILE <pattern> <step>)
         | (UNTIL <pattern> <step>)
         | (PARALLEL <id> <step>+)
         | <simple-step>

<simple-step> ::= (GOAL <pattern>)
                | (ASSERT <pattern>)
                | (DENY <pattern>)
                | (VERIFY <pattern>)
                | (CONSTRAINTS <expression>+)
                | (APPLY <operator> <pattern>)
                | (WAIT <id>)
                | (SIGNAL <id>)
```



Goal: (IN ?BOX RM1)

Constraints: (NOT (AND (IN ?BX ?RM) (IN W ?RM)))

Fig. 2. Boxes and wedge constraint problem

2.2. A planning example

In this section will be given an example problem which shows some of FPS's capacities; in particular, the use of constraint expressions and planning variables is shown. In the 'Boxes and Wedge' problem (first described in Ref. 1), the goal is to have an arbitrary box in **RM1** with the explicit constraint that at no times can a box occupy the same room simultaneously as the wedge (see Fig. 2). Much of the variability in the resulting high-level plan depends upon the operator definitions – the ordering of critics' applications to the plan – and how the planning variables acquire values and resolve their conflicts. Therefore the plan described in this section is not to be construed as the *only* one that the planner can generate for this problem, but as one in a family of equivalent plans, after renaming of variables and equivalent expressions. The solution presented in this section is based on one first presented in Ref. 16.

The operators used in this example are **PUSHTOROOM**, **GOTOROOM**, **GOTOOBJECT**. For this example the preconditions for the operators are as follows:

1. for **PUSHTOROOM** the only precondition is **(NEXTTO ROBOT ?OBJ)**,
2. for **GOTOROOM** the precondition is **(INROOM ROBOT ?RM)** (see Fig. 1), where **?RM** is a room connected to the goal room, and
3. for **GOTOOBJECT** the precondition is **(INROOM ROBOT ?RM)**, where **?RM** is the same room that contains the desired object.

The initial plan, which consists of a single planning node, (shown in Fig. 3a) is created from the goal description input by the user. The only critic applicable to this plan is 'Resolve Constraint Violations'; it decides that for a box to be in **RM1** (denoted by the planning variable **?B** in Fig. 3) the wedge must *first* be removed from **RM1**. The initial plan, after application of criticism, is shown in Fig. 3b. Expanding this plan to the next lower abstraction level produces the plan shown in Fig. 3c. Note that the node **(NOT (IN W RM1))** of the Level 1 plan has been transformed to a logically equivalent node in the Level 2 plan with a constraint on the planning variable **?RM1**, which signifies that **?RM1** cannot be bound to **RM1**.

Because no critics are applicable, the plan is then expanded to Level 3 (see Fig. 3d). Constraints are added to two nodes of this plan:

(1) in order that the global constraint not be violated, the room into which the wedge is to be pushed (i.e., **?RM1** is not allowed to have any box (i.e., **?B2**) in it, and
(2) the robot must be in the same room as the box to be pushed (i.e., **?B**).

The constraint violation critic determines that the first constraint, i.e., **(NOT (INROOM ?B2 ?RM1))**, must be achieved before the preconditions of pushing the wedge into the room denoted by **?RM1**. The resulting plan (see Fig. 3e) contains a disjunctive sub-plan for removing either of the two boxes, **B1** or **B2**, which would violate the global constraint if not moved*. In addition, the allowed rooms for these boxes are also constrained.

The disjunctive sub-plan is further expanded at the next iteration of the plan expansion process. The critic 'Eliminate Redundant Preconditions' combines the two goals **(NEXTTO ROBOT B1)** and **(NEXTTO ROBOT**

* It should be noted that other possible node expansions are possible, e.g., a conditional planning node, but the system currently only establishes an **OR** plan. Furthermore, the planner could have alternatively created a single node, **(INROOM ?B2 ?RM5)**, with more complex constraints on **?B2** and **?RM5**; but since the constraints were separable into two cases, one for each box, the disjunction was selected – heuristically deemed a less complex representation.

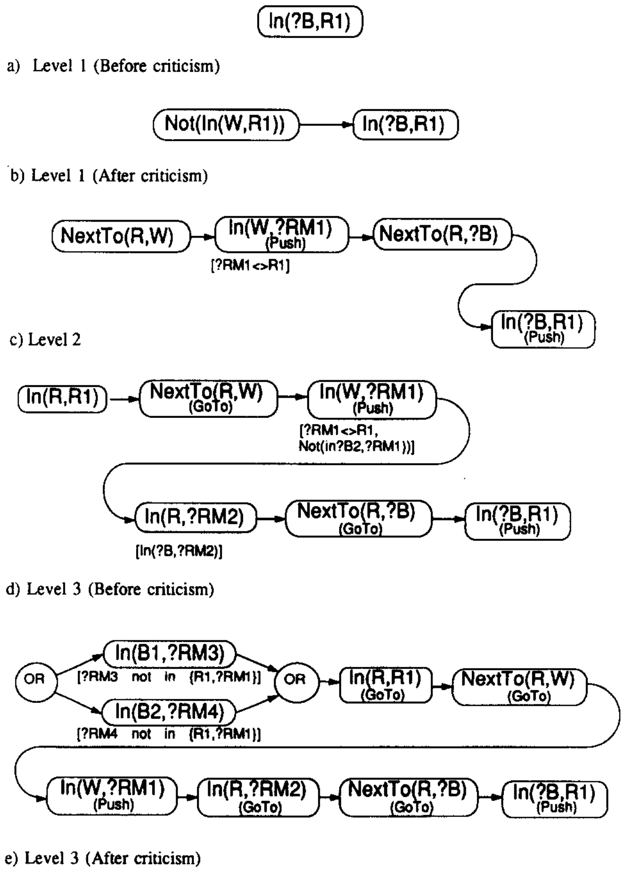
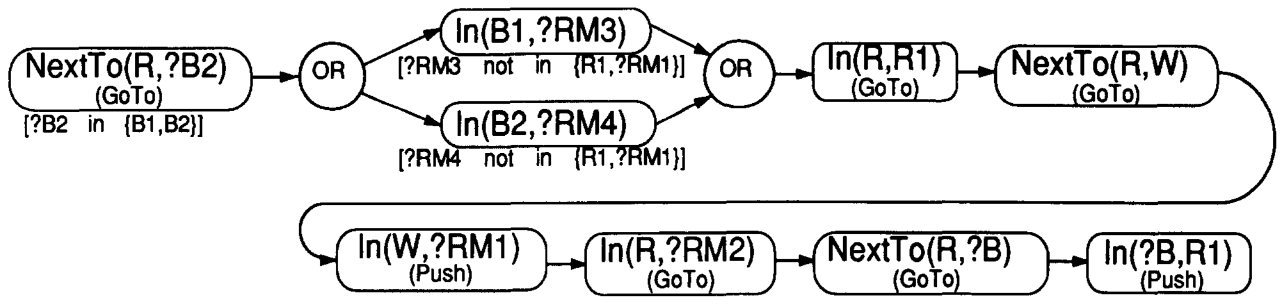
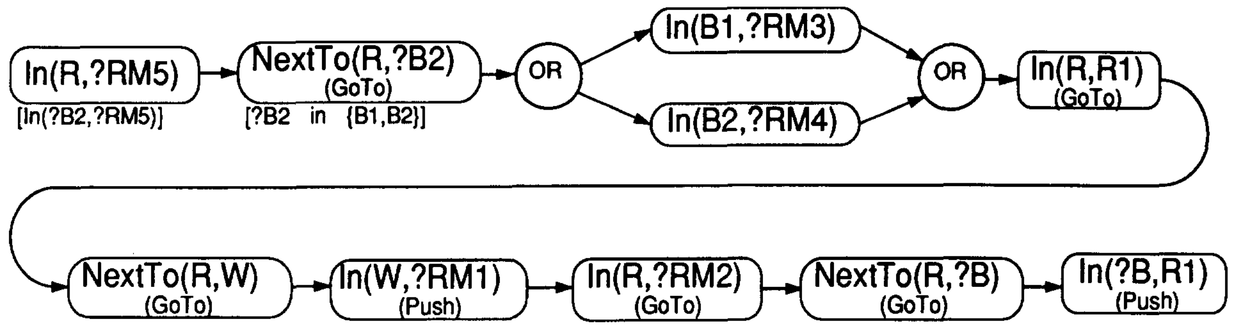


Fig. 3. Boxes and wedge solution plan – part 1



f) Level 4



g) Level 5

Fig. 3. Boxes and wedge solution plan – part II

Table 3.

?RM5 = ?RM2	?RM5 = ?RM3
?B2 = B2	?B2 = B1
?RM4 = RM3	?RM4 = RM2
?RM1 = RM2	?RM1 = RM3
?RM2 = RM3	?RM2 = RM2

B2) into the single goal (**NEXTTO ROBOT ?B2**) with the constraint **?B**∈{**B1, B2**}, which results in the plan in Fig. 3f. This plan is expanded to create the final plan shown in Fig. 3g. The new planning variable **?RM5** is constrained such that it is bound to the room containing the box referenced by **?B2**; otherwise, no conflicts arise in the final plan.

This final plan allows flexibility for optimization by including *all* possible linear plans for solving the 'Boxes and Wedge' problem. The planner is allowed to optimize the plan by specifying the free planning variables, or it may pass this information (semi structured) to the execution phase and the Robot Control System. For example, **?RM5** has two allowed bindings from which are derived the constraints of the other planning variables (see Table 3). One can readily see that the decision of which box to push into **RM1** (*viz.*, the bonding of **?B**) can be postponed until the moment that the robot is in **?RM2**; neither of the above sets of possible bindings has any effect on the binding of **?B**.

In summation, FPS produces a general and flexible plan. Since the planning process descends from higher levels of the abstraction hierarchy to lower levels, the decisions are made in a hierarchical order, thereby eliminating the need for backtracking during *this* instance of the planning process. Only five steps (or iterations) were necessary in order to solve this problem.

3. THE ROBOT CONTROL SYSTEM

3.1. Requirements for a real-time controller

The execution control of a plan in the real world is a very important issue that was not enough addressed by the AI literature. What distinguishes planning from execution control is that in the former a coherent planning structure is established, whereas in the latter the necessary verifications of coherence must come from the real-world environment. Note that a large part of the representation for both planning and execution monitoring *must* be the same in both in order to facilitate their communication and sharing of models.

The planned actions and motions are to be carried out in the real world where the robot is not the unique actor, and under time constraints. The inadequacy between the models and reality should also be taken into account.

The control system has to achieve two important objectives:

- It has to operate the robot, sequencing the elementary tasks, the operators and the processing modules, and monitoring the execution of actions.
- It has to ensure *reactivity* or *real time behaviour*. This means that external or internal asynchronous events should be taken into account at their own time scale, and a course of action decided for when such events occur.

The controller should be able to check if the planned actions can be executed, given the actual environment and robot state (plan validation). It has to detect failure and error situations.

One important problem to be handled by the controller is serendipity, or detection of fortuitous events relevant to the current plan or goal. At some point during planning and/or execution it may arrive that either one of the robot's sensors registers something or the user enters some new data. The problem is to filter the incoming information into some usable and timely form, which does not disallow *multiple* forms for this information, to detect the useful information, and to use it to save the planner innumerable steps in the current plan, or to shortcut the execution of a task. An open problem is the efficient recognition, selection, and use of important information in a data stream.

As such, the problem could be complicated to involve a hierarchy of goals such that many goals could be triggered. For some goals the new information would be of secondary importance, whereas for others it would be urgent, but some deduction would be necessary. In any case, the controller architecture should provide for the detection of the relevant information.

Control for mobile robots was addressed by several authors. Execution control was first mentioned in relation to the execution of plans produced by STRIPS for Shakey[14,22] introducing the *triangle table* formalism to express the plan. This formalism permits the explicit access of operator preconditions so that the plan could be validated with respect to the current world model and operator applicability verified. But execution control is also to be related to the underlying software (and hardware) architecture.

First oriented towards sensor data fusion, the CODGER architecture developed at CMU[23], addresses also the problem of control. CODGER is based on a central blackboard-style database (called whiteboard) that enables the system's modules to communicate and synchronize.

The purpose of the architecture proposed by Brooks[24] is not plan execution control but to ensure reactivity. It is based on implementing layers of control achieving simple to more complex behaviours. Each layer is composed of a number of asynchronous processes that can be subsumed by higher-level modules.

In the system presented in this paper, execution control uses the same planning structures* to establish when failure should be reported or error recovery should be intiated. A failure situation is detected when an operator's execution can no more be carried out because of an external or internal event, whereas an error situation is detected when there is a discrepancy between an operator's expected possible outcomes and the real-world responses[25]. These discrepancies are analysed by execution-error critics which determine if the plan is still valid, or whether the error has a fixed solution (e.g., local planning), or that replanning will be necessary.

To enable local planning by the controller, an *execution model*[26] is transmitted to it by the planner, together with the plan to be executed. This execution model contains parts of the world state as it is supposed to be during the execution and after the completion of each plan step. It includes world model elements, relationships and predicates that are directly concerned with the current plan (e.g., topological relationships, some objects statuses). The execution model also contains plan mending flags for each plan step that authorizes or not local corrective actions by the controller in case of failure of the plan step, for instance, local avoidance of unexpected objects without global path or task replanning.

* FPS and RCS share a number of symbols in the robot system.

3.2. Controller structure

3.2.1. Robot system architecture

The mobile robot HILARE system is composed of a number of modules realizing its basic functions. This functional decomposition is mainly oriented towards the achievement of mobility. It contains the functions enabling the robot to perceive and model its environment, to manipulate these models for its position reckoning and for task and motion planning and execution.

The robot system is thus decomposed into a partial hierarchy of Processing Modules (PMs) and Decisional Modules (DMs). To date, three DMs are implemented: routing (RP), navigation (NP), and task planning (FPS). PMs shown in Fig. 4 are those used for navigation. Others, that would be 'application specific', such as manipulation or loading tool control, could be added to the structure.

The Robot Control System (RCS), operates the robot, sequences the PMs according to the task to be executed, and monitors the actions, external situation, and internal status. The PMs include the low-level execution operators which reflect the robot's capacities.

To meet the requirements of real-time operation and decisional capacities, we propose the following structure for the robot controller[27] (see Fig. 5):

- a decisional kernal, written as a rule-based system,
- and a set of distributed monitors.

The execution operators are mirrored in RCS's rule-base as well as FPS's planning operators. Their general syntax is the following.

$$\langle\text{execution-operator}\rangle\ \langle\text{arg}\rangle^*\ [\langle\text{optional-arg}\rangle^*]$$

$$\langle\text{optional-arg}\rangle ::= \langle\text{flag}\rangle$$

$$\mid \langle\text{identifier}\rangle = \langle\text{value}\rangle$$

3.2.2. Decisional kernel

The decisional kernel is a rule-based system containing knowledge on robot operation and failure diagnosis[28]. Rule systems enable granularity of knowledge and are very suited to expressing operating and diagnostic

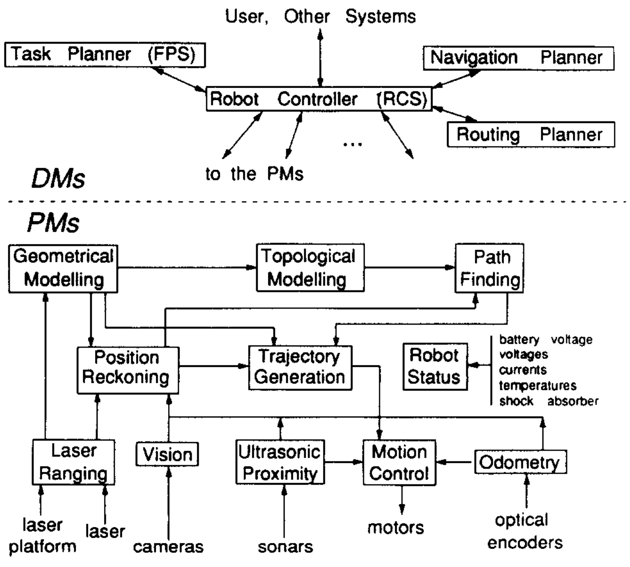


Fig. 4. Robot system architecture

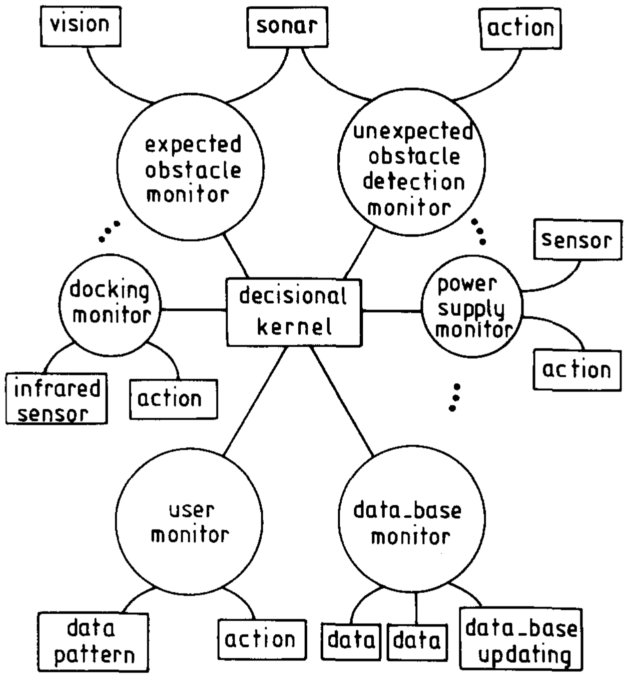


Fig. 5. Robot controller structure

knowledge. The controller's rules are simple and can be written in propositional logic. Indeed, as far as the control does not include actual planning, and since the planner provided an execution plan, the state space is finite, and the possible failures are also known beforehand. Therefore, this rule system can be precompiled into a decision tree. This obtains efficiency of operation while flexibility of expression is kept. The kernel instantiates, activates, and inhibits, the monitors.

In addition, the capacity of a production system to react to asynchronous events is well documented[11]. The kernel is able to decide for local plan mending or modification, according to the instructions given by the planner through the execution model.

In this case, the interactions between the planner and the controller are very important; in order to address the serendipity problem in general, a planner should be permanently replanning from the current state that is updated by the controller. In the current state of development of planners, this not a realistic solution*, given the rather long response time of planners. Thus, the controller handles serendipity by *programming* surveillance monitors that detect events that are related to the plan being executed and which react to their occurrence. The monitors are programmed by the kernel according to instructions in the execution model.

To illustrate the problem, suppose that the robot is exploring its environment, and that it is traversing some reasonably known room. During a particular trajectory an unexpected ultrasonic sensor reading is made. The robot may go automatically into 'Local Obstacle Avoidance' mode if it is enabled in the execution model. Meanwhile, further perceptions are made and a local geometric model is built. Now let us suppose that the reason for this exploration was the command, 'Find a red block'. In this case, if the readings could come from a red block, at an early level of interpretation, then it becomes urgent that the robot decide to stop and analyse further the unknown object.

* And maybe *not* a desirable one.

3.2.3. Monitors

The monitors enable real-time reaction to asynchronous events. They are semantically equivalent to some of the kernel's rules (*viz.*, the surveillance rules), and are instantiated by them. The monitors are simple rules that react immediately to some events, and notify the kernel of the event occurrence. They are distributed among the robot's modules. Monitor syntax is:

$$\text{MNTR}\ \langle\text{condition}\rangle^{+} \Rightarrow \langle\text{action}\rangle^{+}$$

For instance, they enable monitoring of robot actions through the sensors. They achieve a programmable closed-loop between the PMs. Their condition side is either sensory information or data patterns in the execution model, and their right-hand side is an action to be executed immediately. The decisional kernel is then informed of the monitor's execution. Several levels of monitoring can be defined, ranging from guarded commands to dynamic modification of the robot's behaviour (navigation mode selection, sensor use, etc.). Logical operations on the monitors (i.e., AND and NOT) are possible. Several kinds of monitors together with examples are next listed.

A. Action and plan monitoring

This involves guarded commands, error and failure conditions, etc. An example guarded command is 'Move until 6 cm from ?X'. Error conditions are related to the expected outcome of an operator after or *during* its execution: the operator's termination conditions are monitored, but also conditions that should hold during all or part of its execution. One example is the expectation of an encounter with an object in the robot's environment. The robot could then trigger either the verification of the object's position or correction of the robot's own position reckoning (see Ref. 29). Here a comparison is made between some local model and the expected state during an action. A failure condition would be to find the door obstructed (e.g., closed) when the action executed was **GOTHRUDOOR**.

Let us consider the monitoring of knowledge updating. The robot should be able to take into account knowledge updating by the user or other external systems, both during planning and during execution. We will focus on the second case, the first already discussed above.

While the robot operates, the execution model is used by the controller to dedicate monitors to the surveillance of the status of items that are relevant to the plan under execution. For example, if the plan specifies that the robot has to cross door D1, then the status (open or closed) of door D1 is monitored in the world model, even when D1 is not in sight, because this model might be updated by an external system.

When the updated information generates a failure, e.g., door D1 is found closed instead of being open, the controller is in some cases able to mend a plan. For example, the controller may select an alternate route (if already given, or by a call to the Route Planner) without the need to replan the whole task. The case of 'positive' updating, i.e., information that modifies the world model by making possible a better plan (discussed in § 3.2.2), requires generally replanning, except if the information comes from a free planning-variable binding or is related to a monitored item whose contingency has already been planned.

B. Action (or plan step) chaining

The termination conditions and preconditions of each operator (plan step) are monitored, so that the interruption of the current step is possible whenever the preconditions of the next one are true. Execution of the latter is begun as long as the fulfilment of the former is not necessary (i.e., its remaining termination conditions are not necessary for any following plan step or the final goal). For example, in the early detection of a workstation the docking procedure consists in performing some predefined manoeuvers. In the usual mode of operation, the robot navigates to a given point in the workstation approach zone*, defined as the workstation entry point, and then begins to dock. However, for several reasons, such as odometry drift or change of the initial trajectory during local obstacle avoidance, the robot could be in the vicinity of the workstation before reaching the entry point. This is also detected by a monitor that compares the sensory readings (sonars, ...) with a specified model pattern. The decisional kernel then may update the sequence of actions in order to generate a new trajectory leading directly to the entry point.

C. Robot operation monitoring

Surveillance of robot operation consists of monitoring events that may jeopardize robot safety or plan completion and that are known to be possible during robot execution, but are asynchronous with respect to the plan. Examples include: unexpected obstacle detection, need to charge battery, need to reckon position because of odometry drift.

In the case of unexpected obstacle detection and avoidance, before robot motion begins, monitors are programmed by the kernel in order to detect any object lying on the robot's trajectory by the ultrasonic system[30]. When the distance is under a given threshold, the robot stops. The monotor can be expressed very simply by the rule:

If (RANGE sonar_i) < threshold_i **then** (STOP)

The decisional kernel then proceeds to analyse the situation by scanning with the ultrasonic sensors, building a local model of the environment, and if local plan mending is enabled in the execution model, it selects an avoidance trajectory, taking into account the known surrounding environment model. It programs new monitors that will be operational during the local obstacle avoidance to detect termination conditions on success or failure. When these monitors are triggered, and in case of success, the previous action is resumed, unless other events occur. An example of local obstacle avoidance is given in Fig. 6. The avoidance trajectory in this example is a circle that is updated by ultrasonic sonar readings[30].

* Fixed objects such as doors, workstations, etc., have defined fixed zones of approach from which the robot's processing modules can perform the necessary manoeuvers in order to execute the desired actions, go-thru-door, dock, etc. Other objects have relative approach zones which depend on the purpose of the encounter, e.g., to push a block.

In the case of robot position reckoning during normal operation, odometry readings provide the robot's position. But as the robot moves, the uncertainty on these readings increases. Therefore, a monitor compares the estimated position error (that is a function of the travelled distance) with a threshold, and sends a message to the kernel when it is exceeded. The kernel decides to reckon its position by matching the known model with a local perception (using vision or laser ranging, see Ref. 31 for details), which depends on the context: if the robot is performing obstacle avoidance or docking, this position reckoning is inhibited by some of the kernel's rules.

Similarly, in the case of robot refueling, the battery voltage level is also monitored and when it is below a given threshold, battery power supply becomes an imperative task. Again, the execution of this task is dependent on the context as expressed by some kernel rules. Furthermore, external (or internal) condition changes impose the use of different sensors, speed modification, navigation mode selection (e.g., wall following).

D. Nondeterministic execution of actions, and execution at a given instant

When given conditions are met they may trigger specific actions. An example would be to take a picture at a given location during motion. This enables parallel execution of actions.

4. AN EXAMPLE EXECUTION PROTOCOL

We present here an annotated sample execution protocol of a robot docking scenario given in Fig. 7. The robot is to

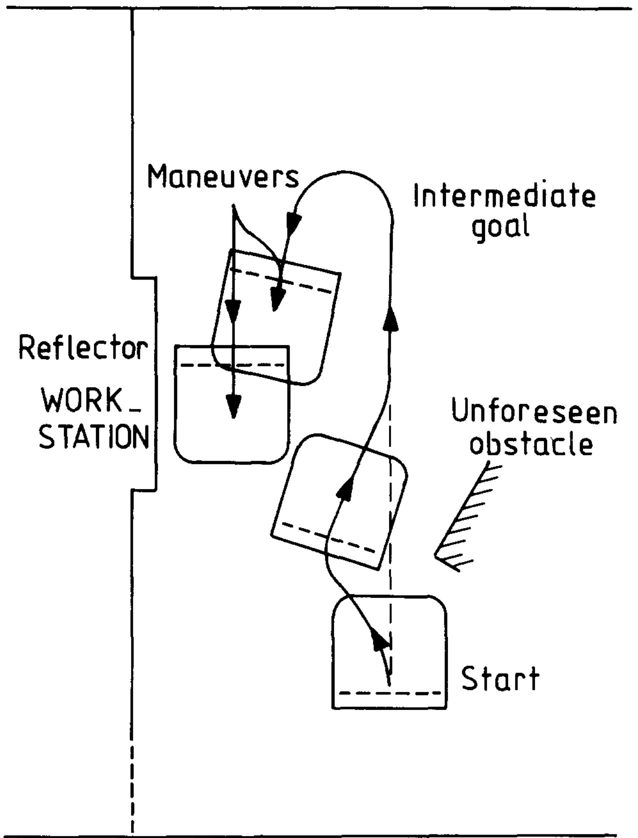


Fig. 6. Local obstacle avoidance

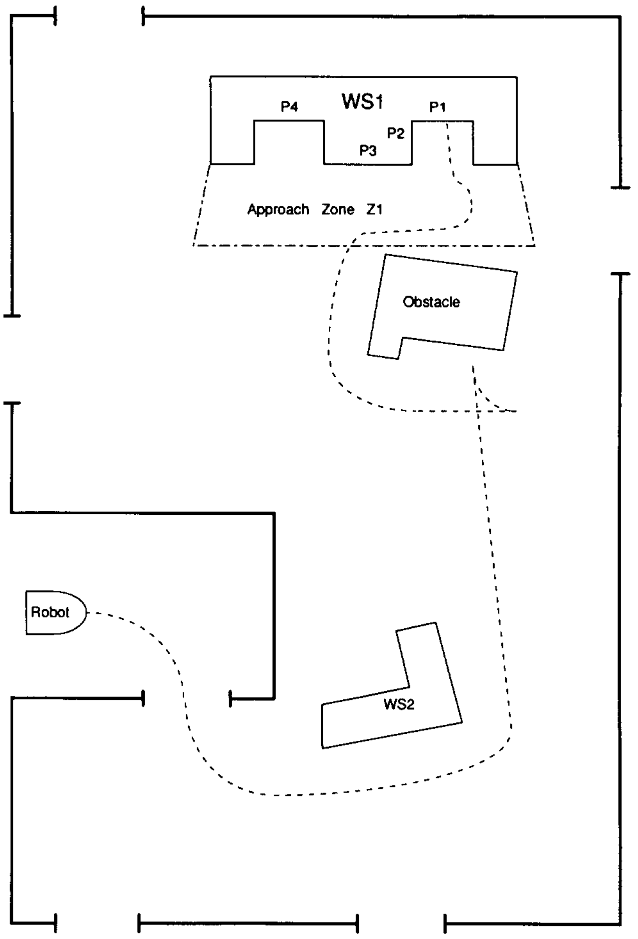


Fig. 7. Docking scenario

go from room **R2** to room **R1** and then execute a trajectory to the approach zone of workstation **WS1** and finally dock at port **P1**. The object of interest in this scenario is that the robot is to go by workstation **WS2** and afterwards encounter an unknown object in its path.

4.1. Plan elaboration

The user specifies the goal to be achieved: (AT HILARE WS1 P1). FPS first checks that the symbols are known to it and establishes that the workstation **WS1** is in room **R1**. FPS builds an initial plan:

```
(SEQ (GOTOROOM R1)
     (GODOCK WS1 P1))
```

It then checks with the 'Route Planner' (RP)[32] and 'Navigation Planner' (NP)[33,34] to find out if the plan is feasible.

FPS first asks RP for the best* route to **R1**. The interactions between FPS, RCS, and RP may get involved: (1) during execution a door can be found closed so that another route will have to be found, or (2) if at some point another door is found open that RP believed closed and it might better fit the criterion for 'best' route. For the moment, RP provides *one* route if it exists. In the future, it may be more advisable to provide a selection among multiple routes; both FPS and RCS would then be able to plan and execute more flexible routes.

At this point the plan has been refined to:

```
(SEQ (GOTODOOR D1)
     (GOTHRUDOOR D1)
     (GOTOAREA Z1)
     (DOCK WS1 P1)
```

The route has been incorporated. The workstation **WS1** has been replaced by its approach zone followed by an explicit docking manoeuver.

Once FPS has the last route segment, it asks NP if a path to workstation **WS1** port **P1** exists. The NP confirms that a path exists *and* that it passes to the right of **WS2**, another workstation. Thus, FPS considers the plan valid. At this point, FPS may generate a secondary plan elaboration phase in order to provide RCS with a simple execution model together with the plan suitably translated. This phase is necessary when a plan's context determines a configuration of monitors which would supercede RCS's default activation of monitors. An example would be the 'Find Red Block' problem (§ 3.2.2) during which the robot must disable Local Obstacle Avoidance (LOA) when certain portions of the path are executed. It then transforms the plan into the following execution plan:

```
GOTO      D1 LOA SPEED=30          goto door
GOTHRU                             traverse door
MNTR      SEGMENT S1 FIXED-OBJ WS2→
          VERIFY-POS WS2
GOZONE    Z1                       goto area
DOCK      WS1 P1
```

The path during which **WS2** is sonar detectable is assigned the symbol **S1**. The 'Verify-Pos' monitor caches the geometrical features of **WS2** that should be visible in zone **S1**.

4.2. Execution control

RCS receives the execution plan from FPS and establishes the processing modules (PMs) and monitors that will execute the plan. Some monitors are plan dependent and some are set by RCS itself (e.g., battery-level or position reckoning monitors). The first step,

```
GOTO D1 LOA SPEED=30*
```

specifies that the robot is to go to the approach zone of door **D1** at 30 cm/sec with Local Obstacle Avoidance (LOA) enabled. Whenever the robot is to go to an approach zone, by default a monitor is established to survey entry into the approach zone (*viz.* planning serendipity) – in this case for the zone in front of door **D1**.

Control is given to the 'Motion Control' PM. Since there are no incidents enroute the robot arrives at **D1**. At this point informs FPS that the last step has succeeded. Then control is passed on to the 'Traverse Door' operator

* The best route may involve a variety of criteria, such as rapid route, short route, reasonably-free route, or safe route. The route chosen does not have to be extreme with respect to any criterion. At the moment, FPS asks only for the shortest route.

* Refer to § 3.2.1 for syntax.

which gets the robot through the door using its sonar sensors. By default, LOA is inhibited. During door traversal the robot's reckoning of its position is refined. Again, after the door traversal has succeeded RCS informs FPS which will update its context due to the room change.

The next step establishes an explicit monitor to detect workstation **WS2** and then verify and update the robot's position reckoning:

MNTR SEGMENT S1 FIXED-OBJ WS2 $\Rightarrow$

VERIFY-POS WS2*

Then the command to go to zone **Z1** (GOZONE) is passed again to the Path Navigation PM. By default, a monitor is activated to survey for the approach zone **Z1**. The Motion Control PM calculates the precise geometric trajectory given the details of the path selected by the Navigation Planner.

The first interesting event occurs when the robot enters the segment of the path known as **S1**. During this segment the sonar devices on the left-hand side of the robot register an object. A local geometric model is built and compared with the cached geometric model of the workstation **WS2**. In the case that they match, the robot can update the estimation of its actual position reckoning.

The trajectory continues uninterrupted until the robot encounters an obstacle on its path. At that point the LOA procedure takes over and depending upon the pattern of sonar measurements decides to turn left to go around the obstacle. RCS is informed of this decision and passes the information on to FPS. At this moment all the following are in progress:

1. LOA is executing with an unknown obstacle at 40 cm on its right.
2. A local geometric model of the object/obstacle is under construction for later comparison with known objects, etc.
3. The Motion Control PM is monitoring the actual trajectory in order to establish the moment when it should take over from the LOA procedure**.
4. FPS reads the new information that filters up through RCS in order to see if the plan and/or its execution should be continued, modified, or aborted and replanned.

In the case that **LOA** takes the robot into the approach zone **Z1**, the plan step is considered satisfied and all current execution is terminated. Otherwise, the normal plan execution and the Motion Control **PM** gain control and get the robot into zone **Z1**.

* A similar monitor exists for increasing the precision with which an object's position is known:

MNTR SEGMENT S OBJECT ?O $\Rightarrow$ UPDATE-POS ?O

It refines the object's position if the robot's own reckoning is sufficiently precise.

** Currently the Motion Control PM is only looking for the point when the actual and desired trajectories can intersect. The handling of the unknown object could be done differently if we involve more DMs or if we elaborate the Path Navigation PMs trajectory prediction. If the Navigation Planner were modified it could help the Motion Control PM so as to optimize the rejoining of the normal path or better yet to shortcut the entry into zone **Z1**.

Finally the Docking operator performs the necessary local manoeuvers in order to dock the robot at the desired port **P1** of workstation **WS1**. This example protocol shows the interactions between the decision modules and processing modules. There is a strong interaction between the modules and a high degree of parallelism is allowed*.

5. SUMMARY

An integrated planning and execution control system has been presented. It is under test on the real-world robot HILARE. FPS combines domain-independent plan structuring critics with domain specific constraints and critics. They watch over a general and flexible plan structure. Since the system is rule-based, heuristics may be added at any level; for the moment few exist. Their usefulness should become apparent when FPS performs error recovery and replanning.

RCS presents execution control as a distributed set of cooperating processing modules (PMs) and monitors. The monitors allow for controlling actions, action chaining, unexpected asynchronous events, and dynamic behaviour modification. RCS decomposes tasks given to it by FPS into execution commands for the PMs. It allows execution under time constraints. Furthermore, RCS is capable of controlling limited local plan mending in case of execution error. It provides an initial solution to the problem of serendipity, though the problem has been shown to be very complex in the general case. Finally, RCS integrates the different decision and processing modules and coordinates their communication.

ACKNOWLEDGEMENTS

The authors would like to thank everyone involved with the HILARE project for providing the necessary support for a real-world testbed for FPS and RCS. They would also like to thank the referees for their helpful comments.

REFERENCES

1 Fikes, R., Hart, O. and Nilsson, N. Some New Directions in Robot Problem Solving in *Machine Intelligence*, (Eds B. Meltzer and D. Michie), Edinburgh University Press, Edinburgh, 1972, 7

2 Brady, M. Artificial Intelligence and Robotics, *Artificial Intelligence*, 1985, **26**, 79–121

3 Sacerdoti, E. Plan Generation and Execution for Robotics, Technical Note 209, SRI International, Menlo Park, Calif., April 1980

4 Giralt, G., Sobek, R. and Chatila, R. A Multi-Level Planning and Navigation System for a Mobile Robot; a First Approach to HILARE, *6th IJCAI*, Tokyo, Japan, August 1979

5 Giralt, G. *et al.* An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots, in M. Brady and R. Paul (Eds.) *Robotics Research: The First International Symposium* (MIT Press, Mass.), 1984, 191–214

6 Waldinger, R. Achieving Several Goals Simultaneously, in *Readings in Artificial Intelligence*, (Eds N. Nilsson and B. Webber), Tioga Publishing, Palo Alto, CA, 1981, 250–271

7 Gupta, A. and Forgy, C. Measurements on Production Systems, Technical Report CMU-CS-83-167, Carnegie-Mellon University, December 1983

8 Sobek, R. Achieving Generality and Efficiency in a Production System Architecture. LAAS-CNRS Memo 85184, Toulouse, France, 1983

9 Sobek, R. AND/OR Match Nets; An Efficient Production System Representation. LAAS-CNRS Memo 87120, April 1987

* Due to our robot's physically distributed architecture, a number of DMs and PMs are physically distributed among separate on-board processors[4].

10 Forgy, C. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, 1982, **19**, 17–37

11 Davis, R. and King, J. An Overview of Production Systems, in *Machine Intelligence*, (Eds E. W. Elcock and D. Michie), Wiley, New York, 1976, **8**, 300–332

12 Minsky, M. Framework for Representing Knowledge, in *The Psychology of Computer Vision*, (Ed. P. Winston), McGraw-Hill, 1975

13 Sobek, R. A Robot Planning System Using Production Rules, *9th IJCAI*, Los Angeles, Calif., Aug. 1985

14 Fikes, R. and Nilsson, N. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, 1971, **2**, 189–208

15 Sacerdoti, E. *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977

16 Sobek, R. Automatic Generation and Execution of Complex Robot Plans, Master's Project Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, September 1975

17 Wilkins, D. Representation in a Domain-Independent Planner, *IJACAI-83*, Karlsruhe, West Germany, 1983, 733–740

18 Stefik, M. Planning with Constraints (MOLGEN: Part 1), *Artificial Intelligence*, 1981, **16**, 111–139

19 Stefik, M. Planning and Meta-Planning (MOLGEN: Part 2). *Artificial Intelligence* 1981, **16**, 141–169

20 Chapman, D. Planning for Conjunctive Goals, *Artificial Intelligence*, 1987, **32**, 333–378

21 Fikes, R. *et al.* Learning and Executing Generalized Robot Plans, in N. Nilsson and B. Webber (Eds.) *Readings in Artificial Intelligence* (Tioga Publishing, Palo Alto, CA, 1981), 231–249

22 Fikes, R. Monitored Execution of Robot Plans Produced by STRIPS, *Information Processing 71*, (North-Holland, 1972)

23 Shafer, S. A. *et al.* An Architecture for Sensor Fusion in a Mobile Robot, *Proceedings IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986

24 Brooks, R. A Layered Intelligent Control System for a Mobile Robot, in *Robotics Research: The Third International Symposium*, (Eds O. Faugeras and G. Giralt), Gouvieux, France, (MIT Press, Mass., 1986), 365–372

25 Srinivas, S. Error Recovery in Robot Systems, PhD Thesis, Computer Science Dept., California Institute of Technology, 1977

26 Chochon, H. and Alami, R. NNS, A Knowledge-based On-Line System for an Assembly Cell, *Proceedings IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986

27 Chatila, R. and Giralt, G. Task and Path Planning for Mobile Robots. In A. K. C. Wong and A. Pugh (Eds.) *Machine Intelligence and Knowledge Engineering for Robotic Applications*. NATO ASI Series, F33, Springer-Verlag, 1987, 299–330

28 Ghallab, M. Task Execution Monitoring by Compiled Production Rules in an Advanced Multi-Sensory Robot in *Robotics Research: The Second International Symposium*, (Eds H. Hanafusa and H. Inoue), MIT Press, Mass., 1985, 191–214

29 Chatila, R. and Laumond, J.-P. Position Referencing and Consistent World Modelling for Mobile Robots, *Proceedings IEEE International Conference on Robotics and Automation*, St. Louis, Mo, March 1985

30 Khoumsi, A. and Migaud, P. Amélioration des Capacités Comportementales d'HILARE – Pilotage et Controle d'Exécution de Mouvements, LAAS-CNRS Memo 86062, Toulouse, France, 1986

31 Boissier, L. Modélisation de l'Environnement et Localisation du Robot Mobile HILARE par Télémétrie Laser, Thèse 3ème cycle, LAAS – Université Paul-Sabatier, Toulouse, France, December 1985

32 Chatila, R. Path Planning and Environment Learning in a Mobile Robot System, *European Conference on Artificial Intelligence*, Orsay, France, July 1982

33 Laumond, J.-P. Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints, *Proceedings of the International Conference on Autonomous Systems*, Amsterdam, Netherlands, December 1986

34 Chatia, R. Mobile Robot Navigation: Space Modelling and Decisional Processes, in O. Faugeras and G. Giralt (Eds.) *Robotic Research: The Third International Symposium*, Gouvieux, France, (MIT Press, Mass., 1986), 373–378