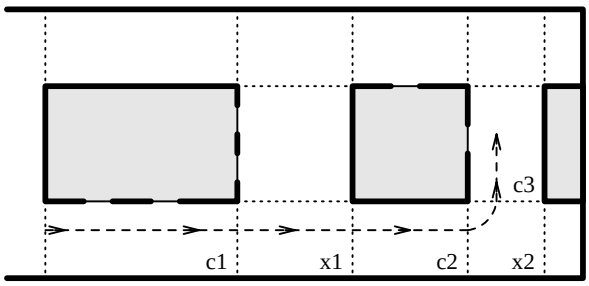

Figure 4: Translating move operations in paths

sponding sequence of move operations; this task can be performed with different modalities, as the same path can in general be described by several move operation sequences. We will investigate the extension of the interface in order to allow direct translation of move operation sequences into a format which is directable executable by the navigation system.

8 Conclusions

The aim of this work is to build an intelligent system for navigation, featuring both planning capabilities and reactivity to external events and navigation failures. We have presented a flexible two-layered framework where these capabilities can be combined and used by programming strategies for the reasoner. This provides abilities that are not available in most of the classical planners.

By keeping the planning of paths (and move operators) independent of the particular low level system, we have defined a paradigm for controlling navigation which is actually independent of the low level features of the navigation module.

The current implementation is at an experimental level. We plan to test performances of different strategies, to enhance the replanning activities, the reactions to failure, the reasoning capabilities and to tighten the communications between the navigation and the planning level to build a robust navigation system.

References

[Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[CC92] R. Cattoni and B. Caprile. Navigation Shell: Planning and Supervising Reflexive Indoor Navigation. Forthcoming IRST Technical Report, 1992.

[DG91a] S. Dalbosco and F. Giunchiglia. Il pianificatore di navigazione. Technical Report 9111-16, IRST, Trento, Italy, 1991.

[DG91b] S. Dalbosco and F. Giunchiglia. La mappa dell'IRST - Versione 1. Technical Report 9111-14, IRST, Trento, Italy, 1991. Internal Report.

[GL87] M. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of the 6th National Conference on Artificial Intelligence*, pages 677–682, 1987.

[GLS86] M. Georgeff, A. Lansky, and M. Schoppers. Reasoning and planning in dynamic domains: An experiment with a mobile robot, 1986. SRI Technical Note 380.

[GTCS91] F. Giunchiglia, P. Traverso, A. Cimatti, and L. Spalazzi. Programming planners with flexible architectures. Technical Report 9112-19, IRST, Trento, Italy, 1991.

[Kae87] L.P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about actions and plans: Procedings of the 1986 Workshop.* Morgan-Kaufmann Publishers, 1987.

[SCT92] L. Spalazzi, A. Cimatti, and P. Traverso. Implementing planning as tactical reasoning. In *Proc. AI Simulation and Planning in High Autonomy Systems Conference*, Perth Australia, 1992. IEEE Press.

[Sim91] R. Simmons. Coordinating planning, perception and action for mobile robots. In *AAAI Spring Symposium on Integrated Intelligent Architectures*, pages 156–159, 1991. SIGART Bulletin vol. 2 no. 4.

[Str90] L. Stringa. An integrated approach to artificial intelligence. Technical Report 9012-11, IRST, 1990.

[TCS92] P. Traverso, A. Cimatti, and L. Spalazzi. Beyond the single planning paradigm: introspective planning. In *Proc. 10th European Conference on Artificial Intelligence*, Vienna, Austria, 1992.

[TD92] P. Traverso and S. Dalbosco. Reasoning with navigation paths and move operations. Forthcoming IRST Technical Report, 1992.

[Wil88] D. E. Wilkins. *Practical Planning: extending the classical AI planning paradigm.* Morgan Kaufmann, San Mateo, 1988.

The "high level" reactivity discussed in section 4 is a uniform tool for a flexbile treatment of execution failures. In order to demonstrate this point let us consider the following global strategy for navigation control: first, the position of the robot is acquired by issuing a `query-position` to the interface; then, the planner `shortest-path` is called, and the resulting path is given as input to the navigation command `start-navigation`. When navigation terminates, the robot position is queried again, and if it matches with the target position the strategies terminates. The interesting cases are when the robot position is different from the required one. If the robot is not on the specified trajectory, it might have failed in recognizing a landmark, and lost its way: in this case a new path is determined to the target position from the current position, and then execution is activated. If the robot is on the specified path, the interface is queried for the reason of the failure with `query-failure-description`. According to the result, different behaviours are possible. If the reason is an obstacle, it is asked to determine if it is a mobile or a fixed one. In case it is fixed, i.e. `perceive-fixed-obstacle` is true, the map is updated by representing a fixed obstacle in the corresponding position by `arc-add-failure-description` and `set-arc-inaccessible`, and a new path is planned. If `perceive-crowding` is true, several reactions are possible: wait a certain time and hope that people move, or try to signal the presence of the mobile robot with `signal-robot-presence`. The same behaviour can be followed when the reason for failure of navigation is a bumper collision or a stop from the emergency button. The strategy also contains global criteria to recover from multiple failures: for instance, when moving obstacles are repeatedly slowing down and interrupting the robot navigating along a given path, the strategy decides to plan for an alternative path that goes through the least overcrowded areas by calling a different planner.

By specifying such a strategy the reasoner can be programmed to integrate the usual replanning capabilities (i.e. replanning for an alternative path not passing through the node where failure occurred or replanning with a different modality) with the effective use of the information which is accessible after the execution.

6 An example of navigation

In this section we document a session of the navigation system developed at IRST. Figure 3 depicts a simplified version of the map of the building, and the path followed by the robot during the navigation.

The target was given to reach the laser printer UX. The reasoner queried the position of the robot, obtaining the reception, and activated the `shortest-path` planner to plan from reception to the printer. The path ran along the west corridor, where a fixed obstacle had been positioned immediately after the intersection with the central corridor. Navigation was activated along the specified path, and a failure occurred at the intersection because the robot could not avoid the obstacle. The reason for failure was recognized to be a fixed obstacle: the reasoner "cut" the arc of the map and replanned for a new path from the current position to the target. The new path went along center, east, north and west corridors to the printer. The navigation along the new path was successful.

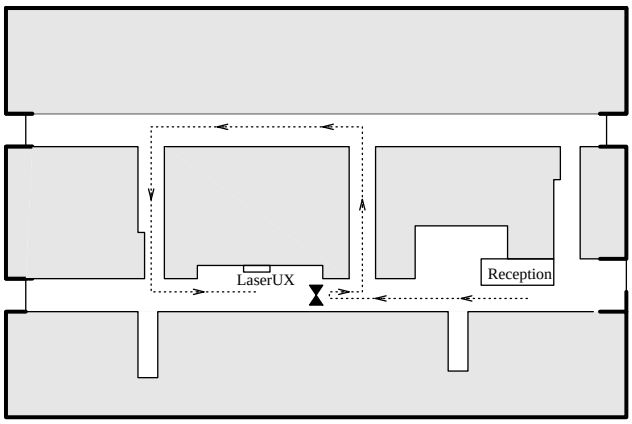

Figure 3: An example of navigation

7 Extending the planning level

In this section we present a future extension to the planning level. We are investigating a second way to reason about and to plan navigation: the idea is to describe navigation by means of **move operations** [TD92], that represent primitive and conceptual navigation operations. They are still independent of the features of the navigation level, and are based on the knowledge contained in the map. They are actions like "follow the corridor to the next cross", "turn to the right", "move forward to the second office on the right".

Sequences of move operations can be used to describe paths. For instance, let us consider the scenario depicted in figure 4, where c_1, c_2 and c_3 are nodes of type "corridor", and x_1 and x_2 are nodes of type "cross". The following sequence of move operations:

```
< follow-to   2nd corridor left,
    turn-to   left,
  follow-to   1st door >
```

would be an alternative (an more expressive) representation for the path $\langle c_1, x_1, c_2, x_2, x_3 \rangle$.

A module has been implemented which translates move operation sequences into the corresponding sequence of nodes, by accessing the map and simulating the move operations. This make it is possible to execute move operation specifications by the navigation level through the interface. Another module performs the opposite task of translating node paths in a corre-

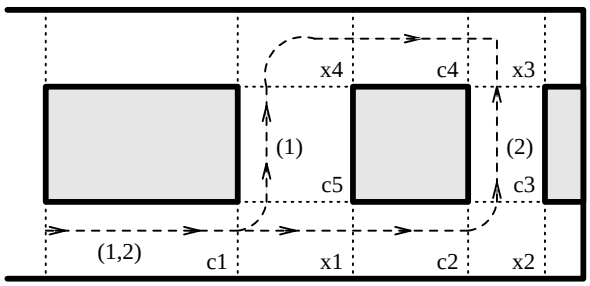

Figure 2: Controlling navigation by acquiring information from the external environment

outside the goals of this papers; in depth descriptions can be found in [GTCS91, TCS92, SCT92]. A strategy is an arbitrarily complicated combination of operations at the planning level. It may, for instance, request information to the navigation level via the reactive-planning interface and, depending on the result, it can decide (or even change during navigation) the strategy to reach the target location.

Particularly significant are strategies based not only on a prior model of the world but also on the acquisition of information from the external environment. By executing such strategies the system may feature a **reactive** behaviour, i.e. it becomes able to react to the changes of the external environment. This notion is a higher level of reactivity than in [Bro86], and much in agreement with [GL87]; it is justified by the capability of the system to acquire from the external environment, during navigation, information otherwise unpredictable, and to deal with sudden changes in the environment. Note that reactivity may coexist at different levels in a system of this kind: in the MAIA project the "high level" reactive system presented in this paper is integrated with the "lower level" reactive robot controller of the navigation level.

In the rest of this section we present some examples of strategies. Let us consider the navigation domain depicted in figure 2; let the initial location be the node c_1 and the target position the node x_3. The typical strategy would be to call a planner (i.e. `shortest-path`) to plan the path from starting to target position, activate navigation with `start-navigation` and verify the result by means of the primitive of the interface `query-state`.

The strategical level allows for many other interesting navigation modalities, which turn out to be useful in different situations. For instance, rather than planning the path to the destination in one shot and then executing it, a strategy can plan for all the paths by calling `all-paths`. Then, by calling the path transformation `common-subpath` it extracts a common subpath (like $\langle c_1, x_1 \rangle$, denoted by $(1,2)$ in figure 2), and activates navigation to x_1. Once the mobile robot has reached the cross x_1, it could choose the "best path" by exploiting information acquired from the external environment: for instance, ultrasound- or camera-detected information might reveal the presence of permanent (`perceive-fixed-obstacle`) or moving (`perceive-crowding`) obstacles, and allow for a more grounded decision. This method of navigation, in spite of requiring more interaction with the navigation level, is much more likely to succeed than the previous one, especially in an unpredictable domain.

Another interesting navigation modality is monitoring: the navigation level, for some reason, may be unable to guarantee that the robot will follow the specified path or will recognize that it has lost the track. In this case, rather than calling the navigation level only once by providing the whole path, a safer strategy would be to "step" the path execution, by splitting it in smaller paths (e.g. by calling `rectify-path`) and querying the position of the robot at the end of every navigation, enforcing in this way a control on the route to be followed.

The capability to program strategies makes this two-layered control system independent[1] of the navigation level: a "dumb" navigation system can be controlled by means of a strategy as well as a very "smart" self-monitoring navigation system. The system is therefore an *open* architecture, so that if the capabilities of the navigation level increase (e.g. the system can recognize new objects), you do not need to modify the whole system but only the strategy.

Note that the definition of the strategies depends on the application being developed; according to the features of the navigation domain (e.g. more or less crowded, more or less predictable), it may be more convenient to choose one strategy rather than another. Thanks to the strategical level, which can be programmed to combine planning level activities in arbitrarily complicated ways, the system is *flexible* and applicable to different domains.

5 Reaction to failure

In previous section we have seen how strategies can be used to implement different navigation modalities, by exploiting information from the external environment in order to decide the activation of different phases of navigation. In this section we focus on a situation which forces acquisition and interpretation of information from the external environment: **execution failure**. Failure occurs during execution when the simplifying hypotheses under which the plan was produced are not satisfied in the real world. In these cases, most classical planning systems modify the world representation and then replan. However this is a rather fixed mechanism, and in certain cases it is not the best of all possible strategies; different kinds of failure may require different system behaviours, ranging from general replanning to the execution of an ad hoc procedure.

[1] Of course the data tranlsation subroutines of the navigation-planning interface depend on the actual navigation level.

- `least-crowded-path` returns the best path according to the crowding cost;
- `easiest-path` returns the best path according to the navigation cost;
- `least-cost-path` returns the best path according to the cost function provided as input, which can be a combination of all the costs contained in the map;

More complex planners search for all possible paths between two locations, and determine paths satisfying constraints of different forms. Some of them are:

- `all-paths`, which returns a list of the possible paths from starting to target position;
- `ord-constrained-paths`, returns the list of all the paths from starting to target position, constrained to pass through a list of specified locations, according to a supplied ordering relation;
- `unord-constrained-paths` returns the list of all the paths from starting to target position, constrained to pass through a list of specified locations, in the most convenient order.
- `shortest-ord-constrained-path` returns the geometrically shortest path between the ones satisfying the set of ordered constraints specified.

The planning library performs other operations related to transformation of paths:

- `common-subpath`, which returns the initial subpath common to the paths given in input;
- `rectify-path`, which splits a path into a sequence of paths which do not contain turns
- `path-to-end` which, given a path and a node, returns the path from the node to the end.

3.3 The navigation-planning interface

The navigation-planning interface is the module which makes possible the comunication between the navigation level and the planning level. It hides the details of the navigation level to the planning level, by providing the primitives to activate and inspect it. In a sense it implements an abstract data type for the navigation level modules.

The interface provides for queries and commands. Navigation commands activate of navigation of the robot along a given path. Some examples are:

- `start-navigation`, which activates the navigation along the path specified as input;
- `suspend-navigation`, which stops the navigation in the current state;
- `restart-navigation`, which restarts a suspended navigation;
- `kill-navigation`, which terminates the navigation;

Other commands may activate different functionalities, provided that the are actually available at the navigation level the system is connected to:

- `signal-robot-presence`, which activates one or more devices (e.g. a beeper, a hazard light) to signal the presence of the robot;
- `perceive-fixed-obstacle`, which activates the navigation level to recognize the presence of fixed obstacles;
- `perceive-crowding`, which activates the navigation level to recognize the presence of crowding.

Queries allow the planning level to inspect the status of the navigation level. Possible queries are:

- `query-position`, which returns the node in the map representing the current position of the mobile robot;
- `query-state`, returning the current state of navigation (e.g. navigating, suspended, success, failure);
- `query-failure-description`, returning a description of the failure;
- `query-navigation-history`, which returns a path describing the navigation so far.

The interface implements a comunication protocol between the two levels. Data must be converted from the format of one level to the other. The planning level reasons in terms of symbolic informations like nodes of the map, while the navigation level performs numerical computations. In particular, the path is a symbolic and rather high level information, far from being an executable specification for the navigation system; the interface translates a path into a sequence of numerical specifications of areas the robot has to pass through.

4 The reasoner: deciding the strategies

The task performed by the strategical level is to activate, control and integrate the modules of the navigation level: it decides which kind of planner should be used, activates the navigation by issuing navigation commands to the interface, queries the interface in order to obtain information about the robot state and the result of navigation, and controls the updating of the map and the interpretation of information stored.

The strategical level is composed of a **reasoner** which can be easily programmed to perform desired strategies. Describing the language for programming strategies is

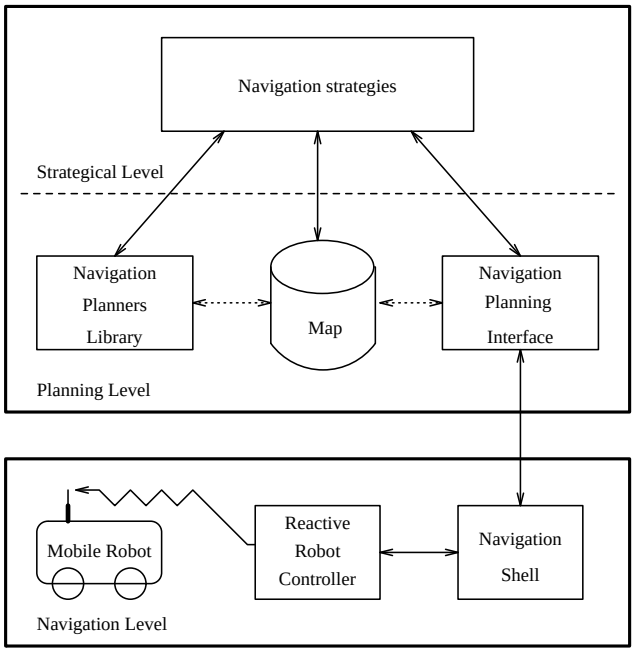

Figure 1: System architecture

group. It is composed of a sensor-based mobile robot, and a reactive controller and a navigation shell [CC92]. Its description is outside the goals of this paper, as the system we present is (largely) independent of the features of the navigation level with which it interacts.

The **planning** level provides the basic building blocks to plan ahead and to interact with the external environment: there are different planners, a map containing a model of the world, and an interface to the navigation level, for executing different navigation commands and acquiring information about the current state of the external environment.

The **strategical** level controls and activates the building blocks of the planning level: for instance, it chooses when to plan for a path, which planner to activate, whether to acquire information from the external or to start navigation, and the navigation modalities. In this way the basic activities of the system may be variously combined to feature different navigation modalities, which integrate planning and reaction to changes in the external environment in the most suitable way.

3 The planning level

The basic functionalities of the planning level are description of the external world, plan formation and plan execution and monitoring. The planning level is composed of three modules (the map, the library of planners and the navigation-planning interface) tightly integrated to perform these basic tasks. For instance, by accessing the map, the planners determine a navigation plan that the interface can use to activate navigation.

3.1 The map

The map [DG91b] is a data base containing information describing various aspects of the navigation domain. The map is structured as a graph, whose nodes represent areas of the building and whose arcs represent the connections between them. Nodes can contain object descriptions, like desks and printers, and landmarks. Each node, object and arc contains geometrical information: the coordinates of its baricenter, its dimensions and so on. Nodes are defined and classified according to their features as "corridor", "open space", "room" and "cross"; similarly, arcs are classified as "open" and "door". Each arc has associated a set of different costs. Some examples are "geometrical cost" (function of the distance between the nodes), "navigation cost" (curves are more "expensive" than aligned nodes), and "crowding cost" (crowded areas are more difficult to navigate). This kind of information can be accessed by means of the suitable primitives (i.e. `arc-get-type`, `node-get-objects`, `arc-get-crowding`).

The map is also used to store other information, which is acquired during navigation: i.e., descriptions of a navigation sessions, success or failure, obstacle descriptions, and so on. This information is associated with arcs by the suitable primitives (e.g. `arc-add-failure-description`). Other information stating if an arc is believed to be accessible or not can be set according to the current state of navigation (`set-arc-inaccessible`) and accessed during the planning process (`node-get-accessible-nodes`). Based on these primitives, more complex manipulations can be performed. One example is belief revision, where the "accessibility" state of all the arcs of the map is recomputed according to the navigation descriptions stored in the map (`revise-arcs-accessibility`). For instance, a crowding failure is usually interpreded as a temporary obstacle, while a fixed obstacle (i.e. a push-cart) is a more persistent object. Future developments will provide the map with the ability to revise costs: for instance, if many navigation failures due to crowding occur in the same area the the corresponding crowding cost can be updated according, i.e. learnt.

3.2 The library of planners

The library of planners [DG91a] is in charge of reasoning about trajectories. At this level a trajectory is represented as a path, i.e. a sequence of adjacent nodes in the map. The modules of the library perform plan formation and transformation. A planner, given a starting and a target position and possibly other information, returns a path or set of paths. The library contains planners of different complexity. The simplest planners return the best path according to a cost function computed by accessing the map:

- `shortest-path` returns the geometrically shortest path;

Navigation by combining reactivity and planning*

Alessandro Cimatti[1], Paolo Traverso[1], Sandro Dalbosco[1], Alessandro Armando[2]

[1]Mechanized Reasoning Group, IRST, 38050 Povo, Trento, Italy. Phone (39)461-814-440. Fax (39)461-810851.

[2]Mechanized Reasoning Group, DIST, University of Genoa, Via Opera Pia 11A, 16145 Genova, Italy

cx@irst.it leaf@irst.it dalbosco@irst.it armando@dist.dist.unige.it

Abstract

In this paper we present a system for intelligent robot navigation. Its main feature is the ability to integrate goal directed planning with information acquisition from the external world, and with reaction to failure. This ability is obtained by means of a two-layered architecture: the higher (strategical) level controls the activation of the modules of the lower (navigation) level, which can perform, in different ways, basic tasks such as path planning, activation of navigation and processing of information acquired from the external environment. By programming the strategical level with the suitable strategies, the system can control navigation with sophisticated modalities.

1 Introduction and motivations

Intelligent navigation systems can be loosely classified in terms of the planning based and reactive paradigms. Systems based on planning [Wil88] heavily rely on a prior representation of the world to determine the actions to execute. Experience with such systems shows that navigation in unpredictable and highly dynamic domains hardly can be done without run-time acquisition of information from the external environment. On the other hand reactive systems [Bro86, Kae87] base their behavior almost completely on the information acquired from the external environment, although a limited planning ahead phase seems useful in order to build a system showing high level behaviours. Therefore navigation systems are becoming more and more integrated [GLS86, Sim91], able to plan ahead and to acquire information from the external environment; their behaviour is based both on a prior model and on the acquisition and analisys of the status of the world.

In this paper we present an integrated system, able to control the navigation of a mobile robot in a complex and unpredictable environment, by combining planning ahead and reactivity. The system has a two-layered architecture: the lower (planning) level provides the building blocks to perform planning ahead with different modalities, and to activate external modules which execute real word actions and acquire information from the environment; the higher (strategical) level controls the activities of the level below, deciding what tasks should be performed (i.e. planning, navigation, perception), in which order and with what modalities. The system is a flexible environment for the integration of planning and reactivity [GTCS91, TCS92, SCT92]; sophisticated navigation modalities can be easily obtained by programming the strategical level suitably.

The system presented in this paper has been developed within the MAIA project at IRST [Str90]. It has been integrated with a lower level system, composed of a mobile robot provided with special purpose vision and ultrasound sensing systems, a reactive controller and a navigation shell. At the moment we are experimenting with the overall system inside the IRST research center.

The paper is structured as follows. In section 2 we describe the overall system architecture. In section 3 and 4 we present the planning and the strategical level of the system, respectively. In section 5 we show how the system is able to deal with execution failures. In section 6 we describe a navigation session. In section 7 we present some extensions currently under development, and in section 8 we draw some conclusions.

2 System architecture

The system presented in this paper has a two-layered architecture: the two (strategical and planning) levels are depicted in figure 1. This system is designed to interact with the modules of an external (lower) level, the navigation level, controlling physical devices for actuation and sensing. Figure 1 also depicts the navigation level developed within the MAIA project by the vision

*This work has been done as part of the project "MAIA", Advanced Model of Artificial Intelligence, under development at IRST. The research described here is partly supported by the Consiglio Nazionale delle Ricerche, Progetto Finalizzato Robotica 1991 and Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo 1991. Many of the underlying intuitions have been provided by Fausto Giunchiglia. He has also provided useful help and encouragement during the development of the whole project. We thank the members of the Vision Group at IRST. The other members of the Mechanized Reasoning Group at IRST are also thanked.

IRST

Istituto per la Ricerca Scientifica e Tecnologica
I 38100 Trento – Loc. Panté di Povo – tel. 0461–314444
Telex 400874 ITCRST – Telefax 0461–302040

Navigation by Combining Reactivity and Planning

A. Cimatti, P. Traverso, S. Dalbosco, A. Armando

May 1992
Technical Report # 9205-11

Published in *Proceedings Intelligent Vehicles '92*, Detroit, June 29 - July 1, 1992.

Istituto Trentino di Cultura