Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

XI Ciclo – 1999– 3

Design and Development of Cognitive Robots

Luca Iocchi

Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

XI Ciclo - 1999– 3

Luca Iocchi

Design and Development of Cognitive Robots

Thesis Committee

Prof. Daniele Nardi (Advisor)
Prof.ssa Luigia Carlucci Aiello
Prof. Umberto Nanni

Reviewers

Dr. Kurt Konolige
Dr. Yves Lesperance

Author's address:
Luca Iocchi
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: iocchi@dis.uniroma1.it
www: http://www.dis.uniroma1.it/~iocchi/

Contents

Acknowledgements

During the years that I spent preparing this thesis, I have had the privilege of working with many extraordinary people, whom I wish to thank here.

First of all, I am deeply indebted to my thesis advisor Daniele Nardi, who has been my guide for this work. He constantly supported my work and helped me to make it better and better. He not only taught me the way of approaching a research problem, but also provided me with the stimuli needed for achieving such an important result.

I wish to express my deepest gratitude to Giuseppe De Giacomo and Riccardo Rosati for their essential contribution to the "Tino project" and their invaluable suggestions for the preparation of this thesis.

I am in debt with Kurt Konolige for his help since the very beginning of my work on robotics, and specially for providing me with the first actual mobile robot on which I focussed all my work. In particular, I thank him for his hospitality and his support during my visit at AI Center of SRI International, where a part of this research has been carried out.

I also appreciate the work of the thesis committee, Luigia Carlucci Aiello and Umberto Nanni, and the very accurate reports of the external reviewers, Kurt Konolige and Yves Lesperance. Their work allowed me to significantly improve the quality of this thesis.

I thank the Coordinator of the "Dottorato in Ingegneria Informatica" of the University "La Sapienza" in Rome, Giorgio Ausiello, and the former one, Luigia Carlucci Aiello, for the organization of a very good PhD program. I also thank my PhD colleagues and all the people at Dipartimento di Informatica e Sistemistica.

Finally, let me express a special thank to my family, Luciano, Margherita, Marco, and Daniela, for continuously supporting and encouraging my activities.

Chapter 1

Introduction

The notion of *agent* is central to the field of Artificial Intelligence and the design of intelligent agents is one of its foundational research goals. Agent-based systems are developed in many different areas (see [87, 88] for a review) with the aim of exhibiting some aspects of intelligent behavior.

Agents are usually defined as entities able to act in an environment according to their goals and their perceptions. In other words, the agent and its environment can be considered as a dynamic system in which changes, due both to the actions performed by the agent and to external events, occur in time. The evolution of the system is driven towards the achievement of a high level goal.

Depending on the actual realization of the agent and on the features of the environment, there are many possible scenarios. We can consider for example a mobile robot embedded in an office environment, or a software system navigating the World Wide Web.

Among all, the most interesting agents' feature for AI researchers is *autonomy*. An *autonomous agent* must be able to perform *complex tasks* in *real environments*, without human assistance or supervision. Let us clarify the meaning of complex tasks and real environments.

We want to realize agents that perform activities that are in some way useful in the environment in which they act. These activities are usually complex in the sense that several objectives must be considered and combined together. For example, a mail delivering robot must be able not only to bring packages from one office to another, but also to avoid obstacles during its path, and to manage the mail schedule.

The second point is that real environments are

1. *dynamic*, changes can occur at every time and a timely response to these changes is sometimes a critical factor;

2. *unpredictable*, the effect of changes in the environment cannot be always and completely foreseen;

3. *partially known*, it is not possible to have complete information on the environment, since many situations cannot be known a priori.

The previous statements lead us to indentify the main capabilities that are required for an agent to be autonomous.

1. It must promptly react to unforeseen situations.

2. It must acquire new knowledge from the real world, in order to detect the actual effects of events.

3. It must reason on the incomplete knowledge it has on the environment.

Research on autonomous agents has been developed within the field of Artificial Intelligence by first considering mobile robots as the actual agents, while the extension of the developed technologies to software systems is much more recent.

Initially, the focus of this research has been on the high-level representation of actions that the robot can perform. However, it soon became clear that it is very difficult to build robots that exhibit the desired behavior in real environments, simply on the basis of such a declarative representation. Consequently, research split into two streams, that developed rather independently of each other. On the one hand, the basic functionalities of the robot, such as navigation or sensor interpretation, have been addressed; on the other hand, sophisticated logic-based representations of the agent have been developed.

Recent work (see for example [13]) has shown that a mobile robot can effectively be provided with reactive capabilities. However, a mobile robot needs not only the ability to promptly react and adjust its behavior based on the information acquired through its sensors, but also to achieve high-level goals. Therefore, it should also be able to reason about the actions it can perform, find plans that allow it to achieve its goals and check whether the execution of actions leads to the accomplishment of the goals. The integration of reactive and planning capabilities has thus become a focus of research in mobile robotics (see for example [75, 40, 83]).

We believe that this renewed effort to combine a logic-based view of the robot as an intelligent agent with its reactive functionalities is essential to devise agents that operate in a real world environment. To this end a new research field has been developing in the last years. It is named *Cognitive Robotics* [56] and aims to design and realize actual agents (in particular mobile robots) that are able to accomplish complex tasks in real environments without human assistance, and that, to this purpose, can be controlled at a high level by providing them with a description of the world and expressing the tasks to be performed in the form of goals to be achieved.

The peculiar feature of a *cognitive agent* is the presence of cognitive capabilities for reasoning about the information sensed from the environment and about the actions it can perform. These reasoning capabilities are usually provided by the deductive properties of a logic-based formalism for reasoning about actions. Moreover a symbolic language, allowing for a high level representation of the environment and the agent's abilities, provides the agent designer with an extremely powerful and flexible design tool. Many researchers are thus studying how to take advantage of logic-based frameworks for building the *cognitive level* of an actual agent.

One main example of cognitive agents is that of a mobile robot embedded in a real environment. A *cognitive mobile robot* is thus a mobile robot which is provided with a high-level description of the world and is capable of achieving a variety of goals in real and dynamic environments.

The design and realization of cognitive agents has been addressed from different perspectives, that can be classified into two groups.

1. A number of theories of actions have been developed in order to represent the agent's knowledge. They are characterized by the expressive power, that is the ability of representing complex situations, by the deductive services allowed, and by the implementation of automatic reasoning procedures.

2. On the other hand, system architectures for mobile robots development have been investigated. There are many features that are considered important in the design of agents' architectures and each proposal describes a solution that provides for some of these features.

Most of the work on Cognitive Robotics is focused on one of the previous issues (i.e. action theories or architectures). Indeed in some cases the only contact point between them is expressed in terms of the implementation of the high level actions by means of control procedures for a mobile robot.

Our opinion is that the integration of a logical reasoning framework within a robot architecture is a central problem in the design of cognitive agents. We believe that it is not appropriate to separate the design of the reasoning systems and the agent's architecture. These activities should be integrated in a global design process, that we consider a fundamental step for the realization of actual cognitive agents.

1.1 Objectives

Our main objective is to devise an *integrated design methodology* for integrating the cognitive abilities provided by a formal theory of actions within a robotic architecture, with the aim of realizing an actual cognitive mobile robot. This methodology should provide the designer with an effective tool for a systematic use of its design abilities in realizing cognitive agents.

The second objective is to *develop an actual cognitive mobile robot*, or, more precisely, to realize a cognitive level on top of an existing mobile robot. To this end, we identify some fundamental aspects to be addressed:

1. the definition of a robot architecture able to effectively integrate a logical formalism for representing dynamic systems and the reactive capabilities required for acting in real environments;
2. the analysis of the technologies to be used for implementing the architecture modules, in particular the decision of making use of existing approaches or devising a new one;
3. the implementation of the modules and the realization of the interfaces among them;
4. the execution of a number of experiments in order to verify the validity of the methodology and of the techniques used for solving specific problems.

1.2 Summary of results

The thesis is focussed on the design and the development of cognitive agents, providing a methodological approach for their design, and showing an actual implementation in the realm of mobile robots. The contributions presented in this thesis are:

1. a design methodology for cognitive agents that allows for an incremental development of actual agents and for reusing most of the work when moving the agent to a new (similar) environment;
2. a robot layered architecture that integrates reasoning and reactivity in a heterogeneous setting, allowing thus for making use of many different technologies for solving application-specific problems;
3. a new framework for reasoning about actions has been proposed in order to implement a reasoning system able to reason on the incomplete knowledge of the agent and to automatically generate conditional plans for achieving the agent's goals;
4. a stereo vision system has been implemented in order to increase the sensing capabilities of the robot, in particular we have devised new multiresolution and camera calibration techniques for an efficient, reliable, and accurate implementation;
5. the integration of all the above features in the implementation of an actual cognitive mobile robot, that we call "Tino".

The methodology we have defined presents two main aspects: (i) a heterogeneous layered architecture is defined in order to implement an effective integration between reasoning and reactivity, (ii) the high-level description of application domains and agent's tasks is obtained by the execution of a sequence of steps that take into account, above all, the realization of the interface between reasoning and reactivity, by first combining the definition and the implementation of actions to be performed and conditions on the world to be verified.

Observe that the first point is important for defining an effective architecture for the development of a complex agent, where the use of many different techniques for solving specific problems in the application domain is fundamental. The second point is crucial in the actual accomplishment of tasks, since it provides the designer with a tool for incremental implementation of the agent's abilities and it allows for reusing the results of the early steps in the methodology for the realization of applications in new similar environments.

Previous works on the realization of cognitive robots [57, 82] follow a different approach: they focus the attention on defining high-level description of the environment, while the integration within a reactive robot architecture is considered as a separate problem, by implementing high-level actions with some specific control procedures. The important feature of our methodology is that it tackles the problem of integrating reasoning and reactivity in the middle, that is by defining first the interface between the two levels. In this way the designer can take advantage of the possibility of identifying design errors in the early stages of the design process.

The methodology has been successfully tested in developing several applications for an actual cognitive mobile robot. To this end, we redesigned the system architecture of our robot and we fully realized all the related modules and the interfaces among them. In contrast with some other previous proposals of hybrid robot's architectures, that define a single formalism for both the reasoning and the reactive system [44, 83, 75], we obtain the integration of reasoning and reactivity by heterogeneous representations of information about the environment. In this way, since there are no constraints on the knowledge representation language, it is possible to exploit many different technologies that are more adequate for solving specific problems.

In the realization of the cognitive architecture we chose to design and implement two novel components: the reasoning system and the stereo vision system.

A new framework for reasoning about actions, presented in [21, 22], has been devised within a Description Logic formalism. This framework allows for representing the incomplete knowledge of the agent about the environment and reasoning on sensing (or knowledge-producing) actions. Moreover we realize a conditional planner based on this formalism. The formalization of sensing actions and the generation of conditional planners are rarely addressed in Cognitive Robotics [61]. We believe that the ability of acquiring new knowledge during task execution is essential for the development of cognitive agents.

Finally, in real environments, actual execution of complex tasks is deeply related to the ability of understanding situations in which the agent can be, and therefore the actual implementation of effective and robust sensing actions is a fundamental step. We thus decided to augment the sensing capabilities of the robot by implementing a new multiresolution stereo vision system, also presented in [45], that has been integrated in the system architecture. With respect to other implementations of stereo vision systems for mobile robots [16, 85] we obtained better performance (we can process more than 10 frames per second, with respect to 2-5 fps provided by other systems) on low-cost hardware (we do not make use of any customized hardware device like in the other cases). Moreover the stereo vision system has been fully integrated in our cognitive architecture, and in this way it is used not only for improving navigation tasks, but also and above all for the implementation of reliable and robust data interpretation routines for verifying high-level conditions on the world.

We have successfully performed many experiments in our Department and in other locations, and many demonstrations since the 1995 International Workshop on Description Logics. The

effectiveness of the methodology for designing cognitive robots is proved by the small amount of time required for setting up an experiment.

1.3 Outline of the thesis

The thesis is organized as follows.

- In Chapter 2 we examine the state of the art of Cognitive Robotics, by examining previous proposals of architectures for cognitive agents, formalisms for reasoning about actions, and a discussion about recent developments of cognitive robots.
- Chapter 3 introduces the hardware devices and control system of the mobile robot that has been the basis for our work.
- In Chapter 4 we introduce the methodology for the design of a cognitive agent.
- In Chapter 5 we describe the methodology at work on our mobile robot "Tino". In particular we describe a new system architecture that allows for the integration of reasoning and reactive capabilities.
- In Chapter 6 we introduce a new formalism for knowledge representation and reasoning about actions, and the corresponding implementation of procedures for conditional plan generation.
- In Chapter 7 a new stereo vision system, that has been developed in order to increase the ability of acquiring new knowledge from the environment, is described.
- Chapter 8 shows the design and the implementation of several applications for our cognitive robot "Tino".
- Conclusions and perspectives on future work are drawn in Chapter 9.

Chapter 2

Cognitive Robotics

In recent years a new research field has been introduced within the AI community. It is called *Cognitive Robotics*[1] and the significant effort in this field is proved by the birth of a number of Cognitive Robotics Groups and the organization of Cognitive Robotics events (e.g. *AAAI-98 Fall Symposium on Cognitive Robotics*).

Cognitive Robotics aims at designing agents (robots) with reasoning capabilities able to act in real, and hence dynamic, unpredictable and incompletely known environments. Every agent can be seen as a system with sensing devices for inputing data from the world and acting devices for performing changes in the world. Depending on the application, the agent can be developed as an entire software system in which sensing and acting devices correspond to input and output routines, as well as a software system for controlling hardware devices. For example, in a mobile robot, sensing devices (such as sonar sensors, video cameras, etc.) and actuators (such as driving wheels) are typically controlled by software procedures.

The designer of one of these agents is required to give a description of the environment and of the agent's goals. The agent embedded in this environment must be able to act according to its goals, its knowledge about the world, and its perception of the world, without human assistance.

The problem of designing "intelligent" agents has been faced in the AI community from two different perspectives. From one hand, a number of systems architectures have been presented, in order to support actual implementations of mobile robots able to perform complex tasks showing an "intelligent" behavior. From the other hand, formalisms for representing the agent's knowledge and for reasoning about actions have been studied in order to provide the agent with cognitive capabilities.

In this chapter we analyse the recent work on Cognitive Robotics, by first examining agents' architectures and reasoning frameworks, and then discussing very recent results in developing cognitive robots.

2.1 Agents' Architectures

The first step in the design of autonomous agents is to discuss effective system architectures. Agents usually present different kinds of sensing and acting devices. The flow of data from the sensors to the actuators is processed by several different modules and the description of the interaction among these modules defines the *agent's architecture*.

In Fig. 2.1 a very general architecture is shown:

[1] The term "Cognitive Robotics" was first introduced by the research group at the University of Toronto led by Ray Reiter.

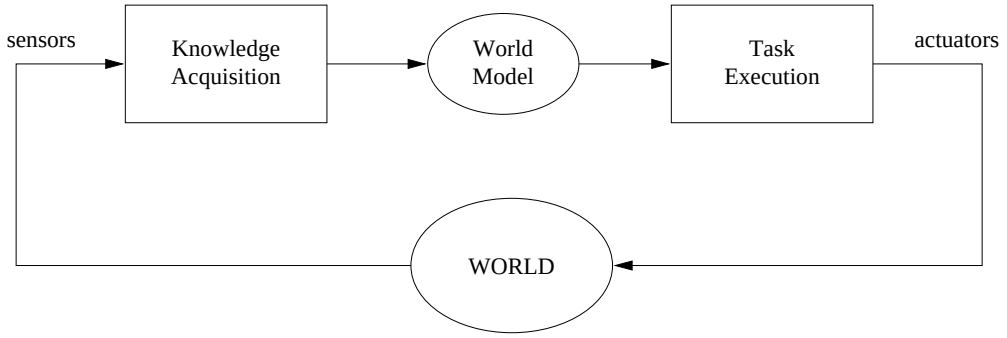


Figure 2.1: General agent's architecture

1. The Knowledge Acquisition (KA) module builds and maintains a model of the world by processing sensor data.
2. The Task Execution (TE) module provides a decision making procedure for choosing the actions to be performed and effective action execution for performing changes in the world.

The schema shows an important feature of the system, that is the *feedback* mechanism given by sensor data processing and world model updating. The feedback allows for actually monitoring the changes in the world, providing the agent with a way to determine the results of its actions and to detect external events.

The effectiveness of this architecture depends on the representation techniques used for modeling the world. In fact, building and maintaining a high-level symbolic representation of the environment in this schema would lead to poor reactive capabilities in the agent, since all data must be processed by (usually time-consuming) interpretation and decision-making procedures. On the other hand, low-level representation are not adequate for describing complex situation and for high-level reasoning.

In the next sections we analyze different proposals of architecture for autonomous agents that have been presented in the last years.

2.1.1 Deliberative Architectures

The research on mobile robots has initially been centered on the view of the robot as an agent embedding a high-level representation of the environment and of the actions that it can perform. Indeed the first attempts in building cognitive agents were mainly concerned with automatic plan generation, that is the generation of a detailed plan (a complete list of actions) to achieve a given goal, often relying on complete and correct knowledge about the world. The execution of such a plan is usually considered as a separate problem that can be addressed later on.

The first remarkable attempt to build a cognitive agent is represented by the robot Shakey [67], which relies on an internal model of the world described in the STRIPS language, and on a planner that generates the actions to be performed in order to achieve the robot's goals. Plans are then executed by an appropriate execution module.

This approach leads to the decomposition of the problem into a sequence of functional modules (shown in Fig. 2.2 a)), where perceptual data are interpreted for creating a model of the world, a planner generates the actions to be performed, and the execution module takes care of executing these plans. In practice a sense-plan-act cycle is repeatedly executed.

The crucial problem is that building a high-level world model and generating a plan are time consuming activities and thus these systems have shown very soon to be not suitable for agents embedded in dynamic worlds.

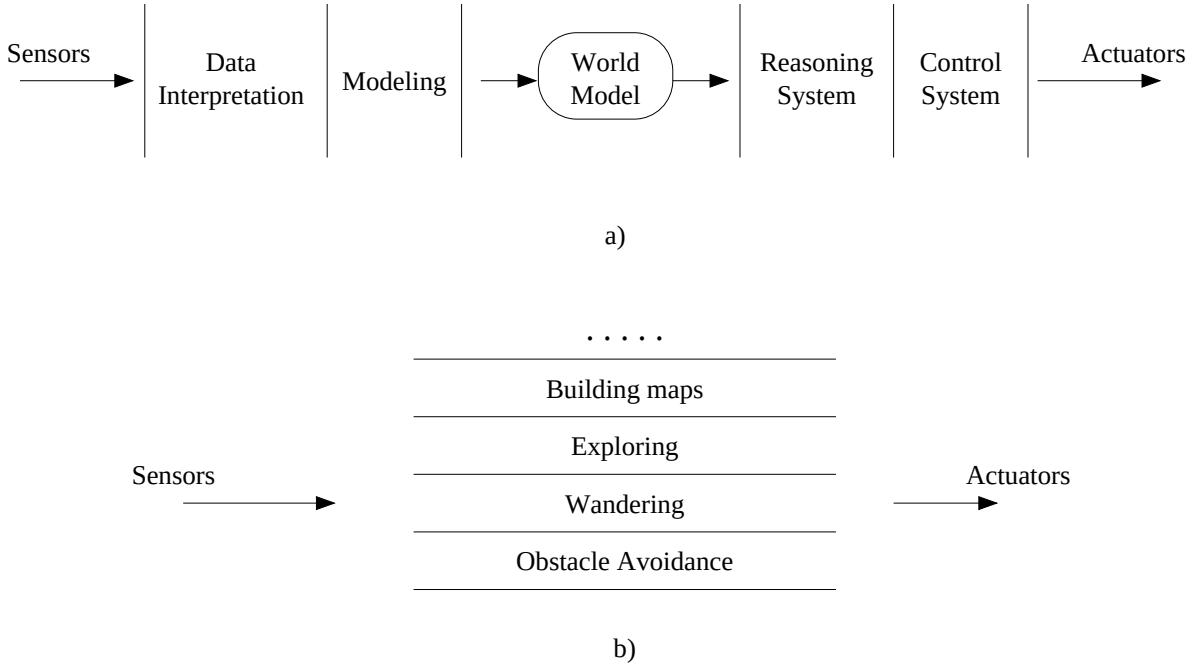


Figure 2.2: a) Deliberative architetcture - b) Reactive architecture

2.1.2 Reactive Architectures

A different line of research, in contrast with the reasoning-based approach, focuses on the basic functionalities of the robot, such as navigation or sensor interpretation, and proposes purely reactive systems.

A different architecture, in which Knowledge Acquisition and Task Execution are not clearly separated, has been first proposed by Brooks in [13] (see Fig. 2.2 b)). It is called the *subsumption architecture* and is composed by levels of competence containing a class of *task-oriented behaviors*. Each level is in charge of accomplishing a specific task (such as obstacle avoidance, wandering, etc.) and the perceptual data are interpreted only for that specific task. Afterwards many reactive systems for controlling mobile robots were proposed following this idea [1, 38, 46, 80].

The peculiar feature of these systems is that they do not build a model of the world. In fact, task-oriented behaviors directly process sensor data for making decision about the actions to be performed. There is no intermediate representation of the world between perception and action. This is the fundamental issue that allows these systems to immediately react to unpredicted situations. This architecture has been proved to be practically usable in realizing basic functionalities for mobile robots embedded in real environments.

However, the main problem with this approach is that it does not allow the designer to consider general aspects of perception (not related to a specific behavior), and to identify complex situations. In fact, the use of a symbolic high-level language is not possible, since it would necessarily require building a world model, and thus reasoning is usually compiled into the structures of the executing program, and automatic plan generation is either not addressed or considered as specifying reactions for each possible situation.

The lack of previsions about the future in these systems is the main cause of inefficient and unsuccessful goal achievement. Moreover, the impossibility to make use of a high-level language for describing the environment and the agent's goals is a severe limitation for the design and the realization of agents embedded in complex environments.

2.1.3 Hybrid Architectures

It was soon clear that the integration of reasoning and reactive capabilities was the key point in order to build actual agents. Therefore a lot of work on this topic has been done in the recent years and layered architectures were taken into consideration with the aim of implementing the integration of deliberation and reactivity.

We can roughly describe a layered hybrid architecture of an agent with two levels: the high-level (deliberative level), in which a high-level state of the agent is maintained and decisions on which actions are to be performed are taken, and the low-level (operative level) in which conditions on the world are verified and actions are actually executed. The central issue on devising hybrid architectures is the definition of interfaces among modules. Indeed an effective integration of reasoning and reactivity is deeply related to the interaction among the architecture modules.

Many attempts in classifying hybrid architectures (see for example [4, 69]) have shown that it is not easy to make effective comparisons among them, mainly because of their strict connection with applications. However, among all the distinguishing features of robotic architectures, we consider the information representation techniques to be the most relevant for designing cognitive robots.

In fact, we can distinguish systems [44, 83, 75] in which a single representation of the world is used by both the reasoning and the control system from those [40] in which an heterogeneous representation of information is present. Observe that the first kind of systems presents a simpler architecture, in which reasoning and control procedures are specifically developed for the used representation language. On the contrary, heterogeneous architectures present a more general solution, allowing for combining classic AI techniques with reactive control mechanisms. The main advantage of heterogeneous architectures is thus the flexibility of developing different techniques by making use of the proper formalisms.

2.2 Knowledge Representation and Reasoning

In order to accomplish its tasks the agent needs to make decisions based on the information it has on the environment, in other words tools for reasoning on the agent's knowledge are needed. The general objective is to define a formal language for representing dynamic systems and reasoning procedures on this language.

Many formalisms for representing dynamic systems adopt a change-based approach in which an explicit representation of time is not present. So the assumptions of instantaneous actions and immediate effects are always made. An alternative approach is to explicitly represent time and develop temporal logics for reasoning about time [2, 3].

In the change-based approach, dynamic systems are typically modeled in terms of state evolutions caused by actions. A *state* represents a situation the system can be in, and is characterized by a set of properties which forms a *complete* description (with respect to some logic/language) of the represented situation. *Actions* cause state transitions, making the system evolve from the current state to the next one.

In principle we could represent the *behavior* of a system, i.e. all its possible evolutions, as a *transition graph*, where each node denotes a state, and is labeled with the properties that characterize the state, and each arc denotes a state transition, and is labeled with the action that causes the transition. Note that building the agent's transition graph requires *complete knowledge* on the possible behavior of the system. However, in a real environment, generally some pieces of information are not available at design time or change in time. Therefore the designer needs a formalism able to represent and reason on incomplete knowledge and to acquire new knowledge by means of sensing (or knowledge-producing) actions.

By using a logical framework for representing a dynamic system, agent's knowledge is phrased

in *axioms* of some logic. These axioms select a subset of all possible transition graphs. All the selected graphs are similar, since they all satisfy the same axioms, but yet different with respect to those properties not imposed by the axioms. Moreover the actual behavior of the system is denoted by one of the selected graphs. Hence one has to concentrate on those properties that are true in all the selected graphs, i.e. those properties that are *logically implied* by the axioms.

Some general problems concerning the representation of dynamic systems (and that are independent of the chosen formalism) are the well-known qualification, ramification, and frame problems.

The qualification problem consists of expressing all the facts that must be true in order to execute an action. Generally it is not possible to describe all the preconditions of an action. For example, we should write that, in order to start the engine of our car, we need that the battery is ok, that we have the right key, that the gas tank is not empty, that all electric wires are properly connected, that there are no bananas in the tail-pipe and so on. This problem is usually addressed by specifying a subset of all the possible conditions as the precondition of an action and by relying on ad hoc procedures for finding out what has gone wrong (if necessary).

The ramification problem is quite similar to the previous one, but concerns the postconditions of the actions; these are the conditions that will be true as a consequence of performing the action. So, in this case, we must describe all the changes that occur when an action is executed. For example moving a box affects not only the position of the box, but also of all the objects in it. A possibility for solving this problem when using a logical framework is to impose state constraints on conditions. For example we should write an axiom expressing that all the objects in a box have the same position of the box.

The frame problem involves all the conditions that do not change when an action is executed. Hand-coding all these conditions is a long and tedious job and makes difficult an incremental construction of the world description. For example, we should write down that moving an object does not change its color, the color of other objects, the position of other objects, and so on, and, whenever a new object is introduced in the world description, we have to write down a lot of new frame axioms. Many different solutions to the frame problem were proposed depending on the formalism chosen for dynamic system representation: both monotonic and non-monotonic solutions when using logical frameworks, while procedural solutions are possible when describing the world with action description languages.

Once the dynamic system has been modeled, different reasoning services are possible: (i) temporal projection, that is predict the results of the application of a specified sequence of actions starting from an initial state; (ii) plan generation, that is generate a plan (a sequence of actions for example) that, when executed, allows the agent to reach a specified goal; (iii) backward projection, that is reconstruct the state of the world at some earlier point in time starting from the current state and the sequence of actions performed.

Developing reasoning tools for autonomous agents deeply relates to the choice of the formalism used for representing the agent's knowledge. The framework for representing such knowledge is thus one of the main design element for the agent, and the designer has a variety of choices. We present in the following sections the most relevant proposals for representing dynamic systems and reasoning about actions.

2.2.1 Situation Calculus

Situation Calculus is one of the most relevant proposal of logical framework for representing dynamic systems [64, 72, 73].

The dynamic system is in a certain *situation* at a certain point in time. The transition to a new situation is caused by an *action*. The situation resulting from the execution of the action a in

the state s is denoted by $do(a, s)$ and S_0 denotes the initial situation. For example

$$do(putdown(A), do(walk(L), do(pickup(A), S_0)))$$

is the situation resulting from the application of the sequence of actions $[pickup(A), walk(L), putdown(A)]$ in the initial situation.

Fluents are properties that depend on situations. For example, $holding(r, x, s)$ denotes that robot r is holding the object x in the situation s.

Preconditions of the actions a are described by the axiom

$$Poss(a, s) \equiv \pi_a(s)$$

where $\pi_a(s)$ denotes all the conditions that must be true in order to execute a in the situation s. Moreover the following axioms describe the effect of action a on the fluent F

$$Poss(a, s) \wedge \gamma_F^+(a, s) \Rightarrow F(do(a, s)) \quad (2.1)$$
$$Poss(a, s) \wedge \gamma_F^-(a, s) \Rightarrow \neg F(do(a, s)) \quad (2.2)$$

where $\gamma_F^+(a, s)$ $[\gamma_F^-(a, s)]$ denotes the conditions under which the action a will lead to a situation in which F is true [false].

For example, the preconditions of the action *pickup*, and its effect on the fluent *holding* can be written as

$$Poss(pickup(r, x), s) \equiv [\forall z \neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextto(r, x, s)$$
$$Poss(pickup(r, x), s) \Rightarrow holding(r, x, do(pickup(r, x), s))$$

The solution to the frame problem, described in [71, 72], is based on the following completeness assumption: the term $\gamma_F^+(a, s)$ $[\gamma_F^-(a, s)]$ describes all the conditions under which action a leads to a state in which F is true [false]. Under this assumption it is possible to write a *successor state axiom* for the fluent F

$$Poss(a, s) \Rightarrow [F(do(a, s)) \equiv \gamma_F^+(a, s) \vee (F(s) \wedge \gamma_F^-(a, s))]$$

In this way the number of axioms to write is reduced from $2nm$ to $n+m$, where n is the number of actions and m is the number of fluents.

The planning problem, as usual in deductive planning [42], can be expressed by the following logical implication:

$$Axioms \models \exists s.Goal(s)$$

in which *Axioms* contains all the action descriptions, *Goal* is a formula describing the goal, and s is a term of the form $do(a_1, do(a_2, ..., do(a_n, S_0)..))$ representing the situation resulting from executing the actions $a_n, .., a_1$ on the initial situation S_0.

Unfortunately solving this problem is computationally very hard. Instead of searching for a plan that achieves a goal, it is possible to verify properties of high-level programs provided by a user, such as termination and correctness.

Complex actions can be derived from simple ones by applying operators like test, sequence, nondeterministic choice of actions, etc. Given a complex action (or *program*) δ, we can define a *macro* $Do(\delta, s, s')$, indicating that s' is the terminating situation of an execution of the complex action δ from situation s, in the following way [59].

1. Primitive actions:

$$Do(a, s, s') \doteq Poss(a, s) \wedge s' = do(a, s)$$

2. Test actions:
$$Do(\phi?, s, s') \doteq \phi \wedge s = s'$$

3. Sequence:
$$Do([\delta_1; \delta_2], s, s') \doteq \exists s''.Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$$

4. Nondeterministic choice of actions:
$$Do((\delta_1 \mid \delta_2), s, s') \doteq Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$$

5. Nondeterministic choice of action arguments:
$$Do((\pi x)\delta(x), s, s') \doteq \exists x Do(\delta(x), s, s')$$

6. Nondeterministic iteration:
$$Do(\delta^*, s, s') \doteq (\forall P).\{(\forall s_1)P(s_1, s_1) \wedge (\forall\ s_1, s_2, s_3)[P(s_1, s_2) \wedge Do(\delta, s_2, s_3) \Rightarrow P(s_1, s_3)]\} \Rightarrow P(s, s')$$

Conditionals and while loops can thus be defined as follows:

$$\begin{aligned} \textbf{if}\ \phi\ \textbf{then}\ \delta_1\ \textbf{else}\ \delta_2 &\doteq [\phi?; \delta_1] \mid [\neg\phi?; \delta_2] \\ \textbf{while}\ \phi\ \textbf{do}\ \delta &\doteq [[\phi?; \delta_1]^*; \neg\delta?] \end{aligned}$$

Given a set of axioms describing the environment, the properties of a program δ that can be proved are the following.

1. Termination:
$$Axioms \models \exists s.Do(\delta, S_0, s)$$

2. Correctness:
$$Axioms \models \forall s.Do(\delta, S_0, s) \Rightarrow G(s)$$

The definitions above provide the basis for the development of a high-level programming language, that is called GOLOG. In [59] a Prolog interpreter for the GOLOG language is also discussed. Executing a GOLOG program δ amounts to finding a ground situation term $s = do(a_n, .., do(a_1, S_0)..)$ such that
$$Axioms \models Do(\delta, S_0, s)$$
The sequence of actions $[a_1, .., a_n]$ is thus the actual execution of δ.

GOLOG has been also used for plan generation [73]. In fact simple breadth-first and depth-first planners are written with GOLOG language, allowing for the generation of plans as (partially ordered) sequences of actions. In order to increase the efficiency of the planning procedure, goal dependent knowledge is added to the world description as in [7]. In this way bad plans are detected very soon and the size of the search space is significantly reduced.

Notice that this approach in using GOLOG language is quite different from what we have seen before that aims to writing GOLOG programs for controlling a robot. Indeed here we are not using GOLOG for writing a control program, but for implementing a planner that generates sequences of actions to achieve a goal. The expressivity of the plans generated in this way is very limited with respect to GOLOG language.

An extension for dealing with concurrent actions is presented in [26]. Concurrent processes are modeled as interleavings of the corresponding primitive actions.

The language CONGOLOG is an extension of GOLOG with the following constructs:

$(\delta_1 \mid\mid \delta_2)$	concurrent execution
$(\delta_1 \rangle\rangle \delta_2)$	concurrent execution with different priorities
$\delta^{\mid\mid}$	concurrent iteration
$\langle \phi \rightarrow \delta \rangle$	interrupt

A concurrent execution of two processes is given by an interleaved sequence of the primitive actions of two programs. Priorities impose a constraint on the possible interleaving of actions: the lower priority process will execute only if the higher priority one is either terminated or *blocked* (e.g. it is waiting for a condition to become true for executing the next action). Concurrent iteration is the concurrent execution of a number of instances of the process. Finally, interrupts $\langle \phi \rightarrow \delta \rangle$ express that process δ will execute whenever the conditions ϕ becomes true. Observe that interrupts are the fundamental mechanism for providing the agent with *high-level reactivity* [57].

When using a logical framework it is possible to rely on epistemic representations of the agent's knowledge in order to formalize knowledge producing (or sensing) actions. The possibility of expressing and reasoning about the epistemic state of the agent has been studied in the framework of the Situation Calculus in [77, 58]. The approach is based on the standard possible-world model of knowledge.

A binary relation among states has been introduced to express possible states of the world. Specifically $K(s', s)$ expresses that s' is accessible from s, that is in situation s it is possible for the agent to consider the world being in situation s'.

A fluent P is known in situation s iff P is true in every situation s' considered possible. $\mathbf{Knows}(P, s)$ is thus defined as

$$\mathbf{Knows}(P, s) \doteq \forall s' K(s', s) \Rightarrow P(s')$$

The successor state axiom for the fluent K has the following form when a is not a sensing action

$$Poss(a, s) \Rightarrow [\, K(s'', do(a, s)) \equiv \exists s' (K(s', s) \wedge (s'' = do(a, s')))\,]$$

expressing that the state $do(a, s')$ is accessible from any state $do(a, s)$ such that $K(s', s)$.

Instead, when a_P is a sensing action on the fluent P, the successor state axiom has the form

$$Poss(a_P, s) \Rightarrow [\, K(s'', do(a_P, s)) \equiv \exists s' (K(s', s) \wedge (s'' = do(a_P, s')) \wedge (P(s) \equiv P(s')))\,]$$

In this way $do(a_P, s')$ is accessible from any state $do(a_P, s)$ such that $K(s', s)$ and $P(s) \equiv P(s')$, that is P is known in the state s, or, in other words, either $\mathbf{Knows}(P, s)$ or $\mathbf{Knows}(\neg P, s)$ holds.

2.2.2 Propositional Dynamic Logics

Propositional Dynamic Logics (PDLs) have been introduced as a formal system for reasoning about computer programs and successively have been proposed as a formalism to represent dynamic systems [74, 24].

Syntactically, a PDL is constituted by expressions of two sorts: *programs*, in our case *actions*, and *formulae*. Formulae denote properties of states, and actions denote state transitions from one state to another. The dynamic system itself is described by means of axioms. Two kinds of axioms are introduced, "static axioms", that describe background knowledge about states, and "dynamic axioms", that describe how the situation changes when an action is performed. As in the deductive-planning tradition, a plan can be generated by finding a constructive existence proof for a state where the desired goal is satisfied. In a PDL setting a plan consists of a sequence of transitions, which leads to a state satisfying the goal.

The semantics of propositional dynamic logics is based on the notion of (Kripke) structure, and the basic semantical relation is "a formula ϕ holds at a state s of a structure". In particular, the modal expressions $[R]C$ and $\langle R\rangle C$ are interpreted as "for all possible executions of the action R, C holds in the successor state" and "there exists an execution of R such that in the successor state C holds".

Following the framework of [74] two kinds of axioms are distinguished:

- *Static axioms*, which are used for representing background knowledge that is invariant with respect to the execution of actions. In other words, static axioms hold in any state and do not depend on actions.
- *Dynamic axioms*, which are introduced to describe under which circumstances it is possible to execute an action and the changes actions bring about, are distinguished in *action precondition axioms*, that have the form
$$C \Rightarrow \langle R\rangle \top$$
and *effect axioms*, with the form
$$C \Rightarrow [R]D$$
in which R is an action, C represents the *preconditions* that must hold in a state, in order for the action R to be executable; D denotes the *postconditions* that are true in the state resulting from the execution of R in a state where preconditions C hold. Multiple axioms per action are allowed.

In deductive planning one is typically interested in answering the following question: "Is there a sequence of actions that, starting from an initial state, leads to a state where a given property (the goal) holds?". Under the assumption of deterministic actions, this is captured by the following logical implication (here we phrase it in PDLs):

$$\Gamma \models S \Rightarrow \langle \alpha^* \rangle G \tag{2.3}$$

where: (i) Γ is the set of both static and dynamic axioms representing the (partial) knowledge about the system; (ii) S is a formula representing the (partial) knowledge about the initial situation (state); (iii) G is a formula representing the goal, which is, in fact, a (partial) description of the final state one wants to reach; (iv) $\langle \alpha^* \rangle G$ (where $\langle \alpha^* \rangle G$ stands for any formula of the form $\langle R_1\rangle\langle R_2\rangle \ldots \langle R_n\rangle G$ with $n \geq 0$ and R_i any action) expresses the existence of a finite sequence of actions leading to a state where G is satisfied. From a constructive proof of the above logical implication one can extract an actual sequence of actions (a plan) that leads to the goal.

An extension of PDL is represented by the logic $\mathcal{DIFR}$ [25]. It is a very powerful logic that allows also for dealing with nondeterministic and concurrent actions.

The result on the computational properties of PDLs shows that they are decidable, and that logical implication both in the simplest PDL and in the richer $\mathcal{DIFR}$ is EXPTIME-complete. Obviously, the high computational cost is the main cause for the lack of implemented systems based on these logics.

2.2.3 Action languages $\mathcal{A}$, $\mathcal{AR}$,...

A family of languages for representing actions and formalizing dynamic systems were developed in the last years [41, 48, 9].

The first and the simplest language is called $\mathcal{A}$ [41]. It is characterized by two disjoint set of symbols: *fluent names* and *action names*. The domain is described by a set of propositions, including

- *value propositions*, specifying the value of a fluent after performing a sequence of actions:

$$F \quad \textbf{after} \quad A_1; ...; A_m$$

- *effect propositions*, that describe the effects of actions:

$$A \quad \textbf{causes} \quad F \quad \textbf{if} \quad P_1, ..., P_m$$

where $A_1, .., A_m, A$ are action names and $F, P_1, ..., P_m$ are *fluent expressions*, that are fluent names possibly preceded by $\neg$.

The semantics of the language $\mathcal{A}$ is based on *structures* (σ_0, Φ), where σ_0 is a *state* (i.e. a set of fluent names), and Φ is a *transition function*, that is a mapping from the set of pairs (A, σ) (where A is an action name and σ is a state) into the set of states. For a complete description of the semantics of the language we refer to [41, 48].

Let us observe that in this language a state is completely specified by the fluents that are true in it. Therefore it is not possible to describe a situation in which a fluent is not known. Furthermore the frame problem is solved by imposing the transition function to maintain the value of a fluent in the successor state, if the action does not change its value.

An extension of the language $\mathcal{A}$ for dealing with indirect effects of actions (ramification) and nondeterministic actions is called $\mathcal{AR}$ [48]. It is defined by a set of *action names*, a set of *fluent names*, a function mapping every fluent name into a nonempty set of range values (that is called *range of* F), and a subset of fluent names called *inertial*.

A *formula* is either an *atomic formula* of the form (F **is** V) (where F is a fluent name and V is a value in the range of F), or a propositional combination of atomic formulas. An example of formula is

$$Position \textbf{ is } OnTable \ \wedge \ Color \textbf{ is } Red$$

When the range of a fluent F is $\{True, False\}$ we abbreviate (F **is** $True$) with F and (F **is** $False$) with $\neg F$.

The domain is described by a set of propositions of the following types:

- *value propositions*, specifying the value of a formula after performing a sequence of actions:

$$C \quad \textbf{after} \quad A_1; ...; A_m$$

- *determinate effect propositions*, that describe the effects of actions:

$$A \quad \textbf{causes} \quad C \quad \textbf{if} \quad P$$

- *indeterminate effect propositions*, that describe nondeterministic effects of actions:

$$A \quad \textbf{possibly changes} \quad F \quad \textbf{if} \quad P$$

- *domain constraints*, that are formulas that are valid in every state:

$$\textbf{always} \quad C$$

where $A_1, .., A_m, A$ are action names and F is a fluent, and C, P are formulas.

The semantics of language $\mathcal{AR}$ is described in [48]. The state is represented by a set of atomic formulas that are satisfied in that state. Again incomplete specification of states are not allowed.

Many other languages of this family have been proposed (e.g. $\mathcal{AR}^-$ [48], $\mathcal{AR}_0$, etc.) that include different features, such as parameters, concurrency, etc.

Among all, we describe a language, called $\mathcal{A}_\mathcal{K}$ [62], that allows for representing incomplete knowledge on the environment and for describing sensing actions. It is characterized by three types of propositions:

- *value propositions*, specifying the value of a fluent in the initial situation:

$$\textbf{initially} \quad F$$

- *effect propositions*, that describe the effects of actions:

$$A \quad \textbf{causes} \quad F \quad \textbf{if} \quad P_1, ..., P_m$$

- *knowledge-producing propositions*, that describe the effects of sensing actions:

$$A \quad \textbf{determines} \quad F \quad \textbf{if} \quad P_1, ..., P_m$$

Observe that in this language the mental state of the agent, rather than the absolute state of the system, is represented. So incomplete knowledge on the environment can be modeled. In fact, the semantics of $\mathcal{A}_\mathcal{K}$ [62] is characterized by the definition of a *state* as a pair of disjoint sets of fluent names. A fluent F holds in a state $\sigma = \langle \Gamma, \Delta \rangle$ iff $F \in \Gamma$, it does not hold in σ iff $F \in \Delta$, and it is not known in σ iff $F \notin \Gamma \cup \Delta$. Knowledge-producing propositions make the value of a fluent F to be known, since the effect of a sensing action is a new state where F is added to either the set Γ or Δ.

The implementation of reasoning procedures on these languages have been devised in many recent works. In [8] a logic programming approach for plan generation in the language $\mathcal{A}_1$, which is an extension of $\mathcal{A}$, is presented. The paper defines a method for translating a domain description D in the language $\mathcal{A}_1$ to a logic program πD. The logic program is proved to be *acyclic* and this guarantees soundness, completeness, and decidability of SLDNF resolution. In [61] a partial order planner for language $\mathcal{A}_\mathcal{K}$ is described. Finally, a planning system for the language $\mathcal{AR}$, which is based on model checking techniques, is presented in [20].

2.2.4 STRIPS

STRIPS was one of the first languages for representing a dynamic system [37], and it is the base for the implementation of many efficient planning systems. In fact, the first significant application of high-level control of mobile robots can be found in Shakey the robot [67], in which STRIPS was used for representing the environment and for planning.

A planning problem in propositional STRIPS [17] is defined as a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where:

- $\mathcal{P}$ is a finite set of *conditions*;
- $\mathcal{O}$ is a finite set of *operators*, where each operator is defined as $\langle \phi, \eta, \alpha, \delta \rangle$, where:

  $\phi \subseteq \mathcal{P}$ is a set of *positive preconditions*,
  $\eta \subseteq \mathcal{P}$ is a set of *negative preconditions*,
  $\alpha \subseteq \mathcal{P}$ is a set of *positive postconditions*,
  $\delta \subseteq \mathcal{P}$ is a set of *negative postconditions*,
  $\phi \cap \eta = \emptyset$ and $\alpha \cap \delta = \emptyset$;

- $\mathcal{I} \subseteq \mathcal{P}$ is the *initial state*;
- $\mathcal{G} = \langle \mathcal{M}, \mathcal{N}, \rangle$ is the *goal*:

  $\mathcal{M} \subseteq \mathcal{P}$ is a set of *positive goals*,
  $\mathcal{N} \subseteq \mathcal{P}$ is a set of *negative goals*, and
  $\mathcal{M} \cap \mathcal{N} = \emptyset$.

A state is described by a subset of $\mathcal{P}$, that includes all the conditions that are true in that state. Therefore a state is always completely specified since a condition is either true or false in that state. In particular, this setting requires a complete description of the initial state. A state $S \subseteq \mathcal{P}$ satisfies a goal $\mathcal{G}$ iff $\mathcal{M} \subseteq S$ and $S \cap \mathcal{N} = \emptyset$.

The effect of a finite sequence of operators $(O_1, ..., O_n)$ on a state S is formalized by the transition function *Result*, that is defined as follows:

$$\begin{aligned} Result(S, ()) &= S \\ Result(S, (O)) &= \begin{cases} (S \cup \alpha) \backslash \delta & \text{if } \phi \subseteq S \wedge S \cap \eta = \emptyset \\ S & \text{otherwise} \end{cases} \\ Result(S, (0_1, ..., O_n)) &= Result(Result(S, O_1), (O_2, ..., O_n)) \end{aligned}$$

A finite sequence of operators $(O_1, ..., O_n)$ is a *solution* to a planning problem iff $Result(S, (0_1, ..., O_n))$ statisfies the goal $\mathcal{G}$. A planning problem is *satisfiable* if it has a solution. Determining whether a planning problem is satisfiable (PLANSAT) in the general setting of propositional STRIPS is PSPACE-complete [17]. However, by restricting the expressivity of the operators, it is possible to retrieve tractability. For example, if operators have only positive preconditions and one postcondition then PLANSAT is polynomial [17].

A large amount of work has been done on practical realization of efficient planning systems (e.g. UCPOP [68], Graphplan [11], etc.), which rely on a STRIPS-based representation of the world. Although they present different features regarding efficiency and the generation of partial order plans, all of them are affected by two common problems: (i) the limited expressivity of the language, that requires a complete knowledge over the world, (ii) no context dependent effects are allowed, that is a solution to the ramification problem is missing.

Furthermore, a proposal for planning in presence of incomplete information in STRIPS domain appears in [35]. The UWL language is an extension of STRIPS, where the truth value of a predicate can be either *True*, *False*, or *Unknown*. In this way it is possible to express incomplete knowledge on a state by setting predicates to *Unknown*. Sensing actions are distinguished by means of an annotation mechanism on the postconditions (*observational postconditions*). The planner makes use of these annotations in order to ensure that predicates values are *observed* before their use. Conditional branches are due to different possible values of predicates and the planning algorithm generates a plan for every possible value and then combines the two plans. A further relevant issue about this approach is the definition of *information goals*, that are goals aiming to achieving a state where the value of a property is known (either *True* or *False*). Indeed this is an important feature for information gathering agents.

2.2.5 Summary

Table (2.1) summarizes the features of the most common formalisms for representing dynamic systems and reasoning about actions. The table is divided in two parts: besides the language name, the left side (the first four columns) shows expressivity of the language by the capability of defining domain constraints (DC), nondeterministic (ND), concurrent (Co), and sensing (Se)

Language	DC	ND	Co	Se	P.Ver	Plan	C.Plan
GOLOG	Yes	Yes	No	No	Yes	R.L.	No
CONGOLOG	Yes	Yes	Yes	No	Yes	No	No
PDL	Yes	Yes	No	No	No	No	No
$\mathcal{DIFR}$	Yes	Yes	Yes	No	No	No	No
$\mathcal{AR}$	Yes	Yes	No	No	Yes	Yes	No
$\mathcal{A}_{\mathcal{K}}$	No	No	No	Yes	Yes	Yes	Yes
$\mathcal{A}, \mathcal{A}_1$	No	No	No	No	Yes	Yes	No
STRIPS	No	No	No	No	Yes	Yes	No
UWL	No	No	No	Yes	Yes	Yes	Yes

Table 2.1: Summary table

actions; while the right side is focused on implementation issues, that are program properties verification (P.Ver), plan generation (Plan), and conditional plan generation (C.Plan). We use the symbol "R.L." to denote that a planner system has been implemented on a restricted language with respect to the original one.

Notice that the expressivity of the language is a critical factor for actual implementation of reasoning procedures and in particular for plan generation. Therefore efficient implementation are developed only for very limited languages.

On the other hand, the reduced expressivity of some of these languages, and above all the assumption of having complete information, are severe limitation for their use in actual realizations of autonomous agents embedded in real environments.

2.3 Discussion

Recent work on Cognitive Robotics is focused on the application of action theories and reasoning procedures for high-level controlling an actual mobile robot. In the literature there are several examples of robots that are controlled by relying on a declarative representation of actions. In some cases there are also common application domains (such as mail delivery in an office environment), that allows for a comparison of the different systems.

A mail delivery prototype system implemented on a RWI B12 mobile robot is described in [57]. This work is mainly concerned with high-level reactivity. In fact, by taking advantage of the interrupt mechanism of CONGOLOG, it is possible to modify the behavior of the robot depending on the current situation. For example, when a high priority delivery task comes to the system, it must be able to interrupt the current task and execute the higher priority one.

The language CONGOLOG is used for describing a high-level program for the mail delivery robot, and the CONGOLOG interpreter is in charge of extracting from this program and from the

description of the environment a sequence of actions that must be executed.

Observe that in this context high-level programs are much more general than plans (considered as a sequence of actions). Indeed these programs contain loops, nondeterministic and concurrent actions, and prioritized interrupts. Observe also that high level programs are written by the user, and the interpretation process is responsible for executing them.

In [57] the possibility of acquiring new knowledge during task execution has not been addressed, in fact sensing actions are not considered. For example, the property of a door to be open or closed is not modeled at high-level, and thus the capability of dealing with possibly closed doors is left to the low level. However in this way it is not possible to reason about entering a room through a second door, if the first one is closed.

Although knowledge producing actions can be added to the high level program of the robot (see also [58]), there is again a problem in the domain description: the high level action *goToRoom(r)* is not admissible anymore, but it must be splitted into at least the two actions *goToDoor(d)* and *enterDoor(d)*.

Therefore the high-level program must be rewritten and the low-level controller must be reprogrammed for executing the new actions. This process can be more or less costly, depending on the used tools, but nevertheless it highlights the lack of a systematic design process.

In [81, 82] another mail delivery prototype based on a Khepera robot is described. The formalism for reasoning about actions is based on circumscriptive event calculus [81]. The planning problem is expressed as an abductive task in the event calculus, and a logic programming implementation has been developed for plan generation.

The integration between planning and reactivity is achieved by interleaving sensing, planning, and acting. In every cycle a bounded amount of time is given to each of these tasks: sensor processing, planning, and actions execution. This approach involves a synchronous protocol among sensing, planning, and acting that forces a limitation on the reaction time of the robot.

Another problem with this approach is that it does not offer the possibility of taking advantage of new knowledge acquired during task execution. In fact, it is not possible to specify that a property is unknown (e.g. "the robot does not know whether door 2 is open or closed"), it is not possible to define knowledge producing actions (e.g. "sense if door 2 is open or closed"), and thus there is no way to generate conditional plans that can be effectively executed starting from a partially known initial state (e.g. "go to door 2; if door 2 is open enter room 3 through door 2, else go to door 4 and if door 4 is open enter room 3 through door 4").

In both the previous systems the relation between the robot's architecture and the logical framework for reasoning about actions has not been explicitly addressed. The link is in the implementation of high level atomic actions with low level control procedures. However this is a critical point in actual realization of cognitive mobile robots and must be considered in the early stages of the design process. Indeed, only by taking into account as soon as possible the integration of the knowledge representation framework and the robot's architecture, it is possible to define an "optimal" balance between the degree of abstraction of high level atomic actions and the difficulty in realizing control procedures that respect the specifications of these high level actions.

We believe that the integration of a logical formalism for reasoning about actions within a robot architecture is a central problem in the design of cognitive agents, but it has not been extensively studied. In other words, we believe that a global design methodology showing how to build actual cognitive agents, by taking into account at the same time aspects regarding knowledge representation and architecture, is missing.

Chapter 3

The Mobile Robot Base

In this chapter we present the mobile robot platform that is the base on which we have built our *cognitive robot*. The robot base belongs to the *Erratic* family[1] [50].

Figure 3.1 shows two different platforms that have been used for experiments. The leftmost one is an *Erratic* platform with basic hardware, while the rightmost one is a *Pioneer I AT* with increased sensing capabilities provided by a mobile stereo camera.

3.1 Erratic Hardware Devices

The *Erratic* basic mobile platform is equipped with a pair of electric motors driving two independent wheels and with two kinds of sensors: motor encoders and ultrasonic sonars. The sensing capabilities of the robot have been augmented by adding a black and white stereo camera.

The on-board computation, provided by a Motorola 68HC11 microprocessor, is responsible for low-level control of the hardware devices. The connection to a remote host is possible either by a direct connection through a serial port, or via radio modems.

3.2 Saphira Architecture

The Pioneer robots present a client-server architecture, such that the on-board computation implements a server that responds to client requests for low-level control of the robot.

[1] The commercial version of this platform is known as Pioneer I produced by ActivMedia Inc..

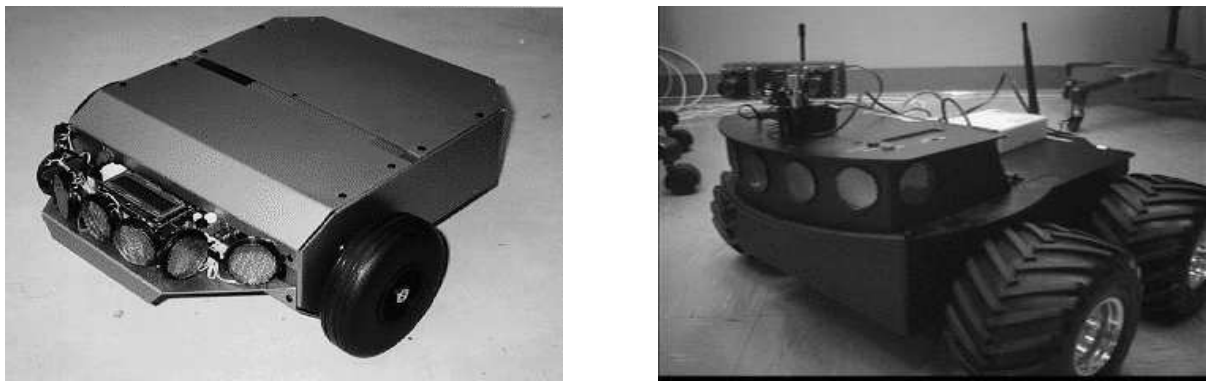


Figure 3.1: Erratic and Pioneer mobile robots

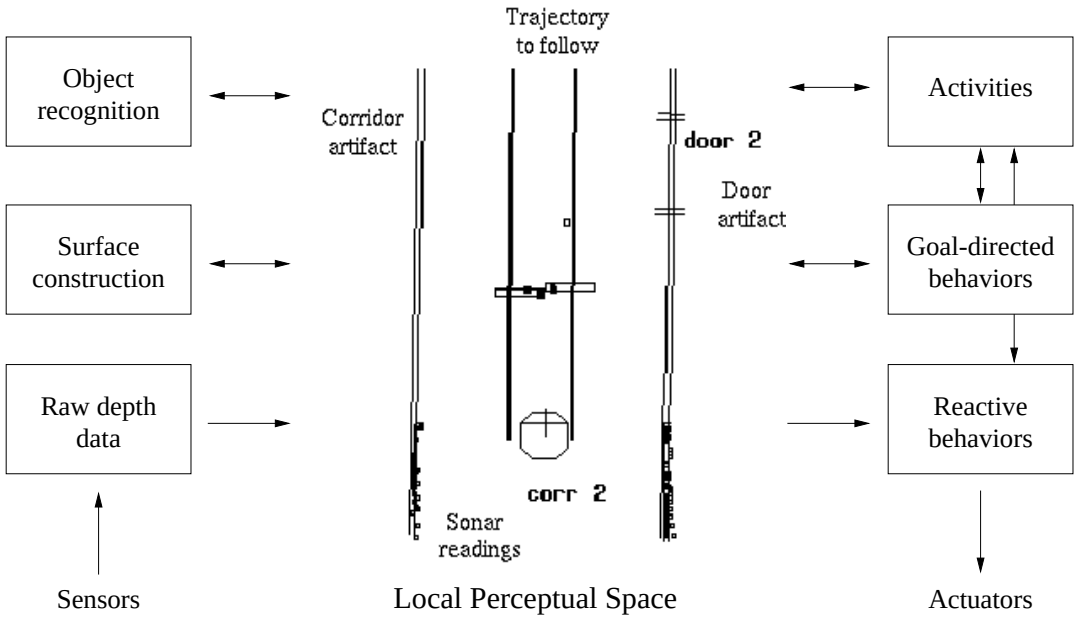


Figure 3.2: Saphira architecture

The robot control system relies upon the Saphira architecture [53], a behavior-based fuzzy controller, that provides integrated routines for sensing and control.

The Saphira architecture is depicted in Fig. 3.2. The Local Perceptual Space (LPS) is an intermediate representation of information about the environment that provides the connection between perception and action. Information in the LPS are represented at different levels of abstraction in order to provide the appropriate input for both goal-directed behaviors and reactive behaviors.

Perceptual data analysis and behaviors execution are decoupled by means of the LPS, in the sense that the former takes care of maintaining an up to date version of LPS, while the latter exploits information in LPS in order to control the robot hardware devices.

This architecture presents a vertical decomposition that is due to different levels of perception and action. Reactivity is ensured by the different levels of data representation in the LPS, that allows time critical behaviors (such as obstacle avoidance) to make use of simple sensor data, that are quickly available.

3.2.1 Local Perceptual Space

The Local Perceptual Space is a geometric representation of the space around the robot. It contains several levels of interpretation of sensor information:

- an occupancy grid indicating object distances read by sonar sensors or the stereo camera;
- analytic representation of surfaces coming from range data interpretation;
- a description of the world in terms of structures characterizing the environment (e.g. walls, corridors, doors, etc.). These structures, that are called *artifacts*, represent the primary information source for goal-directed behaviors (such as "follow corridor") and are the basis for updating the position of the robot in the environment.

Artifacts in the LPS can be added by data interpretation routines when the sensed artifact does not match any existing artifact. Moreover it is possible for the user to provide the system with a preconstructed map of the environment, specifying the properties of the artifacts.

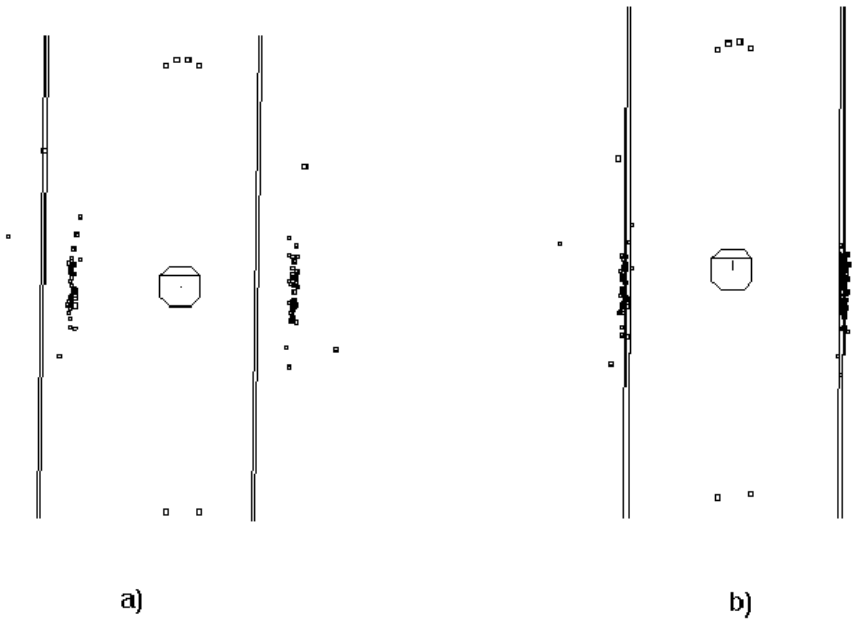


Figure 3.3: Updating artifacts position

3.2.2 Sensor data interpretation

Sensor data analysis is responsible for adding spatial information on the LPS, extracting features and generating surfaces from raw data, and computing properties of artifacts (such as the direction and the width of a corridor).

Since all the behaviors rely on information in the LPS, maintaining these data accurate and updated is a fundamental task. In particular the correct position of the robot with respect to some artifact is a critical element for the success of goal-directed behaviors. The necessity of updating the robot's position is given by the fact that, due to odometric errors in motor encoders, dead-reckoning mechanisms are not reliable over long distances.

Updating the artifacts in the LPS is performed in three steps. First a number of features, such as linear surfaces, are extracted from sensor data, then an object hypothesis based on the extracted features is done, and finally a matching process between the object hypothesis and the artifacts registered in the LPS produces information for updating the artifact's position with respect to the robot, and hence the robot's position in the environment.

In Fig. 3.3a there is a portion of the LPS, that contains sensor data, linear surfaces, and an artifact referred to a corridor. Notice how the displacement between the actual sensing data and the artifact has been removed after the matching process (Fig. 3.3b).

3.2.3 Fuzzy Controller

The control problem is decomposed into small control units, called *behaviors*, that are responsible for driving the robot according to their specification.

Saphira behaviors [75] are defined by:

- a *context*, that indicates the conditions for which the behavior is active;
- a set of *artifacts*, that are used as a reference for evaluating the current situation;

- a *control schema*, that expresses the control strategy and is defined by a desirability function $Des(s, a)$ denoting how much action a is desirable in situation s.

The desirability function is implemented by means of *fuzzy rules* of the form

$$\text{If } fuzzy\ expression \text{ Then } control\ set$$

A *fuzzy expression* is an expression involving fuzzy variables, that are variables assuming values into the interval $[0, 1]$. The *control set* is a function mapping a control variable in the interval $[0, 1]$, or, in other words, a fuzzy variable representing a control value. The intuitive meaning of a fuzzy rule is that the more the antecedent is true (close to 1), the more the consequent is high.

For example, consider the following rule

$$\text{If } ObjectInFront \text{ Then } TurnLeft$$

where *ObjectInFront* measures how much there is an object in front of the robot and *TurnLeft* indicates how much the robot should turn to its left. The effect of this fuzzy rule is that the more the robot is close to the object, the more the command "turn left" is enforced.

When many rules affect the same control variable, a process of *defuzzification* is required in order to obtain a single value for that variable. The defuzzification in Saphira [75] is implemented by the following weighted average of the desirability function

$$\hat{a} = \frac{\int a Des(s, a)}{\int Des(s, a)}$$

and produces a control $\hat{a}$ that is sent to the robot's actuators.

Behaviors are distinguished between *reactive behaviors* and *goal-directed behaviors*. Reactive behaviors are responsible for promptly reacting to unpredicted situations, by quickly processing simple sensing information. For example the *ObstacleAvoidance* behavior relies on a fast detection of objects in front of the robot and makes the robot turn in order to avoid them.

Goal-directed behaviors instead are characterized by a goal to be achieved and a set of artifacts as parameters. These behaviors can either succeed, when they reach the specified goal, or fail, if the goal is not reached (possibly after a time out). For instance, the *FollowCorridor* behavior makes use of the information provided by an artifact associated to the corridor for controlling the direction and the position of the robot, and a target point in the corridor to be reached.

Complex behaviors are obtained by combining basic behaviors by means of the following operators: *chaining*, that implements sequential execution, *conjunction*, that allows for concurrent execution, and *context-dependent blending*, that is a special form of concurrency (see [75] for details).

The control schema of a behavior resulting from context-dependent blending of more behaviors is defined by the following equation

$$Des(s, a) = (Cxt_1(s) \wedge Des_1(s, a)) \vee \ldots \vee (Cxt_n(s) \wedge Des_n(s, a))$$

where $Cxt_i(s)$ and $Des_i(s, a)$ are respectively the context and the desirability function of the behavior i. Observe that the control schema of each behavior $Des_i(s, a)$ is weighted by its context $Cxt_i(s)$. In this way behavior i affects the blended behavior only if it is in its own context.

Context-dependent blending is quite adequate for coordinating reactive and goal-oriented behaviors. Consider for example the combination of *AvoidObstacles* and *FollowCorridor*. The context of the first behavior is having an object in front, while the context of the second one is being in a corridor and not close to an object. Therefore, as long as there are no objects in front of the robot

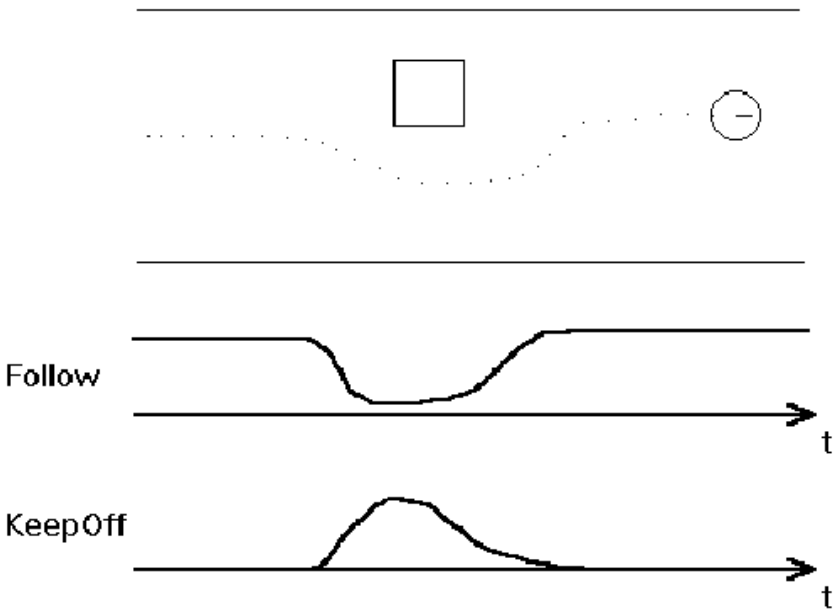


Figure 3.4: Context-depending blending

the control schema of *AvoidObstacles* is disabled and the control actions are given by *FollowCorridor*. However, when an object in front of the robot is detected, the desirability of executing actions from *AvoidObstacles* increases and eventually the robot turns to avoid the obstacle. Fig. 3.4 show the trajectory resulting from the blending execution of the two behaviors, as well as the value of their contexts over time.

3.2.4 Colbert language

We conclude the chapter by describing the Colbert language [51], a robot control language that is integrated in Saphira. Colbert programs, that are also called *activities*, include conventional sequential, conditional, and iterative constructs, and determine the sequence of actions performed by the robot.

The semantics of the language is given in terms of Finite State Automata (FSA), that represent the internal states of the robot in the nodes and state transitions in the arcs. However a Colbert activity looks much like a C procedure and, in fact, since the use of memory variables is allowed, it comes out to be more expressive than FSA. Furthermore, the use of a C-like language provides for programming and debugging facilities for the programmer.

The language supports concurrency and communication between activities. Concurrency is implemented by a round-robin cycle that manages state transitions of the running activities. The basic cycle time is 100 ms, which represents a good compromise between response time and activities switch overhead. Coordination among activities is achieved by a signaling mechanism. In fact, an activity can send and receive signals and ask for the internal state of another one.

Activities are also able to start other activities or behaviors and to monitor their execution. Consider, for instance, the following definition of an activity for reaching a room from corridor C through door D, while avoiding possible obstacles.

```
act goToRoom(corridor C, door D)
```

```
{
   start FollowCorridor(C,D) timeout 100;
   if succeeded(FollowCorridor)
      start EnterDoor(D2) timeout 100;
   else
      fail;
}


act maintask()
{
   corridor C;
   door D;

   C = ...;
   D = ...;

   start AvoidObstacles noblock;
   start goToRoom(C,D);
}
```

Objects C and D are artifacts in the LPS that must be initialized. The behavior *AvoidObstacles* is executed in parallel with the activity *goToRoom*, which includes a sequence of *FollowCorridor* and *EnterDoor*.

Colbert interpreter, integrated in Saphira, is a useful tool for the development of complex tasks, since it allows the user for writing and debugging high-level programs for controlling the robot.

Chapter 4

A Methodology for Building Cognitive Agents

In this chapter we describe a new design methodology for building cognitive agents. The objective is to realize a cognitive level on top of an agent able to perform basic tasks. Therefore, we do not provide here methods for implementing a single subsystem of the agent, but the general framework for combining (possibly existing) technologies in order to develop an actual agent.

The important feature of our methodology is that it attacks the problem of integrating reasoning and reactivity in the middle, that is by defining first the interface between the two levels. The main features of this approach are: (i) design errors are detected very soon during the development process; (ii) incremental design and development of the agent's capabilities is possible; (iii) a large amount of work can be reused when moving the agent to a new environment.

Previous approaches to the realization of cognitive robots [57, 82] follow a different approach: they focus the attention on defining a high-level description of the environment, while the integration within the reactive robot architecture is considered as a separate problem, solved by the implementation of high-level actions with some specific low-level control procedures. In other words, in these works the relation between the robot's architecture and the logical framework for reasoning about actions has not been explicitly addressed. However, we consider the integration of a logical formalism for reasoning about actions within a robot architecture to be the central problem in actual realization of cognitive mobile robots. Therefore, we believe that it must be considered in the early stages of a global design process. Indeed, only by taking into account as soon as possible the integration of the knowledge representation framework and the robot's architecture, it is possible to define an "optimal" balance between the abstraction degree of high-level atomic actions to be performed and conditions to be verified on the environment, and the difficulty in realizing reliable and robust low-level control procedures and data interpretation routines that comply the high-level specifications.

Another interesting feature of our approach is the possibility of making use of different existing techniques that have been developed and experimented with for solving some specific problems in the application domain. For instance, we made use of an existing fuzzy controller for our mobile robot.

Furthermore, it also offers a method for a comparison of different techniques in the same application domain. Indeed we can replace a module with another using a different technique, provided that it uses the proper interfaces with other modules.

4.1 Design choices

Before illustrating the methodology, we need to discuss some basic issues that characterize the design process.

The integration of reasoning and reactivity is usually obtained by hybrid layered architectures. They can be distinguished into two groups: *homogeneous* architectures, that make use of a single framework for representing knowledge needed by both the reasoning and the reactive system, and *heterogeneous* ones, that make use of different representation of the information. In the realization of actual cognitive agents, there are many specific problems to be solved, often depending on the application domain. An important feature of the design process is thus represented by the possibility of applying adequate approaches to solve each problem, making use of ad hoc (possibly existing and well-tested) techniques. We believe that the heterogeneous layered architectures offer an advantage in integrating different techniques [40].

The second important point is the representation of information in the agent. The complexity of real environments and of the tasks to be accomplished can be better described by a high-level symbolic language. A logical formalism for representing the agent's knowledge is thus the preferred way for dealing with the complexity of the world and it also provides the designer with extremely flexible and powerful development tools.

We believe that, since a high-level language is required for knowledge representation, dividing Knowledge Acquisition from Task Execution, in order to create an intermediate representation of the agent's state, is necessary. On the other hand the ability to quickly react to unexpected situations must be preserved.

Summarizing we make the following choices for the design of the agent.

- The agent will be implemented by an heterogeneous layered architecture, that includes an explicit representation of knowledge and combines reasoning and reactive capabilities.
- We will make use of a logical framework for knowledge representation and reasoning, in order to deal with the complexity of real environments and high level tasks.

We describe the architecture and the methodology in detail in the next sections.

4.2 Architecture

In order to allow for an effective integration of many different techniques, we propose a heterogeneous layered architecture (see Fig. 4.1), which combines a deliberative and an operative level. At the deliberative level reasoning tools are developed in order to maintain a model of the world and to select the actions that will lead to the agent's goals. While the operative level is mainly concerned with the implementation of all the techniques that will be used for sensor data interpretation and actual execution of actions.

The different levels are characterized by the degree of abstraction of data representation. Thus, Knowledge Acquisition and Task Execution are performed over two levels: a symbolic level (used by logic-based methods for reasoning about actions) and a numeric level (used by real-time control mechanisms). The important feature of this architecture is thus the presence of different representations (at different degree of abstraction) of the same information on the environments.

Two kinds of data run across these levels: the first one, representing *perceptual data*, move from the sensing devices to the deliberative level, and the second one, representing *action data*, move from the deliberative level to the actuators.

In contrast with some other previous proposals of architectures for integrating reasoning and reactivity [44, 83, 75], our architecture provides a solution based on different levels for representing information about the environment. The proposed architecture is also quite different from a

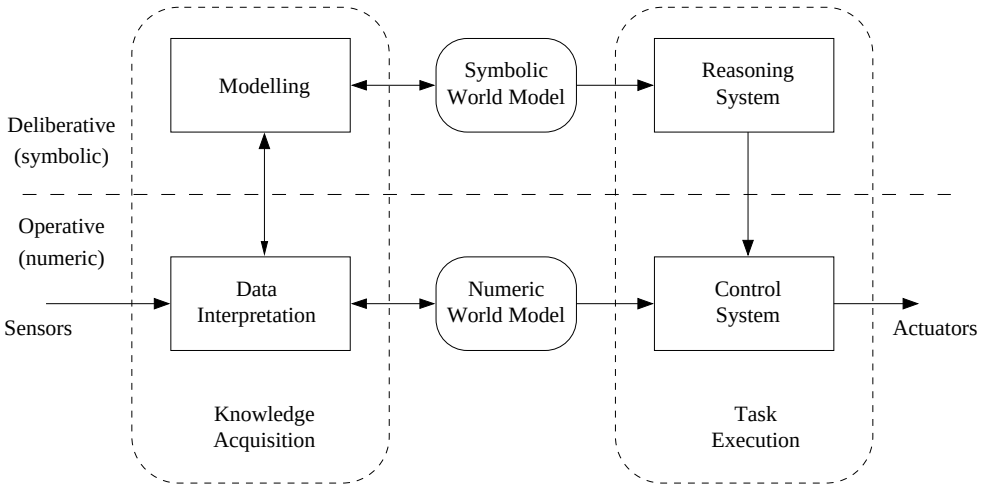


Figure 4.1: General layered architecture for cognitive agents

functional one [67, 82], since every level can take decisions if needed and there is not a predefined synchronous sense-plan-act cycle.

The presence of different models of the internal state of the robot, given by different levels for knowledge representation, is the effective technique that we use for integrating reasoning and reactivity. In this way, it is not always the case that the high-level state must be updated, when the low-level state changes. For example, while in the lower level we can represent the exact position of a robot in the environment, in the higher level we can be interested only in a topological position, for instance the robot being in a corridor. So, as long as the robot position changes within the corridor, there is no need to update the high-level state of the robot. Moreover some information never reach the higher level, but it is only used by the low-level control system (e.g. data for obstacle avoidance).

We consider this relation between the two world models to be a critical element for the development of cognitive agents, since updating a logic-based knowledge base can require high computational time and affect reactive abilities. Therefore it must be accurately defined in the design process.

The above schema presents four distinct modules for data interpretation, modelling the world, reasoning on the agent's knowledge, and controlling the execution of actions, that are described in the following sections.

4.2.1 Data Interpretation

The Data Interpretation module is responsible for capturing, interpreting and integrating perceptual data coming from several sensor devices. The main objective is to provide the other modules with all the information needed to understand changes in the world. To this end, the module maintains a data base of information characterizing the internal state of the agent and its beliefs on the environment.

There are three kinds of input data for this module: (i) raw data coming from sensors (e.g. digitized images, sonar readings, etc.); (ii) the current state of the agent stored in the data base; (iii) high-level information coming from the cognitive level (e.g. properties that are verified according to a deduction process). The outputs are new instances of the data base containing the recent updates, and information needed from the Modelling module for maintaining its high level world model.

Sensor data interpretation is generally divided into two parts: interpretation of raw sensor

data, in order to extract relevant information from raw data, and data integration, that involves analysis of the same information coming from different sensors. For example, consider a mobile robot equipped with ultrasonic sonars and a stereo vision camera. The input data are a pair of images of the scene viewed from the robot and the sonar readings. By processing these data with the appropriate techniques, one can retrieve range data about the objects that are around the robot. Now, since in general a situation is modeled by data values referred to the environment (e.g. the position of these objects with respect to the robot), the results of data analysis for the two sensors must be combined together.

As for information coming from the higher level, these are integrated into the low-level world model by updating the world model according to the high-level specifications. This process is necessary in order to maintain the two representations of the agent's state aligned.

Summarizing, the Data Interpretation module should allow the designer to define and implement both interpretation routines and integration techniques for sensor data.

4.2.2 Modelling the World

The Modelling module is in charge of providing a high-level description of the current situation in which the agent is embedded. It is the fundamental task that returns a feedback to the reasoning system on the actual execution of actions. Indeed observe that acquiring new knowledge on the environment involves detecting plan failures and external changes in the world.

This module usually builds a symbolic representation of the world from pre-processed structured sensor data, generating and maintaining a Knowledge Base (KB) containing axioms and facts about the environment. Notice that a part of such a KB, containing a general description of the environment and of the actions that the agent can perform, is usually defined a priori from the designer.

Moreover note that, during updates of the KB, it is possible that some properties change as an effect of a deductive process. In some of these cases, when a low-level component of the system can process one of these changed properties, it is necessary to communicate the new information to the lower level. For example, consider a robot's KB that includes a constraint on the environment specifying that either $Door1$ or $Door2$ is open. Suppose a situation in which the robot initially does not know about the state of the two doors and, after a sensing action, it detects that $Door1$ is closed. During the KB update, the deductive mechanism allows the robot to know that $Door2$ is open. This information must be sent to the lower level in case there is a component of the system that relies on it.

Finally, it is important to highlight that the abstraction process must be asynchronous with respect to the lower Data Interpretation module. This is important since updating a logic-based KB can require high computational times. In fact, it is not always the case that KB must be updated, since information about the environment are represented at a higher degree of abstraction. In this way the update-rate in the higher level is much less than the one in the lower level.

4.2.3 Reasoning on agent's knowledge

The Reasoning System is the decision-making module that implements deductive routines, such as plan generation and temporal projection. In particular, this module is responsible for generating plans for achieving desired goals and for verifying the feasibility of the current plan in the current state during plan execution.

The cognitive capability relies on a declarative description of the environment and the agent features, and on reasoning services offered by a knowledge representation system. High-level information are included in a knowledge base containing axioms and facts about the world and the reasoning process on the agent's knowledge is developed in order to generate plans able to lead the

agent in a state in which its goals are satisfied. In fact, plans can be expressed as high-level control programs, that specify the actions that must be executed by the Control System.

Observe that this module is not in charge of monitoring plan execution, that is of checking the correct execution of actions in the plan, by a direct feedback from the Control System. Instead this is done in the overall perception-action cycle, so that either goal achievement or plan failure is detected by analyzing sensor data and by updating the world model. In fact, the necessity of generating a new plan is determined by the verification that in the current high-level state the current plan do not guarantee the achievement of the goal.

Moreover, since deductive processes are usually time-consuming tasks, they must be asynchronous with respect to low-level action execution. And again the higher level of abstraction is such that only a few changes in the world will require a plan generation task.

For example, consider a mobile robot with the goal of entering a room. The plan computed by the reasoning systems suggests to follow a corridor until the corresponding door is reached, then to check whether the door is open and, in case it is open, to enter it. Now, if while following the corridor the robot encounters an unpredicted obstacle, the underlying reactive control system is in charge of avoiding it, and the reasoning system does not even have to know about this obstacle, since it is not represented in the high-level world model. However, if the door is detected to be closed, then the information is passed at the higher level, that must consider that the current plan is not feasible in the new situation for reaching the goal and possibly start a new plan generation procedure.

4.2.4 Control System

The Control System is in charge of actual execution of actions, by low-level control of the agent's devices. In particular it must provide the following capabilities:

- executing elementary actions, that are assumed to be atomic by the higher level but consists in a sequence of low-level commands to the agent's actuators;
- managing the combination of elementary actions (such as sequential and parallel execution), that is executing control programs.

The Control System can take advantage of both a control program specifying the actions to be performed, and a low-level description of the environment containing information about how to execute actions. The combination of these two kinds of information is also needed for relating objects mentioned in the control program with their representations in the world model.

We provide a further example from mobile robots domain, by considering the high level action "follow the corridor C_1 until reaching the door D_2". Since the low-level commands to the motors depend on data about the position of the robot in the corridor, we need first to map objects C_1 and D_2 into their representations in the low level world model. In this way the robot can be oriented towards the target position and, whenever it is too close to a wall, it can be moved towards the middle of the corridor.

4.3 A methodology for building cognitive agents

In this section we present a methodology for building cognitive agents which relies on the previously described architecture. The methodology is divided in two parts: the first is domain independent and consists in the realization of the architecture proposed above; the second is the definition of the application domain.

4.3.1 Implementing heterogeneous layered architectures

This design process is agent dependent, but not domain dependent. In other words, it aims to provide the agent with general capabilities, that are not related to the particular task to perform, but to the proper features of the agent and, possibly, to the kind of environment in which it is going to be embedded.

The heterogeneous architecture can be effectively implemented by making use of existing modules. For example a mobile robot project can take advantage of existing control and reasoning systems. The fundamental step is thus defining the interfaces among different modules. Therefore, we do not enter here into details of how to design a single module (two such modules are described in Chapters 6 and 7), but we analyze the requirements for allowing the modules to effectively interact together.

The interaction among the modules of the architecture is characterized by two kinds of interfaces: the first is between the deliberative and the operative levels, the second is between Knowledge Acquisition and Task Execution.

The interface between Knowledge Acquisition (KA) and Task Execution (TE) is represented by models of the world, that contain a description of both the knowledge that the agent has about the environment and its goals. The representation language is different in the two levels. At the cognitive level a logic framework defines a knowledge base of facts and axioms about the world, while at the operative level structured information about the environment are maintained. World models are built by different KA modules using different technologies. This requires an effort for maintaining the world models consistent and aligned, but provides a useful mechanism for integrating reactive and reasoning capabilities. Instead the TE modules make use of these data for deciding and performing actions, but they do not explicitly modify the world model. Any modification will result from executing actions and sensing their actual effects.

The interface between the deliberative and the operative levels is represented by correspondences between the representations of the information to be exchanged in the two levels. The interaction between the reasoning and the control system is characterized by the mapping between the high-level representation of plans and low-level control programs that can be executed by the controller.

A similar situation occurs in the interface between Data Interpretation and Modelling System. Indeed data structures containing processed sensor data should be defined according to the kind of information that the sensors are able to capture from the environment. Modelling is the process of describing in a high level language data coming from perceptual analysis, in other words it aims at relating structured data with a symbolic description of the information they represent.

Summarizing, the first part of the design methodology consists of two steps.

1. Designing the four principal modules of the layered architecture, by considering the agent's abilities and the kind of the environment (e.g. which kind of sensors and actuators are available, which sensor data interpretation routines are effective, which kind of control techniques are needed, etc.).

2. Designing the interfaces among these modules by taking into account all the considerations given above in this section.

4.3.2 Defining the application domain

The design process of the application domain is performed by the following steps.

1. Defining abstract actions and conditions. It is an important design element, that characterizes both the high-level representation of the dynamic system and the implementation of the

corresponding control routines. In general a trade-off must be considered. Indeed abstraction is required in order to make the reasoning process effective, but more abstract actions and conditions involve more complex and difficult control and data interpretation as well as less reasoning.

2. Defining control procedures that implement high-level elementary actions, and sensor data interpretation routines for verifying conditions on the world. A number of different solutions for implementing actions can be used (fuzzy behaviors, ad hoc procedures, etc.), so it is important to prove their effectiveness in the environment before continuing on. This step ends with a test phases for proving the effectiveness of the correspondence between high-level actions and control activities, and between conditions and sensor data analysis. It could be necessary to go back to the previous step in case it turns out to be difficult to realize effective and robust control procedures or reliable sensor data interpretation routines.

3. Providing a high-level description of the environment, based on actions and conditions specified above. Also this step can lead back to the previous ones, if the definition of actions and conditions is not sufficient (or not adequate) for describing some situations or events.

4. Describing agent's goals and generating the corresponding plans. This step can be performed either off-line for standard goals, or on-line as the answer of a query while the agent is working.

The first important feature of the methodology we have proposed is that it attacks the problem of integrating reasoning and reactivity in the middle, that is by defining first the interface between the two levels. This allows for individuating very soon possible design errors in connecting the deliberative and the operative levels. For example, the designer can verify very soon the reliability of a sensor interpretation routine for detecting a property on the world. If this is not adequate, the design must be revisited since it is of no use to write a high level description of the environment by means of a property that the agent cannot detect properly.

Second, the methodology is adequate for incremental design of agents, since the steps above can be iterated in order to refine the description of the environment and thus to augment the agent's capabilities. For example, after some basic features of the agent have been developed, the designer can consider to increase its capabilities by adding new properties to be known and new actions to be performed.

Third, most of the work realized to implement an application can be reused (possibly with minimal modifications) when the agent is moved to a new environment (with similar characteristics). In fact, once steps 1 and 2 are well-designed and realized, that means having implemented basic general actions that are executable in several environments, providing a description of the new environment and the agent's goals is a fast and easy process.

One of the main objectives in using a design methodology is the possibility of having a tool for identifying design errors as soon as possible, since their correction is much more costly when the project is at an advanced stage. Once a design error has been detected, all the previous steps must be revisited. In some cases it could be the case that also some component of the architecture must be redesigned.

We believe that the methodology we have proposed provides the designer with a systematic tool for effective realization of cognitive agents. The methodology has been used for implementing the cognitive level of a mobile robot that has been successfully tested in different environments and for different tasks. The effectiveness of the methodology is proved by the ease of adaptation in new environments, as described in Chapter 8.

Chapter 5

The Mobile Robot "Tino"

The methodology proposed in the previous chapter has been used in practice for the realization of an actual cognitive mobile robot. Starting from the basic mobile platform described in Chapter 3, we have designed and developed all the components needed for realizing our cognitive robot, that we have named "Tino".

The crucial point in the development of such a *cognitive robot* is the integration of the following capabilities: (i) achieving high-level goals by reasoning about its knowledge about the environment, (ii) promptly reacting to unpredicted situations and adjusting its behavior based on new knowledge acquired through its sensors during task executions.

The above requirements have been obtained by:

1. defining a new robot architecture able to integrate reasoning and reactivity by providing heterogeneous representations of the world;

2. defining a framework for reasoning on the incomplete knowledge of the robot and generating conditional plans;

3. implementing reliable sensing actions for knowledge acquisition.

Specifically in this chapter we concentrate on the definition of the robot's architecture, that is composed by both existing and novel modules. Therefore, in the next two chapters, we describe two novel modules we have developed: (i) the knowledge representation formalism and the reasoning procedures for generating conditional plans; (ii) the stereo vision system for acquiring spatial information through a stereo camera, that is the basis for the implementation of sensing actions. Finally in Chapter 8 we present some application domains for our cognitive mobile robot.

5.1 Architecture of the robot "Tino"

According to the requirements for the cognitive architecture we have presented in the previous Chapter, the Pioneer base described in Chapter 3 has been expanded in several directions:

1. the software architecture has been redesigned, according to the considerations about the hybrid layered architecture given in Chapter 4, in order to effectively integrate a high-level reasoning system within the Saphira fuzzy controller;

2. a complete cognitive level has been added on top of the Saphira controller, this new level is able to maintain a symbolic representation of the robot's knowledge and to generate Colbert programs for achieving its goals;

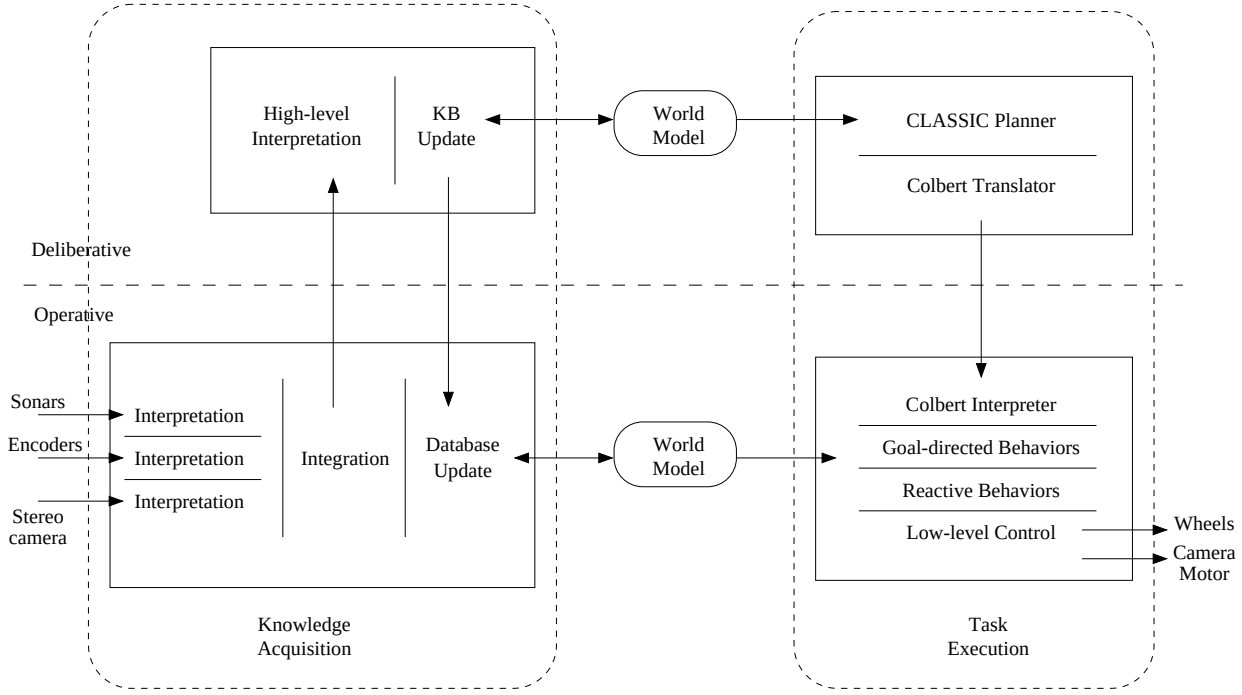


Figure 5.1: Layered architecture for the robot Tino

3. new interpretation routines taking advantage of the stereo camera have been developed;

4. new fuzzy behaviors (in particular sensing behaviors) have been realized for implementing high-level actions.

5.1.1 Software Architecture

The software architecture, shown in Fig. (5.1), of the robot Tino is designed by following the lines reported in Chapter 4. It presents a cognitive level, based on a logic representation of the robot's knowledge, and an operative level, based on a geometrical representation of the environment and a set of values denoting the internal state of the robot.

The main features of the architecture we propose for the implementation of reasoning and reactivity are *heterogeneity* and *asynchronism*.

The first property (*heterogeneity*) is important in order to make use of many different techniques, for example a fuzzy reactive controller, a knowledge representation system based on Description Logics, and low-level image processing routines. The use of a single representation system, as proposed in [44, 83, 75], for both the reasoning and the reactive system introduces a limitation in the technologies to be used, since a constraint on the representation formalism is imposed. With our architecture, we provide the designer with all the flexibility required by the complexity of the development process.

The second property (*asynchronism*) is relevant for an effective integration of reasoning and reactivity. Indeed, asynchronous architectures, unlike synchronous ones [67, 82], allow time-critical modules to have processing time available when needed. In our case this has been obtained by forcing the system to give control to low-level modules every 100 ms.

The knowledge of the robot about the environment is constituted by a description of the dynamic system, containing information about the state of both the robot and the environment. This knowledge is represented over two levels: a cognitive model, including a symbolic knowledge base,

and an operative model, constituted by a data base of objects properties.

With respect to the schema in Fig. 5.1, we made use of some components of the Saphira architecture: the sensor data interpretation routines, the artifacts updating process, the fuzzy controller, and the Colbert interpreter. The other components have been designed, implemented and integrated into the architecture in order to realize a full working robotic system and to make a number of tests in different environments.

In the following sections we describe the basic functionalities of the deliberative and the operative levels, while further details on specific modules (the reasoning system and the stereo vision system) are given in the next chapters.

5.2 Deliberative Level

The deliberative level is mainly concerned with an explicit representation of the robot's knowledge based on Description Logics. This knowledge is formed by both a general description of the environment provided by the robot's designer and the information acquired during task execution. The language specification and the reasoning services used in this level are described in chapter 6.

The deliberative level is formed by two components: a high-level knowledge acquisition module and a reasoning system.

The KA module is in charge of building and maintaining a knowledge base expressing a high level model of the world in which the robot is embedded. This process makes use of artifacts in the LPS, representing objects in the environment, and the results of sensing behaviors to update the robot's knowledge about the environment. In this way every high level atomic predicate can be monitored. For example, the property of being in a room is satisfied as long as the position of the robot is within the range of the artifact representing the room, while a door closed is detected by an interpretation routine driven by a sensing behavior.

The reasoning system processes the knowledge base containing information on the world in order to make decisions about actions to be performed for goals achievement. In particular, a conditional planner (described in the next chapter), based on the reasoning services provided by a well-known knowledge representation systems (CLASSIC), has been developed in order to generate conditional plans. The planner expresses these plans in the form of activities in the Colbert language. This is obtained by the implementation of a translator from the internal representation of plans in CLASSIC to Colbert activities. These activities representing the plans can thus be directly executed from the Colbert interpreter in Saphira controller. Moreover, during plan execution, the reasoning system check for the validity of the current plan in the current situation (provided by the KA module), and, if needed, it performs a new plan generation task.

5.3 Operative Level

The operative level is mainly based on the Saphira fuzzy controller [75, 53] described in Chapter 3. The operative world model is thus represented by the Local Perceptual Space.

The sensor data are processed by data interpretation routines and then, if needed, they are combined together in order to deal with a single world model. The control system can be divided into four modules: (i) the interpreter for executing Colbert activities, (ii) a set of goal-directed behaviors that are managed by the activities for actual execution of actions, (iii) a set of reactive behaviors, usually active at every time during task execution, that take their inputs from the low-level world model for promptly reacting to unexpected situations, (iv) low-level control of the hardware devices provided by the low-level routines of Saphira.

With respect to the basic components of the Saphira architecture [53], we added to the operative level the following features:

1. *sensing behaviors* that effectively implement sensing actions, by making use of perceptual data processing,

2. a stereo vision system able to extract dense spatial information about the environment,

3. the interfaces with the cognitive level.

We introduce in this setting the concept of *sensing behaviors*. Sensing behaviors are goal-directed behaviors able to verify the value of a property in the world. The characterizing feature of this kind of behaviors is the ability of controlling the robot in order to ensure the effective execution of a data interpretation routine. Observe that sensing behaviors can affect the position of the robot in the environment, but they should not affect the high-level state of the robot. For example, during the execution of a sensing action for detecting if a door is open, the corresponding sensing behavior can actually move the robot and put it in front of the door, without changing its high-level state, that is being in the corridor and close to that door.

We have implemented on our robot a number of sensing behaviors for detecting if a door is open, that make use of different techniques and different sensor devices (sonars and stereo camera). Each of them is in charge of moving the robot to a "good position" for the data interpretation routine. This position is different depending on the sensor device used, for instance when using sonars the robot should be exactly perpendicular to the door.

Let us remark that knowledge acquisition is a key point to actually execute the robot's tasks, because of its incomplete knowledge about the world. Sensing behaviors are the effective tools we use for acquiring new knowledge about the world.

Finally, a new stereo vision system (that is described in detail in Chapter 7) has been fully implemented and integrated in the robot's architecture. Many experiments have been done in order to verify the effectiveness of the system and we have obtained significant results showing a better performance with respect to other sensor devices, such as sonar sensors. In particular, we take advantage of efficiency and accuracy of this system not only for improving the robot's navigation capabilities, but also in order to develop reliable and robust data interpretation routines for implementing sensing actions. For instance, the implementation of the sensing behavior for detecting if a door is open based on stereo vision analysis came out to be much more reliable than the others based on sonar readings.

Chapter 6

The Reasoning System

In this chapter we describe a formalism for reasoning about actions, that has been recently proposed in [21, 22], and the implementation of a conditional planner based on this formalism. This setting constitutes the basis for the reasoning system of the cognitive robot "Tino".

The main objective is to develop reasoning tools for the agent, and in particular a plan generation procedure for selecting the actions that will lead to the goal. Observe however that, because of incomplete information on the environment, achieving a goal usually depends on the possibility of performing actions for knowledge acquisition. So the reasoning system must be able to take decisions about the execution of both actions involving changes in the dynamic system, and *sensing actions*, that are knowledge producing actions affecting only the mental state of the agent.

The formalism we present in this chapter is able to deal with sensing actions and to produce conditional plans [22], including both moving and sensing actions so that new knowledge can be discovered and exploited during task execution.

The basis of our proposal for reasoning about actions is provided by Propositional Dynamic Logics (PDLs), following the work of [74, 25]. However, the fundamental step towards the implementation has been to rely on the tight correspondence that exists between PDLs and Description Logics (DLs) [78, 23]. By exploiting this correspondence we have been able both to develop an interesting theoretical framework for reasoning about actions and to obtain an implementation that uses a knowledge representation system based on DLs.

Description Logics have been developed to support a rational design of knowledge representation languages and their associated reasoning services, centered around the notions of frame and hierarchical organization of knowledge, and constitute the basis of several implemented tools [86]. The key idea underlying our proposal is to exploit the features for knowledge representation and reasoning of DLs to obtain a principled implementation of reasoning about actions in a setting derived from PDLs. In particular, we rely on a DL enriched with an epistemic operator, originally introduced to formally characterize several practical features of knowledge representation systems based on DLs, not captured by a first-order setting [30]. Through the epistemic operator one can distinguish what is true in the world from what is known by the agent, thus allowing both for a new characterization of the dynamic system and for a weaker notion of inference.

The work on computational aspects of reasoning in DLs (see [32] for a survey) shows that the typical form of dynamic axioms leads to cyclic assertions in the knowledge base and thus requires sophisticated reasoning techniques. Hence, we have reinterpreted dynamic axioms by means of the so-called procedural rules. By relying on the epistemic interpretation of these rules given in [31], we have defined a framework which provides both an epistemic representation of dynamic axioms and a weak form of reasoning. In this way, we obtain a simplified reasoning task and a semantically justified approach to deductive planning. Furthermore, an extension of such a framework has been developed [22] in order to allow the robot to reason in presence of incomplete information and to

make use of sensing actions for knowledge acquisition.

Several studies propose DLs for the development of planning systems (among them [28, 5, 49]). In these proposals the language of DLs is extended with specific constructs that allow actions to be represented as concepts. The planning system can thus reason about plans, by exploiting subsumption in DLs.

Our proposal takes a different perspective from other DL planners, derived from the correspondence with PDLs, where actions are represented as roles, and properties of states are represented as concepts. In our case, plans are generated through a combination of the propagation mechanism for the procedural rules and taxonomic reasoning for checking the static properties of states.

One of the motivations underlying our proposal for reasoning about actions is the possibility of relying on a knowledge representation system based on DLs for the implementation. In particular, we have chosen CLASSIC [12], a well-known system based on DLs, to take advantage of an efficient and reliable reasoning system.

We present in the next sections the Description Logic formalism and its epistemic extension, the general framework for the representation of dynamic systems, our specific proposal for representing and reasoning about actions, and the implementation of a conditional planner in CLASSIC.

6.1 Autoepistemic Description Logics

The technical background of our proposal is constituted by Description Logics (DLs) enriched with an epistemic operator. We refer to [32] for a deeper introduction to DLs.

DLs have been developed to represent the domain of interest in terms of *concepts* (unary predicates) that characterize subsets of the objects, called *individuals*, in the domain, and *roles* (binary predicates) over such domain. Concepts are described by means of concept expressions that can be constructed through set operators plus special constructors involving roles that link the individuals belonging to a concept to those of other concepts. One can establish a hierarchy of concepts, based on set containment (called *subsumption*).

We focus on a well-known DL, $\mathcal{ALC}$, [79] and its autoepistemic extension, $\mathcal{ALCK}_{\mathcal{NF}}$ [33], obtained by adding two nonmonotonic modal operators: a *minimal knowledge operator* $\mathbf{K}$ and a *default assumption operator* $\mathbf{A}$, that are interpreted according to the nonmonotonic modal logic *MKNF* [60]. Such an extension plays a critical role in our formalization of dynamic systems.

The abstract syntax of $\mathcal{ALC}$ is defined as follows:

$$C, D \quad ::= \quad A \mid \bot \mid \top \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \exists R.C \mid \forall R.C$$

where A denotes an atomic concept, C and D denote generic concepts, R denotes an atomic role.

We shall give the semantics of $\mathcal{ALC}$ using the so called Common Domain Assumption (CDA): every interpretation is defined over the same, fixed, countable-infinite domain of individuals Δ. CDA is not essential for $\mathcal{ALC}$. It can be easily shown that CDA does not affect the reasoning tasks considered in non-epistemic DLs. However, CDA is required for $\mathcal{ALCK}_{\mathcal{NF}}$.

An *$\mathcal{ALC}$-interpretation* $\mathcal{I}$ as a function mapping each concept expression into a subset of Δ and each role expression into a subset of $\Delta \times \Delta$, such that:

$$\begin{aligned} A^{\mathcal{I}} &\subseteq \Delta \\ R^{\mathcal{I}} &\subseteq \Delta \times \Delta \\ \top^{\mathcal{I}} &= \Delta \end{aligned}$$

$$\begin{aligned}
\bot^{\mathcal{I}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta \setminus C^{\mathcal{I}} \\
\forall R.C^{\mathcal{I}} &= \{d_1 \in \Delta \mid \forall d_2.\, (d_1, d_2) \in R^{\mathcal{I}} \Rightarrow d_2 \in C^{\mathcal{I}}\} \\
\exists R.C^{\mathcal{I}} &= \{d_1 \in \Delta \mid \exists d_2.\, (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}
\end{aligned}$$

DLs are typically used for representing the knowledge about a problem domain by providing mechanisms for specifying relationships among concepts, and for providing information about specific individuals. Accordingly, an $\mathcal{ALC}$ *knowledge base* Σ is defined as a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, called the *TBox*, is a finite set of inclusion assertions of the form $C \sqsubseteq D$, with $C, D \in \mathcal{ALC}$, and $\mathcal{A}$, called the *ABox*, is a finite set of membership assertions of the form $C(a)$ or $R(a, b)$, where $C, R \in \mathcal{ALC}$ and a, b are *names* of individuals. We assume that different names denote different individuals, and with a slight abuse of notation, we do not distinguish between individuals and their names.

The semantics of inclusion assertions is defined in terms of set inclusion: $C \sqsubseteq D$ is satisfied in $\mathcal{I}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Membership assertions are interpreted in terms of set membership: $C(a)$ is satisfied in $\mathcal{I}$ iff $a \in C^{\mathcal{I}}$ and $R(a, b)$ is satisfied in $\mathcal{I}$ iff $(a, b) \in R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies an $\mathcal{ALC}$-knowledge base Σ iff every inclusion and membership assertion of Σ is satisfied in $\mathcal{I}$. A knowledge base Σ *logically implies* an inclusion or membership assertion σ, written $\Sigma \models \sigma$, iff every interpretation $\mathcal{I}$ satisfying Σ also satisfies σ.

The autoepistemic description logic $\mathcal{ALCK}_{\mathcal{NF}}$ is an extension of $\mathcal{ALC}$ added with an epistemic operator $\mathbf{K}$ interpreted as (minimal) knowledge and a default assumption operator $\mathbf{A}$. More precisely the $\mathcal{ALCK}_{\mathcal{NF}}$ abstract syntax is as follows:

$$C, D \quad ::= \quad A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall Q.C \mid \exists Q.C \mid \mathbf{K}C \mid \mathbf{A}C$$

$$Q \quad ::= \quad R \mid \mathbf{K}R \mid \mathbf{A}R$$

where A denotes an atomic concept, C and D denote generic concepts, R denotes an atomic role, and Q a generic role.

Non-epistemic concepts and roles are given the standard semantics of DLs, while epistemic concepts and roles are interpreted on the class of Kripke structures where worlds are $\mathcal{ALC}$-interpretations, and all worlds are connected to each other, i.e. the accessibility relation among $\mathcal{ALC}$-interpretations is universal, thus Kripke structures correspond to sets of $\mathcal{ALC}$ interpretations.

An $\mathcal{ALCK}_{\mathcal{NF}}$*-interpretation* is defined as a triple $(\mathcal{I}, \mathcal{M}, \mathcal{N})$, where $\mathcal{M}$ and $\mathcal{N}$ are sets of $\mathcal{ALC}$-interpretations over the domain Δ, and $\mathcal{I}$ is a distinguished interpretation belonging to $\mathcal{M}$ and $\mathcal{N}$, such that:

$$\begin{aligned}
(A)^{\mathcal{I},\mathcal{M},\mathcal{N}} &\subseteq \Delta \\
(R)^{\mathcal{I},\mathcal{M},\mathcal{N}} &\subseteq \Delta \times \Delta \\
(\top)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \Delta \\
(\bot)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= (C)^{\mathcal{I},\mathcal{M},\mathcal{N}} \cap (D)^{\mathcal{I},\mathcal{M},\mathcal{N}} \\
(C \sqcup D)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= (C)^{\mathcal{I},\mathcal{M},\mathcal{N}} \cup (D)^{\mathcal{I},\mathcal{M},\mathcal{N}}
\end{aligned}$$

$$
\begin{aligned}
(\neg C)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \Delta \setminus (C)^{\mathcal{I},\mathcal{M},\mathcal{N}} \\
(\forall Q.C)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \{d_1 \in \Delta \mid \forall d_2.\, (d_1,d_2) \in (Q)^{\mathcal{I},\mathcal{M},\mathcal{N}} \Rightarrow d_2 \in (C)^{\mathcal{I},\mathcal{M},\mathcal{N}}\} \\
(\exists Q.C)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \{d_1 \in \Delta \mid \exists d_2.\, (d_1,d_2) \in (Q)^{\mathcal{I},\mathcal{M},\mathcal{N}} \wedge d_2 \in (C)^{\mathcal{I},\mathcal{M},\mathcal{N}}\} \\
(\mathbf{K}C)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \bigcap_{\mathcal{J}\in\mathcal{M}} ((C)^{\mathcal{J},\mathcal{M},\mathcal{N}}) \\
(\mathbf{A}C)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \bigcap_{\mathcal{J}\in\mathcal{N}} ((C)^{\mathcal{J},\mathcal{M},\mathcal{N}}) \\
(\mathbf{K}R)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \bigcap_{\mathcal{J}\in\mathcal{M}} ((R)^{\mathcal{J},\mathcal{M},\mathcal{N}}) \\
(\mathbf{A}R)^{\mathcal{I},\mathcal{M},\mathcal{N}} &= \bigcap_{\mathcal{J}\in\mathcal{N}} ((R)^{\mathcal{J},\mathcal{M},\mathcal{N}})
\end{aligned}
$$

Intuitively, an individual $d \in \Delta$ is an instance of a concept C iff $d \in (C)^{\mathcal{I},\mathcal{M},\mathcal{N}}$ in the particular $\mathcal{ALC}$-interpretation $\mathcal{I} \in \mathcal{W}$. An individual $d \in \Delta$ is an instance of a concept $\mathbf{K}C$ (i.e. $d \in (\mathbf{K}C)^{\mathcal{I},\mathcal{M},\mathcal{N}}$) iff $d \in C^{\mathcal{J},\mathcal{M},\mathcal{N}}$ for all possible $\mathcal{ALC}$-interpretations $\mathcal{J} \in \mathcal{M}$. The individual $d \in (\mathbf{K}C)^{\mathcal{I},\mathcal{M},\mathcal{N}}$ is said to be a *known instance* of a concept C; also, C is said to be *known* for the individual d. Similarly, an individual $d \in \Delta$ is an instance of a concept $\exists\mathbf{K}R.\top$ iff there is an individual $d' \in \Delta$ such that $(d,d') \in R^{\mathcal{J},\mathcal{M},\mathcal{N}}$ for all possible $\mathcal{J} \in \mathcal{M}$. The individual d' is said to be a *known* R-successor of d. An individual $d \in \Delta$ is an instance of the concept $\neg\mathbf{A}C$ (i.e. $d \in (\neg\mathbf{A}C)^{\mathcal{I},\mathcal{M},\mathcal{N}}$) iff $d \in \neg C^{\mathcal{J},\mathcal{M},\mathcal{N}}$ for at least one possible $\mathcal{ALC}$-interpretation $\mathcal{J} \in \mathcal{N}$. In other words it is not consistent to assume d belonging to C if there exists one possible world in $\mathcal{N}$ in which d belongs to $\neg C$.

An $\mathcal{ALCK}_{\mathcal{NF}}$ knowledge base Σ is defined as a pair $\Sigma = \langle \mathcal{T}, \mathcal{A}\rangle$, where the TBox $\mathcal{T}$ is a finite set of inclusion assertions of the form $C \sqsubseteq D$, with $C, D \in \mathcal{ALCK}_{\mathcal{NF}}$, and the ABox $\mathcal{A}$ is a finite set of membership assertions of the form $C(a)$ or $R(a,b)$, with $C, R \in \mathcal{ALCK}_{\mathcal{NF}}$ and a, b names of individuals.

An inclusion assertion $C \sqsubseteq D$ is satisfied by a structure $(\mathcal{M},\mathcal{N})$ iff every interpretation $\mathcal{I} \in \mathcal{M}$ is such that $(C)^{\mathcal{I},\mathcal{M},\mathcal{N}} \subseteq (D)^{\mathcal{I},\mathcal{M},\mathcal{N}}$. The membership asertion $C(a)$ is satisfied by $(\mathcal{M},\mathcal{N})$ iff for every interpretation $\mathcal{I} \in \mathcal{M}$ $a \in (C)^{\mathcal{I},\mathcal{M},\mathcal{N}}$ and $R(a,b)$ is satisfied by $(\mathcal{M},\mathcal{N})$ iff for every interpretation $\mathcal{I} \in \mathcal{M}$ $(a,b) \in (R)^{\mathcal{I},\mathcal{M},\mathcal{N}}$. To express that a structure $(\mathcal{M},\mathcal{N})$ satisfies an assertion ψ we use the notation $(\mathcal{M},\mathcal{N}) \models \psi$.

A *model* for an $\mathcal{ALCK}_{\mathcal{NF}}$-knowledge base Σ is a structure $(\mathcal{M},\mathcal{N})$ of $\mathcal{ALC}$-interpretations that satisfies every inclusion and membership assertion of Σ.

Next, a preference semantics on universal Kripke structures is defined, which allows one to select only those models where the knowledge is minimal. This is achieved by maximizing in each epistemic model the number of possible worlds (i.e. $\mathcal{ALC}$-interpretations), which can also be explained as maximizing ignorance.

A *preferred model* $(\mathcal{M},\mathcal{N})$ for Σ is a model for Σ such that $\mathcal{M}$ is a maximal set of $\mathcal{ALC}$-interpretations, in the sense that for each set $\mathcal{M}'$, if $\mathcal{M} \subset \mathcal{M}'$ then $(\mathcal{M}',\mathcal{N})$ is not a model for Σ. Σ is *satisfiable* if there exists a preferred model for Σ, *unsatisfiable* otherwise. Σ *logically implies* an (inclusion or membership) assertion ψ, written $\Sigma \models \psi$, iff ψ is satisfied in every preferred model for Σ.

By using the epistemic operator, it is possible to formalize several practical features provided by implemented knowledge representation systems based on DLs [31]. In particular, here we recall the so-called *procedural rules* (or simply rules).

Procedural rules take the form:

$$C \mapsto D$$

(where C, D are $\mathcal{ALC}$ concepts). Roughly speaking, their meaning is "if an individual is proved to be an instance of C, then conclude that it is also an instance of D". They can be viewed as a weak form of implication for which the contrapositive does not hold. A procedural rule $C \mapsto D$ can be formalized in $\mathcal{ALCK}_{\mathcal{NF}}$ by the epistemic assertion

$$\mathbf{K}C \sqsubseteq D$$

A consistent knowledge base in which the epistemic operator occurs only in rules of the above form has a unique preferred model; moreover, in such a case reasoning can be accomplished by constructing a knowledge base, called *first-order extension* [30, 31].

The reasoning services provided by a DL system include two special forms of logical implication: *subsumption*, expressed by $\Sigma \models C \sqsubseteq D$, and *instance checking*, expressed by $\Sigma \models C(a)$. Observe that, from a computational point of view, these two forms of logical implication are equivalent.

Regarding the computational aspects of reasoning in DLs, there is a trade-off between the set of concept forming constructs allowed in a DL and the complexity of reasoning. More specifically, two aspects must be carefully considered: the set of constructs allowed in the expressions and the form of the inclusion assertions.

In particular, it has been shown [79] that logical implication in $\mathcal{ALC}$ is an EXPTIME-complete problem. The high complexity of logical implication in $\mathcal{ALC}$ is due to the general form of inclusion assertions allowed. When considering general inclusion assertions of the form $C \sqsubseteq D$, reasoning becomes EXPTIME-hard even for a simple language as $\mathcal{FL}_0$, [63], subset of $\mathcal{ALC}$ which contains only intersection $\sqcap$ and universal quantification $\forall$. Hence, restrictions on the form of inclusion assertions are normally considered. In particular, a set of inclusion assertions is cyclic, when a chain of individuals connected through roles can be built that connects a concept with itself. Cycles are especially problematic from the computational point of view [66, 15, 18], and typically they are not allowed by implemented systems.

However, in the case of $\mathcal{ALC}$ knowledge bases with acyclic inclusion assertions, subsumption and instance checking are PSPACE-complete [32], and we get the same complexity characterization even if we consider an empty knowledge base. Moreover, instance checking remains PSPACE-complete, even when $\mathcal{ALCK}_{\mathcal{NF}}$ is used to express the query concept C and procedural rules are allowed in the knowledge base [30].

6.2 Formalization of Dynamic Systems

Dynamic systems are typically modeled in terms of state transitions caused by actions. A *state* represents a possible state of affairs of the system. Each state is associated with a set of properties which hold in the represented state of affairs. *Actions* cause state transitions, making the system evolve from the current state to the next one. The behavior of the system can thus be represented with a *transition graph*, where each node denotes a state and is labeled with its associated properties, and each arc is associated to an action denoting a state transition.

When using a logical formalism for representing dynamic systems, the properties of actions and states are expressed in terms of *axioms* of some logic (e.g. Situation Calculus or PDLs). These axioms select a subset of all possible transition graphs and the actual behavior of the agent is determined by reasoning on the properties that are true in all the selected graphs, that is those properties that are logically implied by the axioms.

PDLs		DLs	
atomic proposition	A	atomic concept	A
true	`tt`	top	$\top$
false	`ff`	bottom	$\bot$
conjunction	$C \wedge D$	conjunction	$C \sqcap D$
disjunction	$C \vee D$	disjunction	$C \sqcup D$
negation	$\neg C$	negation	$\neg C$
diamond ("some executions...")	$\langle r \rangle C$	existential quantification	$\exists R.C$
box ("all executions...")	$[r]C$	universal quantification	$\forall R.C$
valid implication (axiom)	$C \Rightarrow D$	inclusion assertion	$C \sqsubseteq D$
		instance assertion	$C(a) \mid R(a_1, a_2)$

Figure 6.1: Correspondence between DLs and PDLs.

Following the approach in [74], the behavior of the dynamic system can be specified by means of three kinds of axioms: "static axioms", action "precondition axioms", and "effect axioms".

- *Static axioms* specify properties which are true in every state and do not depend on actions. In other words, static axioms are used for representing background knowledge that is invariant with respect to the execution of actions.

- *Precondition axioms* specify circumstances under which it is possible to execute an action. We assume that the designer of the system is able to specify *sufficient conditions* for actions to be executed.

- *Effect axioms* specify (direct) *effects* of an action if executed under given circumstances, i.e. if executed in a state satisfying certain *premises*. Obviously, through static axioms additional effects can be inferred from those specified by effect axioms. Observe that in general we do not require the designer to specify all the effects of an action.

The last two kinds of axioms are referred to as *dynamic axioms*, since both are used to model dynamic aspects of the system.

In addition to these kinds of axioms, we assume that the designer specifies the knowledge on the initial state, by providing an *initial state description* in terms of the properties (not involving actions) that are associated with the initial state.

Actions are assumed to be *deterministic*, which means that in each transition graph, given a state and an action, at most a single successor state is determined. However, the properties associated with such successor state are generally different in different transition graphs. We might say that actions are deterministic but their effects are *underdetermined* in general.

The formalization in DLs is strictly related to those based on Propositional Dynamic Logics (PDLs) [74, 25]. PDLs [39] are modal logics for reasoning about computer programs that have been studied extensively (see [54] for a survey). It has been shown that there is a tight correspondence between DLs and PDLs [78, 23], which allows for considering PDLs and DLs as syntactic variants of each other. Fig. 6.1 shows how the constructs of PDLs can be translated into constructs of DLs and vice versa. In view of this correspondence, the DL-based formalization considered above can be seen as a variant of the one proposed in [74]. The only difference is that in [74] precondition axioms were not considered, forcing each action R to be executable in every state.

In DLs *states* are represented by *individuals*, i.e. elements of the domain of interpretation. *Properties of states* are represented by *concepts*. *Actions*, which are assumed to be deterministic, are represented as *functional roles*, i.e. roles interpreted as functions instead of relations. In fact, we

distinguish two kinds of roles: *static-roles*, which represent the usual notion of role in DLs and can be used for structuring properties of states, and *action-roles*, which are functional roles that denote actions. When R is an action-role, the construct $\exists R.\top$ expresses that there exists an execution of the action R, i.e. R is possible in the current state. Instead, the construct $\forall R.C$ expresses that all executions of R lead to a state where C holds, i.e., since R is deterministic, if the action R is executable then it leads to a state where C holds.

The behavior of the dynamic system is specified in terms of ($\mathcal{ALC}$) inclusion assertions as follows:

- *Static axioms*

  $$C \sqsubseteq D$$

  where C and D are concepts not involving action-roles. The assertion expresses that every state satisfying the property denoted by C satisfies also the property denoted by D.

- *Precondition axioms*

  $$C \sqsubseteq \exists R.\top$$

  where C is a concept representing a property of the current state that is sufficient in order to execute the action R. C does not involve action-roles. Multiple axioms per action are allowed.

- *Effect axioms*

  $$C \sqsubseteq \forall R.D$$

  where D is a concept representing a property that holds in the state resulting from executing R in a state satisfying the property C. C and D do not involve action-roles. Multiple axioms per action are allowed.

We denote with Γ the set of axioms in the specification. The specification apart from Γ includes the *initial state description*, which is expressed by a concept S (not a set of assertions).

Observe that a major difference between the formalization of dynamic systems in the Situation Calculus and the one in PDLs and DLs, is that in the former there is an explicit denotation of states through situation-terms, while in the latter states are implicitly described through their properties. In this respect, the DL-based formalization is analogous to a modal formalization of dynamic systems.

The framework presented here is quite standard both in Artificial Intelligence and in Computer Science. However, it does not deal directly with well known formalization problems studied in Artificial Intelligence (see e.g. [76]). In particular, the *qualification problem* – i.e. the problem of finding out the actual conditions that must be true to execute an action – does not arise since the designer is required to specify sufficient conditions for the execution of an action. While the *frame problem* – i.e. the problem of specifying what remains unchanged when an action is executed – is dealt with by requiring the designer to specify explicitly what remains unchanged by using frame axioms, that is effect axioms of the form $C \sqsubseteq \forall R.C$. Note that the designer is allowed to specify effects of actions partially. Furthermore, static axioms combined with effect axioms can be used to specify *ramifications*, i.e. indirect effects.

The kind of reasoning we are interested in can be phrased in the DL-based formalization as a logical implication of the form

$$\Gamma \models S \sqsubseteq D \tag{6.1}$$

which expresses that, given a specification of the behavior of the system Γ, and an initial state described by S, it follows that the initial state also satisfies the property denoted by D, possibly involving the execution of actions.

Specifically, in deductive planning one is interested in answering the following question: "Is there a sequence of actions that, starting from an initial state, leads to a state where a given property (the goal) holds?". Under the assumption of deterministic actions, this is captured by the following logical implication:

$$\Gamma \models S \sqsubseteq PLAN\_FOR\_G \tag{6.2}$$

where $PLAN\_FOR\_G$ stands for any concept of the form $\exists R_1.\exists R_2.\ldots\exists R_n.G$ (with $n \geq 0$ and R_i any action), and expresses the existence of a finite sequence of actions leading to a state where the goal G is satisfied. Formally, $PLAN\_FOR\_G$ is any concept belonging to the set $\mathcal{P}_\mathcal{S}$ that is inductively defined by: (i) $G \in \mathcal{P}_\mathcal{S}$; (ii) if $C \in \mathcal{P}_\mathcal{S}$, then $\exists R.C \in \mathcal{P}_\mathcal{S}$, for every action role R.[1]

From a *constructive proof* of the above logical implication one can extract an actual sequence of actions (a plan) that leads to the goal.

Observe that in this setting one may have very sparse knowledge about the system – say a few laws (axioms) one knows the system obeys – and yet be able to make several non-trivial inferences. Unfortunately, this generality leads to a high computational cost, as shown by the following proposition, where by propositional concept we refer to those concept expressions that are formed excluding quantifications on roles.

Proposition. *Consider a specification Γ of a dynamic system having the following characteristics:*

- *Static axioms of the form $C \sqsubseteq D$, with C and D propositional.*
- *Precondition axioms of the form $C \sqsubseteq \exists R.\top$, with C propositional.*
- *Effect axioms of the form $C \sqsubseteq \forall R.D$, with C,D propositional.*
- *Initial state description constituted by a concept S which is propositional.*

Deciding logical implications $\Gamma \models S \sqsubseteq D$, with D propositional, is an EXPTIME-hard problem.

The proposition is an easy consequence of a complexity result in [18]: reasoning with primitive inclusion assertions, i.e. inclusion assertions of the form $A \sqsubseteq C$ with A an atomic concept, in the DL $\mathcal{ALU}$, obtained from $\mathcal{ALC}$ by restricting existential qualification to the form $\exists R.\top$, is EXPTIME-hard. In fact, the same complexity bound holds if we restrict the class of specifications to those having precondition axioms of the form $\top \sqsubseteq \exists R.\top$, as in [74].

Note that in the proposition above static axioms are trivially acyclic, since they are propositional. Instead, the cyclicity of dynamic axioms is unavoidable, otherwise we could not relate the truth-value of a property in state resulting from executing an action to that in the current state. For example we could not express that a property C persists over an action R, i.e. $C \sqsubseteq \forall R.C$.

6.3 Reasoning about actions in autoepistemic DLs

In the framework described in the previous section, the dynamics of the system is specified in terms of *what is true in the world*. Now, we modify it, obtaining a new framework in which the dynamics

[1] We use $\exists R.C$ as an abbreviation for $\exists R_i.\top \sqcap \forall R_i.C$. Indeed, since actions are assumed to be deterministic, the two concept expressions are equivalent.

is specified in terms of *what the robot knows of the world*. The idea is that the robot achieves its conclusions based on its *epistemic state* and not on the actual state of the world.

As we shall see, this change of viewpoint in the representation has two important consequences:

- It simplifies reasoning, and in particular it simplifies the task of deducing plans.
- It makes deductive planning always constructive, i.e. the reachability of a state in which the goal holds is deduced only if there exists a known sequence of actions that leads to it.

Below we introduce the new representation framework and, subsequently, address reasoning.

6.3.1 Representing actions

The behavior of the agent is again described by means of both static axioms and dynamic axioms, divided into precondition axioms and effect axioms. Static axioms are inclusion assertions not involving action-roles, as before, and not involving cycles, in order to avoid the difficulties of reasoning about them. Dynamic axioms have a different form that makes use of the epistemic operator $\mathbf{K}$ (in the following C, D denote $\mathcal{ALC}$ concepts):

- *Precondition axioms* have the form

$$\mathbf{K}C \sqsubseteq \exists \mathbf{K}R.\top \tag{6.3}$$

  which is interpreted as: if a state (individual) x is an instance of C in all possible interpretations, then there exists a state y, the same in all possible interpretations, which is the (unique) $\mathbf{K}R$-successor of x.

- *Effect axioms* have the following form

$$\mathbf{K}C \sqsubseteq \forall \mathbf{K}R.\mathbf{K}D \tag{6.4}$$

  that can be read as: if a state (individual) x is an instance of C in all possible interpretations, then in every interpretation each $\mathbf{K}R$-successor of x (in fact at most one, being actions deterministic) is a known instance of D.

A special form of effect axioms considered above corresponds to *frame axioms*, i.e. axioms expressing which properties do not change when certain actions are executed. Indeed by writing

$$\mathbf{K}C \sqsubseteq \forall \mathbf{K}R.\mathbf{K}C$$

we express the fact that the property C is not affected by the execution of action R.

Also the *initial state description* is given in a different way. We denote by the individual name *init* the initial state, and give the initial state description in terms of a membership assertion on *init*. In other words, the initial state description has the form

$$S(init)$$

where S is a concept that expresses the known properties of the initial state, and *init* is an individual name that denotes the initial state.

The above formalization differs from the one given in the previous section in two major aspects. First, we have introduced a mechanism to denote states independently of the interpretation of the concepts, namely chains of any length of roles of the form $\mathbf{K}R$, with R an action-role, starting from

init. In fact, *init* denotes an individual which in every interpretation represents the initial state. Similarly, the $\mathbf{K}R_1$-successor of *init* is an individual $x_{init;R_1}$ which represents in every interpretation the $\mathbf{K}R_1$-successor of the state represented by *init*. The $\mathbf{K}R_2$-successor of $x_{init;R_1}$ is an individual $x_{init;R_1;R_2}$ which represents in every interpretation the $\mathbf{K}R_2$-successor of the state represented by $x_{init;R_1}$. And so on for chains of any length of roles of the form $\mathbf{K}R$, starting from *init*. As observed, the above described mechanism is not available in non-epistemic DLs as well as in PDLs, although it is provided by the Situation Calculus through situation-terms.

Secondly, the possibility of executing an action, and the effects that are obtained by executing that action, are given in terms of *what is known* about the current state. In fact, the presence of $\mathbf{K}C$, instead of C, in the left-hand side of both precondition axioms and effect axioms, requires C to be known for the individual x representing the current state, or more precisely requires C to be true for the state denoted by the individual x in all interpretations, i.e. C is required to be *valid* for the individual x. As mentioned in Section 2, this use of $\mathbf{K}$ is tightly connected with the realization of procedural rules in epistemic DLs. Observe that, for every property C, either C is valid for an individual or not, i.e. either C is known or it is not known in the current state. Namely, the robot has *complete* knowledge on its epistemic state. Such epistemic state in turn expresses the *partial* knowledge the robot has on the actual current state.

Planning problem. Let Γ be the set of static axioms, precondition axioms, and effect axioms, specifying the behaviour of the dynamic system. Given an initial state *init* satisfying certain properties S, a plan exists for a specified goal iff there exists a finite sequence of actions that, from the initial state *init*, leads to a state satisfying the goal. This condition is expressed by a logical implication similar to (6.2), namely:

$$\langle \Gamma, \{S(init)\}\rangle \models PLAN\_FOR\_G(init) \tag{6.5}$$

where $PLAN\_FOR\_G$ stands for any concept expression of the form $\exists\mathbf{K}R_1.\exists\mathbf{K}R_2.\ldots.\exists\mathbf{K}R_n.\mathbf{K}G$ in which $n \geq 0$ and each R_i is an action-role, and it expresses the fact that from the initial state *init* there exists a sequence of successors (the same in every interpretation) that terminates in a state (the same in every interpretation) where G holds (in every interpretation). Formally, $PLAN\_FOR\_G$ is any concept belonging to the set $\mathcal{P}_\mathcal{S}$ that is inductively defined by: (i) $\mathbf{K}G \in \mathcal{P}_\mathcal{S}$; (ii) if $C \in \mathcal{P}_\mathcal{S}$, then $\exists\mathbf{K}R.C \in \mathcal{P}_\mathcal{S}$, for every action role R.

Condition (6.5) holds iff for each preferred model $\mathcal{W}$ for $\Sigma = \langle \Gamma, \{S(init)\}\rangle$, there exists a state $x \in \Delta$ such that $x \in G^{\mathcal{I},\mathcal{W}}$ for all $\mathcal{I} \in \mathcal{W}$. Indeed, due to the special form of the dynamic axioms, such a state exists iff it is linked to the initial state by a chain of $\mathbf{K}R_i$, i.e. if there exists a sequence of successors (the same in every possible interpretation) that terminates in x.

We next show, through a simple example, that the use of epistemic inclusion assertions in the formalization of the dynamic system actually weakens the deductive capabilities of the robot. Let us first focus on the non-epistemic framework. Let Γ be the following specification of a dynamic system behavior:

Precond. axioms:	*Effect axioms:*
$C_1 \sqcap C_2 \sqsubseteq \exists R_1.\top$	$C_1 \sqsubseteq \forall R_1.D$
$C_1 \sqcap \neg C_2 \sqsubseteq \exists R_2.\top$	$C_1 \sqsubseteq \forall R_2.D$

and the initial state description be C_1. Suppose we want to know whether exists a plan for achieving D, i.e.:

$$\Gamma \models C_1 \sqsubseteq PLAN\_FOR\_D$$

It is easy to see that the answer to this planning problem is yes, since $\Gamma \models C_1 \sqsubseteq \exists R_1.D \sqcup \exists R_2.D$.

Now let us consider the epistemic framework. The specification of the dynamic system behavior becomes Γ':

Precond. axioms:	*Effect axioms:*
$\mathbf{K}(C_1 \sqcap C_2) \sqsubseteq \exists \mathbf{K} R_1.\top$	$\mathbf{K}C_1 \sqsubseteq \forall \mathbf{K} R_1.\mathbf{K}D$
$\mathbf{K}(C_1 \sqcap \neg C_2) \sqsubseteq \exists \mathbf{K} R_2.\top$	$\mathbf{K}C_1 \sqsubseteq \forall \mathbf{K} R_2.\mathbf{K}D$

and the initial state description: $C_1(init)$. The planning problem becomes:

$$\langle \Gamma', \{C_1(init)\}\rangle \models PLAN\_FOR\_D(init)$$

The answer to this planning problem is no, since neither the robot knows that R_1 is executable, nor it knows that R_2 is executable, hence it has no plan to reach D. The difference is that in the actual state of the world either C_2 is true or $\neg C_2$ is true. However neither C_2 nor $\neg C_2$ is known to be true by the robot (neither of the two is valid), hence the robot concludes that it cannot perform any action.

In fact, there is no *known* sequence of actions leading to the state satisfying the goal, yet in the non-epistemic framework the answer to the planning problem is yes, whereas it is no in the epistemic framework.

The example highlights the differences between the epistemic setting and the non-epistemic one. On the one hand, there are cases in which in the non-epistemic setting the answer to a planning problem is yes, i.e. the logical implication (6.2) holds, while in the epistemic setting the answer is no, i.e. the logical implication (6.5) does not hold. Conversely, it is easy to see that, if in the epistemic setting the logical implication (6.5) holds, then the logical implication (6.2) holds as well in the non-epistemic setting. That is, reasoning in the epistemic framework is a sound and well characterized approximation of the non-epistemic case. On the other hand, in the epistemic setting only constructive proofs are taken into consideration, in the sense that the logical implication (6.5) holds only if there exists a *known* sequence of actions leading to the state satisfying the goal. That is, if (6.5) holds then we are always able to extract an actual plan. As we have seen above this is not always the case in the non-epistemic setting. Hence we can conclude that the use of the epistemic operator in the formalization of actions allows for a *principled weakening* of the deductive capabilities of the robot.

6.3.2 Representing sensing actions

We now extend our framework in order to deal with sensing actions. Sensing actions are special actions that change the knowledge of the robot without changing the state of the external world in which the robot is embedded. Such an assumption may seem restrictive for sensing actions that can produce changes in the environment (i.e. sensing through the sonars may require motion). This is in fact not the case as we shall see later on, while it is useful to characterize sensing actions simply as knowledge producing actions.

We assume that the robot can sense certain facts represented by special atomic propositions, called *sensed propositions*; each sensed proposition is associated with a specific sensing action.

Action precondition axioms in Γ_P for the action R_S which senses the proposition S have the form:

$$\mathbf{K}C \sqcap \neg\mathbf{A}S \sqcap \neg\mathbf{A}\neg S \sqsubseteq \exists \mathbf{K} R_S.\top \tag{6.6}$$

where C is an $\mathcal{ALC}$ concept. It can be read as: if C holds in the current state s and the truth value of S is not known (i.e. it is consistent to assume both that S holds in s in every interpretation and that $\neg S$ holds in s in every interpretation), then it is possible to perform R_S, in the sense that there exists a unique R_S-successor s' of s which is the same in every interpretation.

Each sensing action R_S for the sensing proposition S has a *unique* **effect axiom** in Γ_E:

$$\mathbf{K}\top \sqsubseteq \forall \mathbf{K}R_S.\mathbf{K}S \sqcup \mathbf{K}\neg S \tag{6.7}$$

This axiom expresses that after having performed the action R_S the robot knows the truth value of the sensed proposition S, i.e. it knows *whether* S holds or not.

We enforce the following **frame axiom schemas** (Γ_{FR}):

$$\mathbf{K}\varphi \sqsubseteq \forall \mathbf{K}R_S.\mathbf{K}\varphi \tag{6.8}$$

one for each sensing action R_S, where φ stands for any $\mathcal{ALC}$ concept.[2] This propagates all concepts that hold in the current state s to the next state s'. The expression $\neg\mathbf{A}S \sqcap \neg\mathbf{A}\neg S$ in the premises of the precondition axioms for R_S prevents the execution of R_S in case either $\mathbf{K}S$ or $\mathbf{K}\neg S$ holds in the previous state. Hence, no contradiction may be generated from instances of the frame axiom schemas and the effect axiom for R_S.

Planning problem with sensing. The robot's sensing ability can be used to extend the notion of plan considered before to the notion of *conditional plan*. Indeed the robot may use its sensing capability to choose different courses of actions leading to a given goal, depending on the value of the sensed propositions. The planning problem then becomes:

$$\Sigma \models COND\_PLAN\_FOR\_G(init) \tag{6.9}$$

where: (i) Σ is the knowledge base including the static axioms Γ_S, the action precondition axioms Γ_P and the effect axioms Γ_E for both moving and sensing actions plus the frame axiom schema Γ_{FR} for the sensing actions, and the initial state description axioms Γ_I; (ii) $COND\_PLAN\_FOR\_G(init)$ denotes that $COND\_PLAN\_FOR\_G$ holds in the initial state, where $COND\_PLAN\_FOR\_G$ is *any* concept belonging to the set $\mathcal{P}_\mathcal{C}$ defined inductively as follows:

1. $\mathbf{K}G \in \mathcal{P}_\mathcal{C}$;
2. if $C \in \mathcal{P}_\mathcal{C}$, then $\exists\mathbf{K}R_{M_i}.C \in \mathcal{P}_\mathcal{C}$, for every moving action R_{M_i};
3. if $C_1, C_2 \in \mathcal{P}_\mathcal{C}$, then $\exists\mathbf{K}R_{S_i}.(\mathbf{K}S_i \sqcap C_1) \sqcup (\mathbf{K}\neg S_i \sqcap C_2) \in \mathcal{P}_\mathcal{C}$, for every sensing action R_{S_i}.

6.3.3 Reasoning

Let us now turn our attention to the problem of computing the logical implication (6.5) or (6.9). It is worth noticing that in general the $\mathcal{ALCK}_{\mathcal{NF}}$-knowledge base $\Sigma = \langle\Gamma, \{S(init)\}\rangle$ has many preferred models, which are distinguishable even up to the renaming of states. Nevertheless, due to the special form of the epistemic inclusion assertions corresponding to the dynamic axioms in Σ, we can build the so-called *first-order extension* (FOE) of Σ [31], which consists of the knowledge base $\langle\Gamma_S, \{S(init)\}\rangle$ augmented by a set of membership assertions which are consequences (up to the renaming of states) of the epistemic inclusion assertions.

The FOE of Σ provides a unique characterization of the knowledge that is shared by all the preferred models of Σ. In fact, by representing dynamic axioms by means of epistemic inclusion assertions of special form above, we recover the ability of representing the behavior of the system by means of a *single* graph, corresponding to the FOE. Such a graph is built in the following way:

- Nodes correspond to individuals denoted by chains of $\mathbf{K}R$ of any length starting from *init* and are labeled by the properties that are known for the corresponding individual.

[2] In fact, a number of instances which is linear in the size of the knowledge base suffices (see the following section).

- There is an edge, labeled by R, from a node x to a node y, if y is an $\mathbf{K}R$-successor of x.

The FOE summarizes the common part of all transition graphs that, by our (partial) knowledge about the dynamic system, are considered possible, thus providing a description of the actual transition graph which is partial, in the sense that: (i) certain states and transitions may be missing; (ii) the properties of the states in the graph may be only partially specified.

In order to compute the first-order extension, we replace each sensing action R_S by two special actions R_S^+ and R_S^-. We denote by $\Gamma_E^{\pm}$ the set of effect axioms Γ_E in which those for the sensing actions R_S are replaced by:

$$\mathbf{K}\top \sqsubseteq \forall \mathbf{K}R_S^+.\mathbf{K}S \qquad\qquad \mathbf{K}\top \sqsubseteq \forall \mathbf{K}R_S^-.\mathbf{K}\neg S.$$

We also use only a finite number of instances of the frame axiom schemas. We denote by $\Gamma_{IFR}^{\pm}$ the set of axioms:

$$\mathbf{K}C \sqsubseteq \forall \mathbf{K}R_S^+.\mathbf{K}C \qquad\qquad \mathbf{K}C \sqsubseteq \forall \mathbf{K}R_S^-.\mathbf{K}C$$

obtained by: (1) instantiating the frame axiom schemas in Γ_{FR} for each concept C such that either $C(init) \in \Gamma_I$, or $\mathbf{K}C$ is in the postcondition of some effect axiom in Γ_E (i.e., C such that $\mathbf{K}D \sqsubseteq \forall \mathbf{K}R_M.\mathbf{K}C$, or $C, \neg C$ such that $\mathbf{K}\top \sqsubseteq \forall \mathbf{K}R_S.\mathbf{K}C \sqcup \mathbf{K}\neg C$ in Γ_E); (2) replacing each sensing action R_S by the two special actions R_S^+ and R_S^-.

Let $\Gamma = \Gamma_S \cup \Gamma_P \cup \Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm}$ be the set of static axioms, precondition axioms, effect axioms, and frame axioms, specifying the behavior of the dynamic system, let $S(init)$ be the specification of the initial state $init$, and let $\Sigma = \langle \Gamma, \{S(init)\}\rangle$. The FOE of Σ, written $FOE(\Sigma)$, is computed by the algorithm shown in Fig. 6.2, in which

$$POST(s, R, \Gamma_S \cup \mathcal{A}, \Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm}) = \{D \mid \mathbf{K}C \sqsubseteq \forall \mathbf{K}R.\mathbf{K}D \in \Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm} \ \wedge\ \Gamma_S \cup \mathcal{A} \models C(s)\}$$

denotes the set of properties representing the effects of the execution of the action R in the state s, and

$$CONCEPTS(\Gamma_S \cup \mathcal{A}, s) = \{D \mid \Gamma_S \cup \mathcal{A} \models D(s)\}$$

denotes the set of properties verified by the state s in Σ.

Informally, the algorithm, starting from the initial state $init$, iteratively proceeds as follows. First, it finds an action R which can be executed in the current state, by identifying in the set Γ_P a precondition axiom for R whose left-hand side is logically implied by the current knowledge base. Then, it propagates the effects of the action R, which again is based on checking whether the left-hand side of each effect axiom for R in the set Γ_E is logically implied by the properties holding in the current state. In this way, the set of properties corresponding to the effect of the execution of R in the current state is computed. A new state is then generated, unless a state with the same properties has already been created. This step is repeated until all actions executable in the current state have been considered. Then, a new current state is chosen among those previously created and the main iteration proceeds, ultimately producing a sort of non-epistemic "completion" of the knowledge base Σ.

The FOE is unique, that is, every order of extraction of the states from the set ACTIVE_STATES produces the same set of assertions, up to the renaming of states. Moreover, the algorithm terminates, that is, the condition ACTIVE_STATES $= \emptyset$ is eventually reached, since the number of states generated is bounded by the number of rules in $\Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm}$. Finally, the condition

$$CONCEPTS(\Gamma_S \cup \mathcal{A}, s) = CONCEPTS(\Gamma_S \cup \mathcal{A}', s')$$

can be checked by verifying whether for each concept C, such that either $C(init) \in \Gamma_I$ or $\mathbf{K}C$ is in the postcondition of some axiom in $\Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm}$, $\Gamma_S \cup \mathcal{A} \models C(s)$ iff $\Gamma_S \cup \mathcal{A}' \models C(s')$.

ALGORITHM FOE
INPUT: $\Sigma = \Gamma_S \cup \Gamma_P \cup \Gamma_E \cup \Gamma_{FR} \cup \Gamma_I$
OUTPUT: $FOE(\Sigma)$
  PROCEDURE CREATE_NEW_STATE(s, R)
  begin
    s' = NEW state name;
    $\mathcal{A}' = \mathcal{A} \cup \{R(s,s')\} \cup \{D(s') \mid D \in POST(s, R, \Gamma_S \cup \mathcal{A}, \Gamma_E^{\pm} \cup \Gamma_{IFR}^{\pm})\}$
    if there exists a state $s'' \in$ ALL_STATES **such that**
      $CONCEPTS(\Gamma_S \cup \mathcal{A}, s'') = CONCEPTS(\Gamma_S \cup \mathcal{A}', s')$
    then $\mathcal{A} = \mathcal{A} \cup R(s, s'')$
    else begin
        $\mathcal{A} = \mathcal{A}'$;
        ACTIVE_STATES = ACTIVE_STATES $\cup\{s'\}$;
        ALL_STATES = ALL_STATES $\cup\{s'\}$
      end
  end;
begin
  ACTIVE_STATES = $\{init\}$;
  ALL_STATES = $\{init\}$;
  $\mathcal{A} = \Gamma_I$;
  repeat
    s = choose(ACTIVE_STATES);
    for each moving action R_M **do**
      if there exists $\mathbf{K}C \sqsubseteq \exists \mathbf{K}R_M.\top \in \Gamma_P$
        such that $\Gamma_S \cup \mathcal{A} \models C(s)$
      then
        CREATE_NEW_STATE(s, R_M);
    for each sensing action R_S **do**
      if there exists $\mathbf{K}C \sqcap \mathbf{A}S \sqcap \mathbf{A}\neg S \sqsubseteq \exists \mathbf{K}R_S.\top \in \Gamma_P$
        such that $\Gamma_S \cup \mathcal{A} \models C(s)$ **and** $\Gamma_S \cup \mathcal{A} \not\models S(s)$ **and** $\Gamma_S \cup \mathcal{A} \not\models \neg S(s)$
      then begin
        CREATE_NEW_STATE(s, R_S^+);
        CREATE_NEW_STATE(s, R_S^-)
      end;
    ACTIVE_STATES = ACTIVE_STATES $-\{s\}$
  until ACTIVE_STATES = $\emptyset$;
  return $\Gamma_S \cup \mathcal{A}$
end.

Figure 6.2: Algorithm computing $FOE(\Sigma)$

The first-order extension of the epistemic knowledge base describing the dynamic system constitutes the basis of a sound and complete planning method. Let us first introduce a translation function $\tau(\cdot)$ for conditional plans.

Definition 1. *Let C be a concept expression representing a plan (i.e. belonging to the set $\mathcal{P}_\mathcal{C}$). Then, $\tau(C)$ is the concept expression obtained as follows:*

1. *if $C = \mathbf{K}G$ then $\tau(C) = \mathbf{K}G$;*
2. *if $C = \exists \mathbf{K}R_{M_i}.C_1$ then $\tau(C) = \exists \mathbf{K}R_{M_i}.\tau(C_1)$;*
3. *if $C = \exists \mathbf{K}R_{S_i}.(\mathbf{K}S_i \sqcap C_1) \sqcup (\mathbf{K}\neg S_i \sqcap C_2)$ then $\tau(C) = \exists \mathbf{K}R^+_{S_i}.\tau(C_1) \sqcap \exists \mathbf{K}R^-_{S_i}.\tau(C_2)$.*

The planning problem in Σ expressed by (6.9) can be thus reduced to the following entailment problem in $FOE(\Sigma)$

$$FOE(\Sigma) \models \tau(COND\_PLAN\_FOR\_G)(init)$$

6.4 Implementation of a conditional planner

A conditional planner based on the above formalism has been implemented by making use of a well-known efficient and reliable knowledge representation system based on DLs, CLASSIC [12].

An interesting feature of our approach is that the reasoning tools provided by such a system can be effectively used in the implementation of the theoretical framework. In fact, we make use of the built-in instance checking mechanism to check the validity of a concept in a state, and of triggering of rules to propagate effects of actions. However, the language for representing knowledge in CLASSIC is less expressive than the DL we have considered so far. So, we use a subset of the DL language corresponding to some of the constructs available in CLASSIC, that we write, for ease of notation, using $\sqcap$ for `AND` and $\forall$ for `ALL`.

Static axioms are expressed either as inclusion assertions ($\dot{\leq}$) or as concept definitions, written $\doteq$ and interpreted as necessary and sufficient conditions (see for example [14]). In this way we can write

$$A \dot{\leq} C \qquad\qquad A \doteq C$$

to define the primitive concept A as a subset of C, or equivalent to C, respectively. In both cases cycles are not allowed.

Dynamic axioms are represented as CLASSIC *rules*, denoted with $\mapsto$. Observe that they implement the epistemic sentences $\mathbf{K}C \sqsubseteq D$, hence the rule is fired only on named individuals satisfying the antecedent of the rule. The concept expression $\exists \mathbf{K}R.\top$ has been implemented by defining a procedural hook in CLASSIC, that we call `KAPPA` and abbreviate with $\exists_{\mathrm{K}}$. Rules representing action precondition axioms and effect axioms are thus written respectively as

$$C \mapsto \exists_{\mathrm{K}} R \qquad\qquad C \mapsto \forall R.D$$

When C is the same concept expression, we sometimes abbreviate $C \mapsto \exists_{\mathrm{K}} R$ and $C \mapsto \forall R.D$ as $C \mapsto \exists_{\mathrm{K}} R \sqcap \forall R.D$.

Notice that the epistemic inclusion assertion $\mathbf{K}C \sqsubseteq \forall \mathbf{K}R.\mathbf{K}D$ is correctly implemented in this setting by the CLASSIC rule $C \mapsto \forall R.D$, since in CLASSIC rules are only applied to individuals explicitly mentioned in the knowledge base.

Finally, concept instance assertions are used to describe initial state properties. Thus

$$C(init)$$

states that C is valid in the initial state. The goal to be achieved is again expressed by means of a concept describing the properties that must hold in the final state.

Let us further comment on the use of `KAPPA` for implementing **K**. The procedural meaning of the `KAPPA` operator is given according to the notion of known individuals at the epistemic level, thus an individual x is instance of (`KAPPA` R) if and only if there exists a named individual y such that $(x, y) \in R^{\mathcal{I}}$. In fact, if x is asserted to belong to (`KAPPA` R) and there is no named R-successor of x, then a new named individual is created and added to the knowledge base as R-successor of x. Subsequently, the new individual can be used for firing rules.

The extension for dealing with sensing actions involves the introduction of three basic ideas.

1. Since in CLASSIC the operator $\neg$ is not available, we use, when needed, a new concept $not{-}S$ for indicating $\neg S$. Observe that in general $not{-}S$ and $\neg S$ are not semantically equivalent, that is $(not{-}S)^{\mathcal{I}} \neq (\neg S)^{\mathcal{I}}$.

2. Two special actions R_S^+ and R_S^- are introduced for each sensing action R_S and the effect of sensing actions is expressed by

$$\top \mapsto \forall R_S^+.S$$
$$\top \mapsto \forall R_S^-.not{-}S$$

3. In the precondition of sensing actions the term $\neg\mathbf{A}S \sqcap \neg\mathbf{A}\neg S$ is replaced by a procedural hook and that checks if the current individual does not belong to both the concept S and $not{-}S$. We call this hook (`NOTKNOWN` S) and we abbreviate it in the formulae with ($\mathrm{not_K}$ S).

   Therefore the precondition axioms are written

$$C \sqcap (\mathrm{not_K}\ S) \mapsto \exists_{\mathrm{K}} R_S^+$$
$$C \sqcap (\mathrm{not_K}\ S) \mapsto \exists_{\mathrm{K}} R_S^-$$

The generation of plans can be obtained by slightly modifying the algorithm for computing the FOE. Indeed, it is sufficient to compute only a portion of the FOE, by terminating the computation as soon as a state satisfying the goal is generated. Then, one can build the plan, namely a term representing a sequence of actions that connect the initial state with the state in which the goal is satisfied. If sensing actions are involved in this plan, then a new planning task is required for exploring the other possible value of the sensing property. In this way a branch in the plan, corresponding to the sensing action, is added. In other words, the plan generation procedure produces a tree where branches are determined by the possible results of sensing actions.

The implementation relies on the following assumptions (directly derived from the FOE algorithm), that are necessary for the termination of the plan generation procedure: (i) rules involving `KAPPA` must be applied only when no other rule can be fired; this prevents incomplete state generations, i.e. new states are generated only after the properties of existing ones have been completely derived; (ii) when a new individual z is created and all the rules able to add properties to it are fired, a search for an existing individual that is equivalent to z (i.e. described by the same concepts) is done, and if such an individual (say x) is found, $z = x$ is imposed, so that individuals are not indefinitely generated.

It is worth noting that different activations of the plan generation procedure may compute the same part of the FOE several times. Therefore, we have adapted the algorithm to reuse the already

computed portion of the FOE, until changes in the world require the knowledge base to be updated. This modification can lead to a significant reduction of the computation time. Furthermore, when the construction of the entire FOE is feasible, a plan can be obtained in two steps. First, one generates the entire FOE following the algorithm in Fig. 6.2; then the plan is searched for in the FOE. In this way, the first step can be seen as a pre-processing of the knowledge base and it allows for a faster plan generation.

The complexity of the plan generation procedure above is polynomial with respect to the size of FOE(Σ), that is the number of new states (individuals) generated by the FOE algorithm, and so, in general, exponential with respect to to the size of Σ.

We observe that, under the assumption that dynamic axioms are of the form $C \mapsto \exists_{\mathrm{K}} R \sqcap \forall R.D$ and for each action R the precondition C in dynamic axioms involving R are disjoint from each other, we can make use of the CLASSIC construct `FILLS` instead of `KAPPA` for writing such axioms. The `FILLS` construct, that we denote as $\exists R.\{a\}$, has the following semantics:

$$(\exists R.\{a\})^{\mathcal{I}} = \{x \in \Delta \mid (x, a) \in R^{\mathcal{I}}\}$$

and can be intuitively interpreted as the set of individuals having a as R-successor. Note that a new individual a is created, if it does not exist, and therefore it is a known individual at the epistemic level. So dynamic axioms can be written as

$$C \mapsto \exists R.\{a\} \sqcap \forall R.D$$

In this way, computing the FOE is done in polynomial time with respect to the size of the knowledge base, because the number of individuals is at most linear in the number of rules. Hence the plan generation task is done in polynomial time too. However the restriction introduced for using `FILLS` limits the expressiveness in the representation of environments.

In any case, the large number of states, due to the presence of different branches related to sensing actions, is a real difficulty for conditional planners. However, in most practical situations, the knowledge obtained by sensing actions does not need to be propagated through all moving actions. For example, when the robot senses whether a door is open, it could use this information just to decide whether or not to enter this door, and then forget it when the selected moving action is executed. In this case the number of states is reduced since we forget sensed properties and so, after a moving action, we have a single successor state regardless of the result of sensing.

With respect to the propagation of sensed knowledge, we can thus distinguish the following two limit cases:

1. Sensed knowledge is propagated only through sensing actions, according to the frame axiom schemas Γ_{FR}. The sub-graph generated by a sequence of sensing actions is such that all its states that have an R_M-successor for some moving action R_M have in fact the same R_M-successor. In other words there is a confluence of edges labeled with R_M from the states in the sub-graph to a single successor state. In this case we get the "minimum" number of possible resulting states. Observe that this confluence is the result of forgetting sensed knowledge acquired in the sub-graph.

2. Sensed knowledge is propagated through every action. This requires the use of explicit frame axioms for propagating sensed propositions through moving actions; such frame axioms are effect axioms of the form $\mathbf{K}S \sqsubseteq \forall \mathbf{K} R_M.\mathbf{K}S$, one for each moving action R_M and for each sensed proposition S. In this case we get the "maximum" number of possible resulting states.

It is straightforward in our setting to model which (if any) knowledge acquired by sensing is propagated: it is sufficient to write explicit frame axioms about the persistence of the chosen sensed

properties through moving actions. On the other hand, we assume that all knowledge is propagated thorough sensing actions by enforcing the related frame axioms schemas in Γ_{FR}.

Another problem is related to the notion of conditional plan we have given with the definition of the set $\mathcal{P}_{\mathcal{C}}$. Indeed our notion of plan is quite strong, since we require a plan to exist whatever the truth values of sensed propositions are. Consider, for example, an environment in which a room can be accessed by two doors. If it is not guaranteed that at least one of the two doors is open, than there is not a conditional plan for reaching the room.

We can introduce a weaker notion of conditional plan. In a *weak conditional plan* we only require that at least one branch will lead to the goal. It is straightforward to modify our formal notion of plan accordingly. Our planner is able to generate a weak conditional plan, if a strong one does not exist.

6.4.1 CLASSIC representation of dynamic systems

Summarizing a CLASSIC knowledge base for representing a dynamic systems is constituted by the following axioms:

- *static axioms*

$$A \dot{\leq} C \qquad\qquad A \doteq C$$

  where A is a primitive concept and C is a concept;

- *precondition axioms*

$$C \mapsto \exists_{\mathrm{K}} R$$

  for moving actions and

$$C \sqcap (\mathrm{not}_{\mathrm{K}}\ S) \mapsto \exists_{\mathrm{K}} R_S^+$$
$$C \sqcap (\mathrm{not}_{\mathrm{K}}\ S) \mapsto \exists_{\mathrm{K}} R_S^-$$

  for sensing actions;

- *effect axioms*

$$C \mapsto \forall R.D$$

  for moving actions and

$$\top \mapsto \forall R_S^+.S$$
$$\top \mapsto \forall R_S^-.not{-}S$$

  for sensing actions;

- *frame axioms*

$$C \mapsto \forall R.C$$

- *initial state axioms*

$$C(init)$$

  where *init* is an individual denoting the initial state.

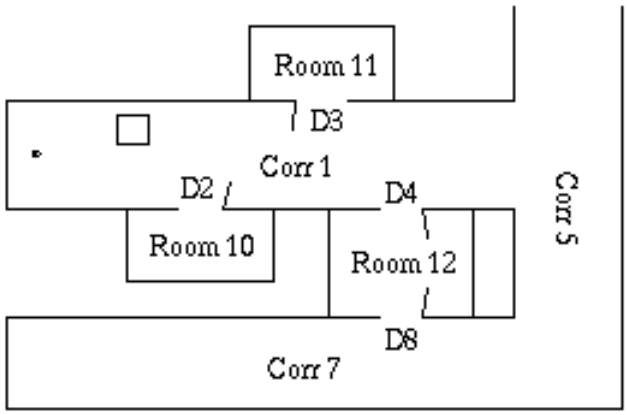


Figure 6.3: Plan execution

6.4.2 Example

Let us consider now the following example: in the map of Fig. 6.3 a room is accessible from two different doors (named *Door4* and *Door8*), the robot is in the upper corridor and its goal is to reach this room.

A portion of the CLASSIC knowledge base describing this environment is given by the following axioms:

$$
\begin{aligned}
CloseToDoor4 &\doteq Corridor1 \\
CloseToDoor8 &\doteq Corridor7
\end{aligned}
$$

$$
\begin{aligned}
Corridor1 &\mapsto \exists_{\mathrm{K}} FollowC1ToD4 \sqcap \forall FollowC1ToD4.CloseToDoor4 \\
Corridor1 &\mapsto \exists_{\mathrm{K}} FollowC1ToC5 \sqcap \forall FollowC1ToC5.Corridor5 \\
Corridor5 &\mapsto \exists_{\mathrm{K}} FollowC5ToC7 \sqcap \forall FollowC5ToC7.Corridor7 \\
Corridor7 &\mapsto \exists_{\mathrm{K}} FollowC7ToD8 \sqcap \forall FollowC7ToD8.CloseToDoor8 \\
CloseToDoor4 \sqcap OpenDoor4 &\mapsto \exists_{\mathrm{K}} EnterD4 \sqcap \forall EnterD4.Room12 \\
CloseToDoor8 \sqcap OpenDoor8 &\mapsto \exists_{\mathrm{K}} EnterD8 \sqcap \forall EnterD8.Room12 \\
Room12 \sqcap OpenDoor4 &\mapsto \exists_{\mathrm{K}} ExitD4 \sqcap \forall ExitD4.CloseToDoor4 \\
Room12 \sqcap OpenDoor8 &\mapsto \exists_{\mathrm{K}} ExitD8 \sqcap \forall ExitD8.CloseToDoor8
\end{aligned}
$$

$$
\begin{aligned}
CloseToDoor4 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor4) &\mapsto \exists_{\mathrm{K}} SenseOpenDoor4^{+} \\
CloseToDoor4 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor4) &\mapsto \exists_{\mathrm{K}} SenseOpenDoor4^{-} \\
CloseToDoor8 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor8) &\mapsto \exists_{\mathrm{K}} SenseOpenDoor8^{+} \\
CloseToDoor8 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor8) &\mapsto \exists_{\mathrm{K}} SenseOpenDoor8^{-} \\
\top &\mapsto \forall SenseOpenDoor4^{+}.OpenDoor4 \\
\top &\mapsto \forall SenseOpenDoor4^{-}.ClosedDoor4 \\
\top &\mapsto \forall SenseOpenDoor8^{+}.OpenDoor8 \\
\top &\mapsto \forall SenseOpenDoor8^{-}.ClosedDoor8
\end{aligned}
$$

$$
\begin{aligned}
OpenDoor4 &\mapsto \forall FollowC1ToD4.OpenDoor4 \\
OpenDoor4 &\mapsto \forall FollowC1ToC5.OpenDoor4
\end{aligned}
$$

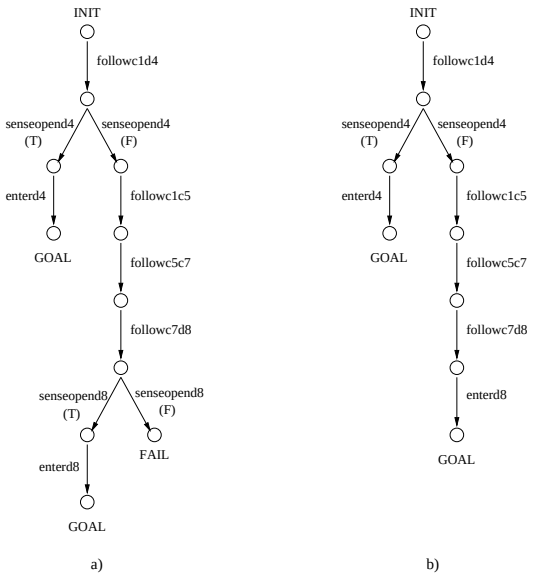


Figure 6.4: Conditional plans

$$
\begin{array}{rcl}
\ldots & & \\
OpenDoor8 & \mapsto & \forall FollowC1ToD4.OpenDoor8 \\
\ldots & & \\
CloseToDoor4 & \mapsto & \forall SenseOpenDoor4^{+}.CloseToDoor4 \\
\ldots & &
\end{array}
$$

$$Corridor1(init)$$

Observe the ramification obtained with static axioms. In the example, the concept *CloseToDoor4* is defined as a specialization of *Corridor1*, that is when the robot is close to the first door, it is also in the first corridor. Now consider the dynamic axiom involving the *ExitD4* action. It has just *CloseToDoor4* as effect, but, thanks to the above static axiom, in the successor state of *ExitD4* the action *FollowC1ToC5* (whose precondition is *Corridor1*) is also executable. In other words, static axioms allows for producing many edges of the graph from a single dynamic axiom.

Suppose that the robot does not have any static knowledge about these doors. Then a strong conditional plan does not exist and the planner is able to generate the weak conditional plan graphically represented in Fig. 6.4a), which is a weak plan because there is a branch in which the plan fails, corresponding to the situation in which both the doors are closed.

Suppose now that in the same situation we say the robot that at least one of the two doors is open, by adding the following static axiom:[3] $\top \sqsubseteq Door4Open \sqcup Door8Open$. In this case the strong plan of Fig. 6.4b) will be generated, in which, if the first door is known to be closed, then no sensing action is done on the other door, which is known to be open.

[3] The disjunction, which is not present in CLASSIC, is simulated by the rules $\mathbf{K}Door4Closed \sqsubseteq Door8Open$ and $\mathbf{K}Door8Closed \sqsubseteq Door4Open$.

Chapter 7

A Multiresolution Stereo Vision System

In this chapter we describe new multiresolution and calibration techniques for a stereo vision system, that has been implemented on the root in order to increase its sensing capabilities. In fact, images contain a very large amount of input data with respect to other sensor data (such as sonar readings), and therefore the reliability of knowledge acquisition techniques is significantly improved.

Stereoscopic vision is a technique for inferring the 3D position of objects from two or more simultaneously view of the scene. Mobile robots can use a stereo vision system as a reliable and effective way to extract range information from the environment. Accuracy of results for some applications (like navigation) are adequate, and real-time implementations have been realized [47, 52] on low-cost hardware. Moreover, stereo vision is a passive sensor and there is no interference with other sensor devices (in particular when multiple robots are present in the environment). Finally, it can be easily integrated with other vision techniques, such as object recognition and tracking. Implementation of stereo vision systems for mobile robots [16, 85], used in map building and navigation tasks, can process up to 5 frames per second, that is a frame rates of 5 Hz, by making use of special-purpose hardware.

We describe a real-time multiresolution stereoscopic vision system [45], that has been implemented on a Pioneer mobile robot. It makes use of the STH-V1 Integrated Stereo Head and a fast stereo algorithm developed at SRI running on a conventional PC architecture [52]. Two features characterize the system.

- A multiresolution technique for increasing the depth of view of the robot. Unlike previous methods using a pyramid scheme, this method has a uniform search model at each resolution, and so is amenable to fast, uniform correspondence algorithms.

- A semi-automatic calibration method for removing lens distortion and computing the parameters of the stereo head. This calibration method separates the calibration of internal and external camera parameters, without requiring any information about correspondences between real world and image coordinates. Further, we have determined and corrected the effect of small variations in internal parameters on the accuracy of the disparity correspondences.

Our approach guarantees real-time and accurate results by making use of conventional low-cost hardware. The system can process more than 10 frames per second producing three depth maps with different resolution and maximum disparities: 320x60x24, 160x60x16, and 80x60x16.

7.1 Stereo Vision

Reconstruction of the world seen through stereo cameras can be divided in two steps:

1. *Correspondence problem.* For every point in one image find out the correspondent point on the other image and compute the disparity (distance in pixels) of these points.

2. *Triangulation.* Given the disparity map, the focal distance of the two cameras and the geometry of the stereo setting (relative position and orientation of the cameras) compute the (X,Y,Z) coordinates of all points in the images.

Both the above tasks are easier to realize when cameras are in a *standard setting*, that is with parallel optical axes and coplanar image planes. In this setting epipolar lines correspond to rows in the frame buffer and point correspondences are searched over these rows.

Since the amount of work in the correspondence problem increases linearly with the size of the disparity search window, it is desirable to limit the search size. But given that disparity increases as the inverse of object distance, the number of disparities needed to correspond close objects grows very large. In order to increase the stereo depth of view (*horopter*) of the robot without increasing the disparity search (and hence the computational time) by this large factor, we propose a special kind of *multiresolution* for stereo vision.

The second issue that must be considered when designing and implementing a stereo vision system is *stereo head calibration.* This is required because of two factors: (i) both the correspondence problem and triangulation make the assumption of an ideal model of the camera (*pinhole model*), that can be very different from actual (low-cost) imaging devices; (ii) the relative position and orientation of the two cameras must be known in order to determine range information. Camera calibration is the task of relating the ideal pinhole model of the camera with an actual imaging device (*internal calibration*) and of determining the relative position and orientation of the cameras (*external calibration*). This is a fundamental step for 3D reconstruction and in particular for stereoscopic vision analysis. It allows not only for determining the geometry of the stereo setting (needed for triangulation), but also for removing radial distortions provided by common lenses.

7.2 Multiresolution Stereo Vision

In this section we develop a multiresolution stereo vision technique to obtain range results for close objects. Multiresolution techniques are often used for improving performance of the matching algorithm by first looking in the low resolution pair of image for correspondences, and then refining the local search in a high resolution pair [10, 70]. This *refinement* method is reasonable if the computational hardware allows the search to be customized at different points in the image. However, our algorithms depend on uniformity of application for their speed, so that it is impractical to search over different disparities.

Here we propose a slightly different multiresolution technique that applies the same stereo algorithm to images at different resolutions independently, and then combines the disparity results. Given a pair of images sized 320x120 pixels (the high-resolution pair), we first generate two other pairs of images sized 160x120 and 80x120 pixels (the medium and the low-resolution pairs). Each pair is processed by the same stereo algorithm and three different *disparity maps*, containing for each position the disparity of the corresponding point in the other image, are produced. Observe that, in order to maintain all the values in the maps aligned, medium and low-resolution disparity maps need to be horizontally expanded by respectively a factor 2 and 4, and the disparities need to be multiplied respectively by 2 and 4. Once the disparity maps are computed we sample a line

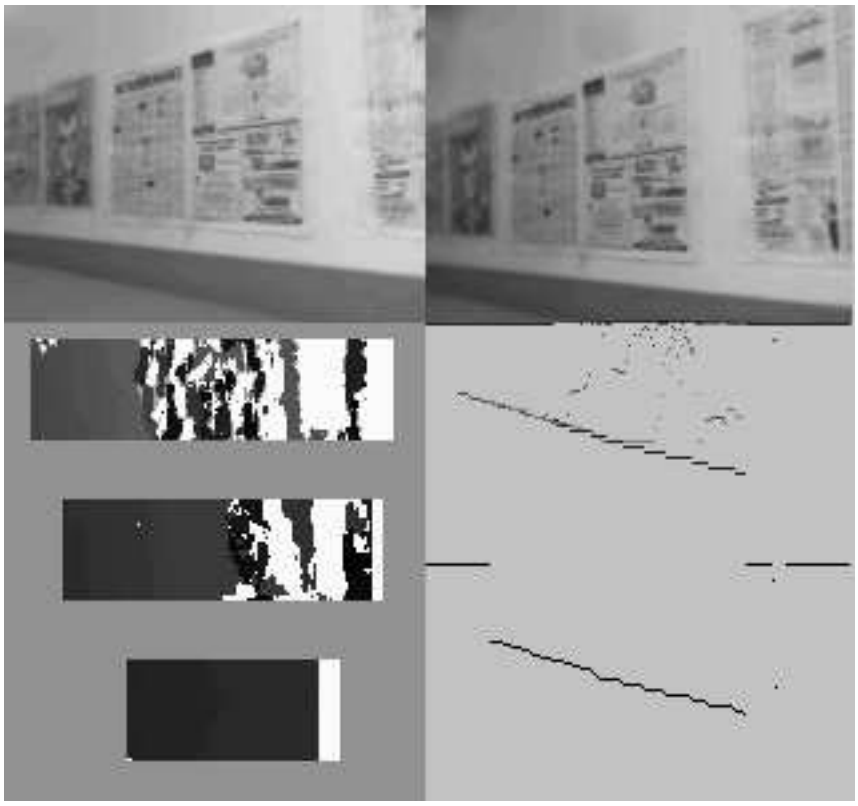


Figure 7.1: Multiresolution technique

in the middle, since we are interested in 2D range information, and thus we obtain three *disparity lines*.

The fundamental point in this multiresolution technique is how to combine three disparity lines into a single one. We chose a simple threshold-based method, that is justified by the property that, given a fixed maximum disparities, objects closer to the stereo camera are better "seen" with the low-resolution images, while object farther are also "visible" with the high-resolution ones. Therefore, for each position we accept disparities from the low-resolution images if they exceed a first threshold, from the medium-resolution ones if they exceed a second (lower) threshold, otherwise we accept value from the high-resolution images.

In this way disparities from high resolution images are used for farther objects (little disparities), while disparities from low resolution ones are used for closer objects (large disparities).

In Figure (7.1) it is shown how the multiresolution works. In the upper side there are the two images taken from the cameras. The left-bottom side contains (from the top) the high, the medium, and the low-resolution disparity maps. The disparity maps are represented with a greyscale image where brighter pixels correspond to higher disparities. Noisy black and white zones represent errors in determining point correspondences. In the middle-right, three linear samples of the disparity maps, we call them *disparity lines*, are plotted: the zero disparity value (corresponding to infinity range) is in the top, and higher disparity values increase to the bottom, so that closer objects "appear" in the bottom.[1] Finally, the combination of the three lines is represented in the right-bottom side.

Notice in the figure the noise in the high and the medium resolution images provided by the closer part of the wall. This noise is indicated by the not uniform black and white zone in the

[1] Although it cannot be seen in the picture, the three lines are displayed together using different colors.

disparity maps, as well as the isolated points in the disparity lines. This noise is fully removed when the disparity lines are combined together, as shown in the final disparity line.

The main advantage of this *independent* multiresolution technique is that it preserves the speed of the original uniform correspondence algorithm. In the above example, with a search of 24 disparities in the high resolution images, the minimum distance of correspondence objects is 90 cm. To decrease this to 30 cm would require increasing the search range to 96 disparities, an increase of 4x in computation. On the other hand, the multiresolution method uses images at 1/2 and 1/4 the horizontal resolution, so the total time is $1 + 1/2 + 1/4 = 7/4$ of the original.

7.3 Stereo Head calibration

Stereoscopic vision analysis make use of an ideal model of the stereo camera for a simplified computation of range data. In order to relate the ideal camera model to an actual imaging device, a set of calibration parameters must be found.

These parameters can be classified in two groups:

1. *internal* (or intrinsic) parameters: geometric and optical characteristics of the lenses and the imaging device.

2. *external* (or extrinsic): position and orientation of the camera in a world reference system.

Internal and external calibration refers to determining respectively internal and external parameters.

Calibrating stereo heads is usually dealt with by calibrating each camera independently and then applying geometric transformation of the external parameters to find out the geometry of the stereo setting. A calibration method for stereo heads is proposed in [36], but it does not consider lens distortion, while in our case this is the main reason of errors in range determination.

A well known method for calibrating a camera taking into account lens distortion has been proposed by Tsai [84, 55]. The method is based on the knowledge of the position of some points in the world and the correspondent projections on the image. The first step is to solve a linear equation to find out some of the external parameters, and then nonlinear optimization is performed for finding out the others. The procedure can be iterated in order to improve accuracy. This calibration procedure requires the user to use a calibration grid (that must be accurately prepared) and to individuate the projections of calibration points in the image.

Another class of calibration methods includes those that do not need any knowledge about the position of points in the world. In this case it is possible to separate internal calibration from external calibration, and the optimization step can be performed only over the internal parameters. An example of this method for internal calibration has been proposed in [29]. It is based on minimizing the curvature of segments in the world determined with edge detection and polynomial approximation from scenes of structured environments.

We are proposing a semi-automatic calibration technique for stereo heads in standard setting. It is called semi-automatic because the user is neither required to provide any geometric data about the scene used for calibration, nor to identify real points correspondences on the images.

Internal calibration is performed independently for each camera and it is based on minimizing radial distortion of an image containing only straight lines. External calibration is done on the overall stereo head by exploiting some properties of the disparity maps provided by the stereo algorithm.

The method has been fully implemented and used for calibrating the stereo head of a mobile robot. Accuracy has been evaluated by measuring range information provided by the system, in comparison with data obtained without calibration.

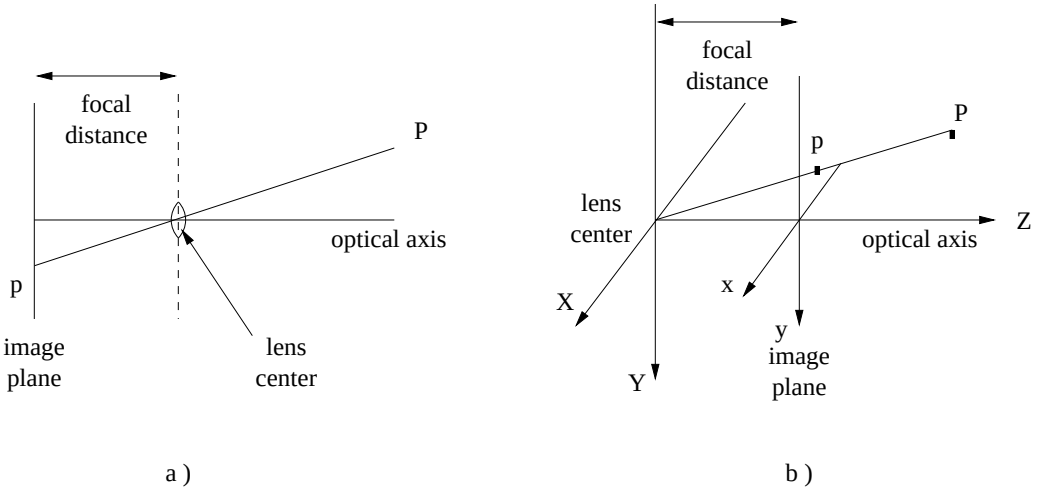


Figure 7.2: The pinhole camera model

7.3.1 The pinhole camera model

The *pinhole model* of a camera is broadly used in computer vision and in particular in stereo vision. Fig. 7.2 illustrates this ideal model characterized by the assumption of dealing with an infinitely small lens, that allows for a perfect focus on the overall scene and does not introduce any distortion.

Given a point (X, Y, Z) in a reference system with origin in the lens center, and its projection (x, y) in the image, the equation of *perspective geometry* are

$$x = f\frac{X}{Z}$$
$$y = f\frac{Y}{Z}$$

where f is the focal distance of the lens.

Notice that coordinates (x, y) are expressed in a continuous reference system with the origin on the center of the image, that is the intersection of the optical axes with the image plan. However images are usually represented by a discrete matrix of pixels (called *frame*), indexed by (i, j) with the origin in the upper-left corner. The relation between image and frame coordinates, with orientation of axis given in Fig. 7.2b, is thus the following

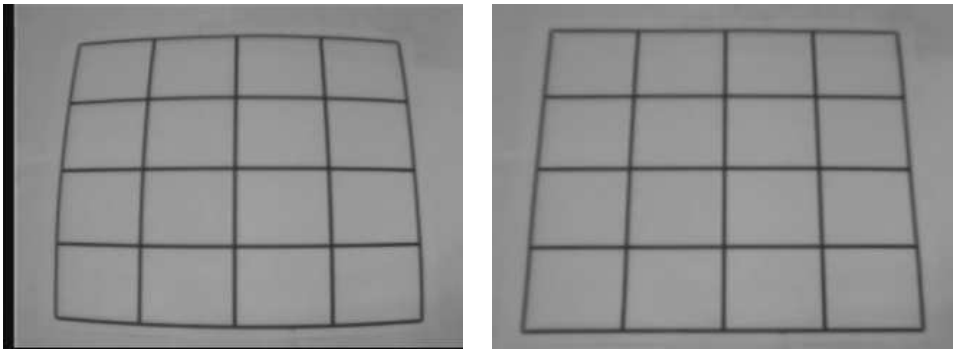
$$\begin{aligned} x &= (i - O_x)P_x \\ y &= (j - O_y)P_y \end{aligned}$$

where P_x and P_y are the horizontal and the vertical dimensions of a pixel in the discrete matrix, and (O_x, O_y) are the indexes in the matrix corresponding to the center of the image. It is important to observe that usually the center of the image is *not* coincident with the median pixel in the matrix, that is $O_x \neq Width/2$ and $O_y \neq Height/2$.

Finally radial distortion introduced by real lenses is modeled by the following equations

$$\begin{aligned} x &= x_d(1 + k_1 r_d^2) \\ y &= y_d(1 + k_1 r_d^2) \end{aligned}$$

where (x, y) is the position of the projection of a point without distortion (considering the pinhole model of the camera), while (x_d, y_d) is the position of the same point when considering a real camera, and $r_d^2 = x_d^2 + y_d^2$. k_1 is the first coefficient of radial distortion.



a) b)

Figure 7.3: Original and corrected images

Summarizing the relations between discrete image coordinates in presence of distortion (i_d, j_d) and the continuous undistorted coordinates (x, y) are

$$\begin{aligned} x &= (i_d - O_x)P_x(1 + k_1 r_d^2) \\ y &= (j_d - O_y)P_y(1 + k_1 r_d^2) \end{aligned}$$

These are the equations that relate the ideal model and the real camera, by considering both image center displacement and radial distortion. We do not take into account in these equations other parameters that are less relevant. They are, for example, higher order parameters in radial distortion and the scale factor due to errors in synchronization between the raster and digitation device (see [84] for further details).

The internal parameters to be calibrated are thus: the focal distance f, the pixels dimensions P_x,P_y, the image center O_x,O_y, and the first coefficient of radial distortion k_1.

7.3.2 Internal Calibration

Internal calibration for a single camera is obtained with a very easy setting: just point the camera to a drawing containing only straight lines, without any limitation in the number and the orientation of lines and the position of the camera with respect to the drawing, and by minimizing a function measuring the distortion in the image. An example of calibration image is given in Fig. (7.3a), while in Fig. (7.3b) the result of removing lens distortion is displayed.

Among all the internal parameters of the camera, we concentrate in this sections in determining the coefficient of radial distortion k_1 and the center of the image O_x, O_y. The others (the focal distance f and the pixels dimensions P_x and P_y) in principle could be retrieved from the technical specifications of the camera, or in practice, once k_1, O_x,O_y are determined, by using simple geometric measurements.

Calibrating k_1, O_x, O_y

In order to calibrate the internal parameters of each camera, we rewrite the relations between the coordinates of the distorted and the undistorted image. The undistorted image coordinates are

$$\begin{aligned} x &= (i_d - O_x)P_x(1 + k_1 {r_d}^2) \\ y &= (j_d - O_y)P_y(1 + k_1 {r_d}^2) \end{aligned}$$

where (id, jd) are the frame coordinates of the distorted image, and $r_d^2 = ((i_d - O_x)P_x)^2 + ((j_d - O_y)P_y)^2$.

In the ideal case they would be

$$\begin{aligned} x &= (i - O_x)P_x \\ y &= (j - O_y)P_y \end{aligned} \tag{7.1}$$

Therefore the relation between distorted and undistorted frame coordinates is

$$\begin{aligned} i &= (i_d - O_x)(1 + k_1 r_d^2) + O_x \\ j &= (j_d - O_y)(1 + k_1 r_d^2) + O_y \end{aligned}$$

By knowing k_1, O_x, O_y (as well as P_x and P_y), it is possible to undistort the image so that it can be considered as coming from a pinhole camera, and the equations (7.1) are valid.

The parameters k_1, O_x, O_y are found by minimizing the distortion of a particular image taken by putting the camera in front of a drawing containing only straight lines. A measure of distortion has been obtained from the curvature of the lines in the image, that can be easily computed from the Hough Transform of the image.

The Hough Transform is a robust and effective method for finding lines fitting a set of 2D points [34], but it also provides an easy way to compute the curvature of a set of points corresponding to a line.

The transformation from (x, y) plane to (ρ, θ) plane is achieved by associating every point $P(x, y)$ with the following curve in the (ρ, θ) plan

$$\rho = xcos\theta + ysin\theta$$

At the same time a point in (ρ, θ) correspond to a line in (x, y). This representation is unique and complete as long as $0 \leq \theta < \pi$.

A graphical representation of the Hough transform (HT) can be obtained by generating a discrete grid of (ρ, θ) plan, and by defining $HT(\rho, \theta)$ as the number of points in (x, y) plane whose curve pass through (ρ, θ). The best fitting lines correspond to local maxima of $HT(\rho, \theta)$.

A standard measure for the curvature of a set of points is given by

$$\chi_r = \sum_i dist(P_i, r)^2$$

where r is the best line fitting the points, and $dist(P_i, r)$ is the distance between P_i and r. Moreover if there are many possible lines the overall curvature can be defined as the sum of the curvature for each line

$$\chi = \sum_j \chi_{r_j} = \sum_j \sum_i dist(P_{i,j}, r_j)^2$$

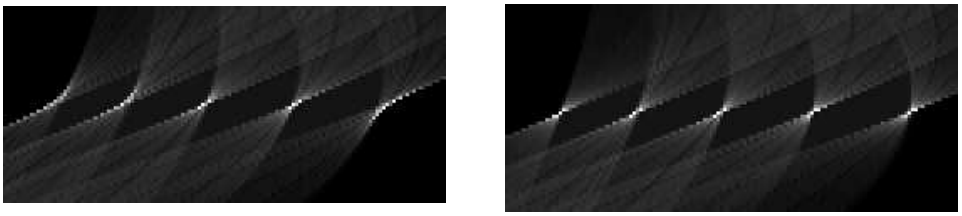
Observe that this function requires the knowledge of: (i) the equation of the best fitting lines r_j for a set of points, (ii) which are the points belonging to every line r_j. Observe that these data are unknown and that it is not trivial to find them out algorithmically.

Now given a point $P(x_P, y_P)$ and a line r in the (x, y) plane (let (ρ_r, θ_r) be the coordinates of r in the Hough domain), the distance between P and r is given by

$$dist(P, r) = |\rho_r - \rho_P(\theta_r)| = |\rho_r - (x_P cos\theta_r + y_P sin\theta_r)| \tag{7.2}$$

From equation (7.2), by using the definition of $HT(\rho, \theta)$ given before, and in the hypothesis that all the points P_i belong to a single line, it is possible to derive

$$\chi_r \simeq \sum_\rho (\rho_r - \rho)^2 HT(\rho, \theta_r)$$



a) b)

Figure 7.4: Hough Transform of a set of lines

where approximation is given by the discretization of HT.

Moreover, if the points are almost aligned, they define a cluster of high values in the HT around a maximum (the best fitting line), while other regions are close to zero. Therefore the measure χ_r can be furtherly approximated by

$$\chi_r \simeq \sum_{\rho \in \Delta_{\rho r}} (\rho_r - \rho)^2 HT(\rho, \theta_r)$$

where Δ_{ρ_r} is an appropriate interval around ρ_r.

Finally, in the case of multiple lines, the overall measure of distortion can be given by

$$\hat{\chi} = \sum_j \sum_{\rho \in \Delta_{\rho r_j}} (\rho_{r_j} - \rho)^2 HT(\rho, \theta_{r_j}) \tag{7.3}$$

which is a good approximation for χ as long as the intervals $\Delta_{\rho_{r_j}}$ are accurately chosen. In fact, this is not a real problem if the lines in the original image are not too close each other. In Figure (7.4a) a portion of the HT of the original (not corrected) image (with brighter points corresponding to higher values of HT) is shown. Notice that clusters corresponding to a line are clearly separate each other, so that it is not difficult to set appropriate intervals $\Delta_{\rho_{r_j}}$. The HT of the correspondent corrected image is in Figure (7.4b), where sharpness of the clusters denotes that lines are more straight.

The HT-based calibration method can be thus summarized in the following steps:

1. Point the camera to a drawing containing only straight lines (with no limitation in the number of lines, orientation, length, width, but it is only required that the lines are not too close each other).

2. Minimize over the internal parameters the distortion $\hat{\chi}$ obtained by

   (a) Undistorting the image with current parameters

   (b) Producing a binary image with thresholding and filtering techniques

   (c) Generating the HT of the binary image

   (d) Computing the distortion of the image by using equation (7.3)

7.3.3 External Calibration

External parameters are needed for both the correspondence problem (determining the epipolar lines for determining point correspondences), and triangulation (for reconstruction).

We choose the world reference system to be the left camera, so the parameters to be found are the translation vector T and rotation matrix R of the right camera with respect to the left one.

We assume in the following to deal with a stereo head in the standard setting (i.e. with parallel optical axes and coplanar image planes). In the ideal case the rotation matrix R will be the identity matrix. However, errors in positioning the cameras are present, so that R is not the identity matrix, but we can still assume that *small angle approximation* is valid, that is angles determining the rotation of the right camera with respect to the left one are small.

Finding the translation vector T

Usually a measure of the distance between the center of the cameras is sufficient for determining T_x (that equals the baseline, that is the distance between the two lens centers). As for T_y and T_z, we consider them to be 0. However T_x and T_y will be refined when computing ΔT_x and ΔT_y as described in the next section.

Finding the rotation matrix R

The rotation matrix R is determined by the three rotation angles around the right camera axes (ϕ, θ, and ψ). Under *small angle approximation* the rotation around X axis (ϕ) and around Y axis (θ) can be approximate as a displacement over T_y (ΔT_y) and T_x (ΔT_x) respectively.

The angle ϕ can be easily found by looking at the uniformity of the disparity map. Indeed, as the matching algorithm looks for correspondent points in the same rows of the two images, when the images are not vertically aligned ($\phi \neq 0$) there is a significant noise in the disparity map. An automatic procedure for finding out the vertical displacement ΔT_y of the two image has been realized [52]. ϕ can be easily derived from ΔT_y.

The angle θ can be derived by the Z component of the fixation point (Z_0). Z_0 can be found by imposing the disparity of a point at infinity to be zero. This can be easily obtained by looking at a far object, and by horizontally shifting one image to the other, until the disparity of that point is zero. Z_0 and θ are then computed from the horizontal displacement (ΔT_x).

Moreover, when looking at a corridor (with parallel walls), bad values in Z_0 lead to disparities that arise convergence (or divergence) of the reconstructed walls. The correct value for Z_0 is the one that allows for the reconstruction of parallel lines as walls.

Finally, the angle ψ can be again found by looking at the uniformity of the disparity map. Indeed rotation of one image with respect to the other leads to significant noise in the disparity map.

Once translations ΔT_x and ΔT_y and rotation ψ are applied to the right image in order to correctly align the pair of images, range data are computed by

$$Z = \frac{T_x f}{d'}$$

where d' is the disparity (in the correct unit) of the aligned images.

7.4 Implementation

Figure (7.5) shows the architectural schema of the implemented system. The images coming from the cameras are first warped to remove lens distortion, then images are splitted in three pairs with different resolution and only a horizontal portion of them is considered, as we are mapping range data in a 2D environment. The three pair of images are processed by the stereo algorithm that

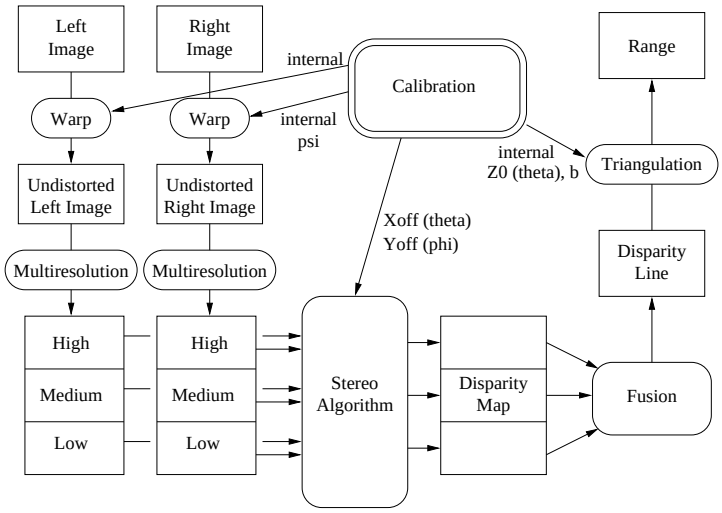


Figure 7.5: Architectural schema

returns three disparity maps. These are sampled and integrated in order to obtain a single 1D disparity line. Finally triangulation is applied for determining range information.

An off-line calibration step is performed in order to compute internal and external parameters of the stereo setting. Internal parameters are used for removing lens distortion from every image, while external parameters are used by the stereo algorithm for determining epipolar lines and in the triangulation process.

We fully implement both the calibration procedure and the stereo vision system on a Pioneer mobile robot equipped with the STH-V1 Integrated Stereo Head and a motor for controlling the direction of the camera. The overall rate of the stereo vision system, obtained with a Pentium II 233 MHz processor, is slightly above 10 Hz even with no heavy code optimization.

Accuracy of vision systems is often measured by how well it can measure the 3D world. We decide to evaluate our stereo vision system by measuring the resulting range data.

We found out that the disparities are very sensitive to internal calibration parameters, and that determining these parameters with high precision is very difficult. So, even if the calibration method we propose provides a significant reduction in range errors, usually a refinement process is required. This has been effectively achieved by minimizing the distortion (curvature) of the disparity line produced by the stereo algorithm when the stereo camera is put in front of a straight surface.

In the example shown in Figure (7.6) the camera was put in front of a door at a distance of 1.5 m. Notice in the left side (without internal calibration) that the relative error in the determination of the distance from the door was above 30% at the borders of the distorted images, while even an approximate determination (within 10%) of the internal parameters is sufficient to lower the relative error to around 5% (right side). The relative error can be further reduced by applying a refinement process in determining internal parameters, which is based on the distortion of the output of the stereo algorithm.

7.4.1 Applications

The proposed stereo vision system has been integrated within the architecture of our mobile robot "Tino". Several experiments have been performed in order to evaluate the system and we describe in this section some applications that have been relevant for augmenting the reliability of sensing

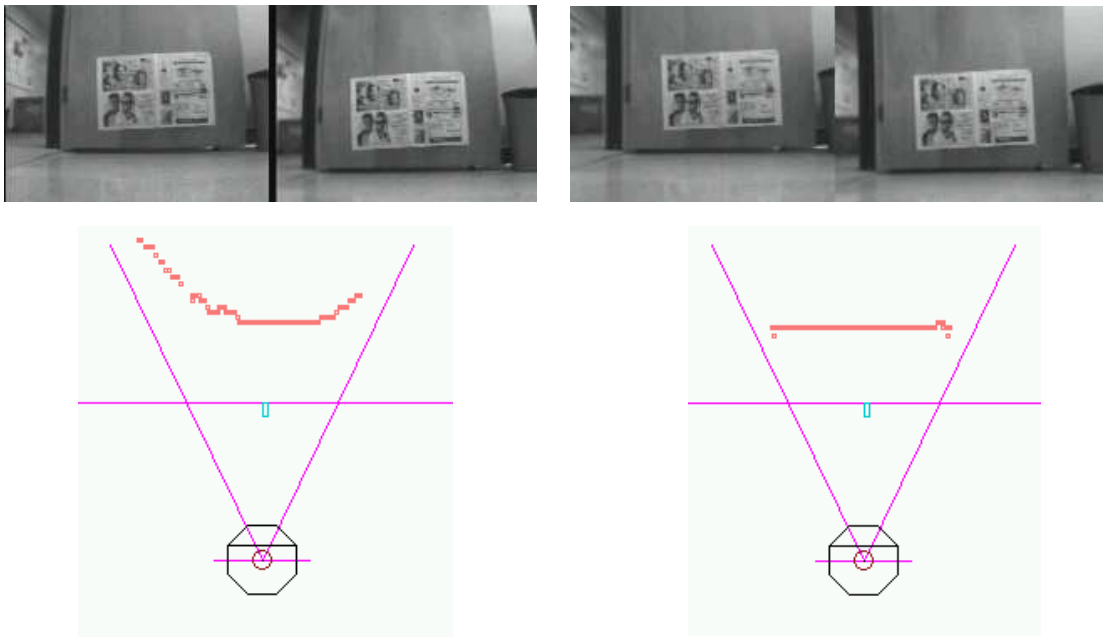


Figure 7.6: Effect of distortion in range determination

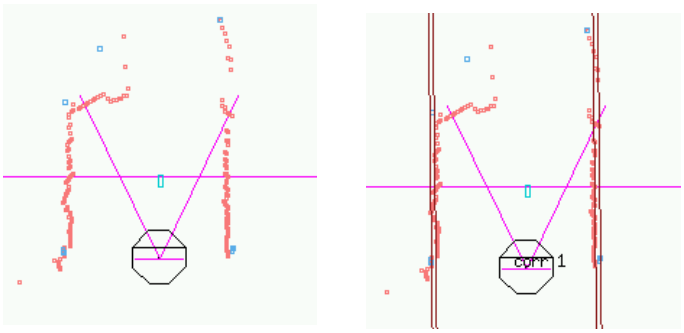


Figure 7.7: Finding out a corridor with range data

in our robot.

The first capability that is useful for acquiring range data from the environment is the generation of a panorama of the environment by rotating the stereo camera, and thus without moving the robot. Figure 7.7a shows a panorama obtained in a corridor. The quality of these range data are much better than sonar readings and moreover they are obtained without moving the robot, thus removing odometric errors. A simple procedure for determining parallel lines (which is again based on the Hough Transform) is able to determine the width and the orientation of the corridor (see Fig. 7.7b). This skill is very useful for robot self-localization.

Other experiments have been done in order to build a map of an unknown environment. Figure 7.8 shows a map produced by merging 16 panoramas taken with our stereo vision system, and correcting for odometric errors of the robot [43].

However, regarding the realization of our cognitive robot, the main advantage of using a reliable and accurate stereo vision system is that it allows for implementing robust data interpretation routines and reliable sensing actions. In particular, we have developed a routine for detecting if a

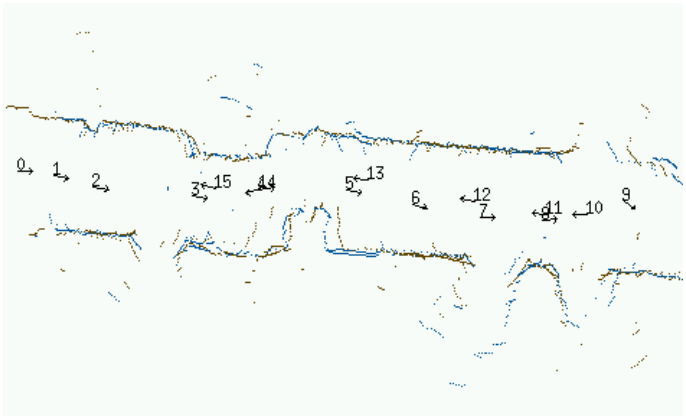


Figure 7.8: Building a map

door is open, by verifying the correspondence of range data retrieved with stereo vision with the supposed position of the doorway. This routine is much more reliable than a similar one based on sonar readings, since sonars are much more noisy, range data are sparse, and the robot is required to be exactly perpendicular to the door in order to avoid sonar reflections.

The integration of the stereo vision module within the architecture of our mobile robot "Tino", have significantly augmented its sensing capabilities and, in particular, the reliability of its sensing actions. Indeed, the rich amount of data provided by the images and the accuracy of the proposed stereo vision system allowed us for processing less noisy low-level data and thus realizing more reliable and robust high-level data interpretation routines.

Chapter 8

Describing Application Domains

In this chapter we address the application of the methodology described in Chapter 4 by following all the steps in detail. We concentrate on a mobile robot embedded in an office environment. We present a first implementation of the basic skills of the robot, then we upgrade the robot's capabilities by extending the description of the world and allowing for more complex tasks: surveillance and mail delivery.

Office-like environments and in particular mail delivery applications has been often considered as adequate experimental settings for cognitive robots, even though some capabilities are only simulated on real robots used for these experiments. (e.g. robots usually are not able to actually pick up packages).

With respect to other descriptions [57, 82] of experiments involving cognitive robots in office environments, we present the implementation of some new features.

1. The initial state is only *partially known*. This involves not only reasoning on incomplete knowledge and the generation of conditional plans, but also the implementation of reliable sensing behaviors and data interpretation routines for actually acquiring new knowledge from the environment.

2. The environment is *dynamic*. Reactivity is guaranteed by the asynchronous layered architecture and is implemented by reactive behaviors, such as obstacle avoidance.

We thus show the effectiveness of the proposed methodology as a design tool for realizing cognitive robots in real environments.

8.1 Elementary tasks

In this section we develop the design of an actual robot able to autonomously move in an office environment and to accomplish basic tasks such as reaching a position in the environment.

In our examples we make use of the map shown in Fig. 8.1, referring to an office environment constituted by corridors and rooms, divided by doors.

8.1.1 Definition of actions and conditions

For the task of navigating in an office-like environment we define moving actions for following a corridor, entering and exiting a door, sensing actions for detecting whether a door is open. The conditions to be verified are the position of the robot in the environment (such as being in a corridor, or in a room, or close to a door), and the state of the doors (open or closed). The specifications for actions and conditions are informally given in the following.

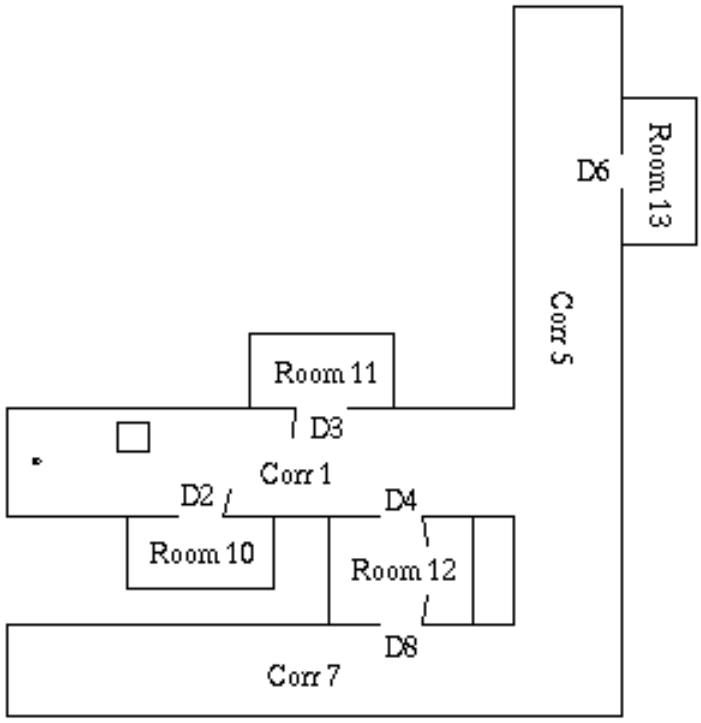


Figure 8.1: A simple environment

1. Moving actions

   - *FollowCorridor*. Follow a corridor towards a target point, that could be a door or another corridor. It is executable when the robot is in a corridor.
   - *EnterDoor*. Enter a door. It is executable when the robot is close to a door and the door is open.
   - *ExitDoor*. Exit a door. It is executable when the robot is close to a door and the door is open.

2. Sensing actions

   - *SenseOpenDoor*. Detect whether a door is open. It is executable when the robot is close to a door.

3. Conditions

   - *Corridor*, *Room*, *CloseToDoor*. Position of the robot in the environment.
   - *OpenDoor*. Door is open.
   - *ClosedDoor*. Door is closed.

Given the capabilities of the reasoning system and the fuzzy controller we are using, actions and conditions specifications given above provide for a good compromise between abstraction of high-level actions and difficulty in their implementation as control procedures. We consider this balance to be the first factor that should be determined in order to integrate high-level reasoning with low-level execution.

8.1.2 Implementation of actions and conditions

The implementation of actions and conditions deeply relates to the Saphira architecture, indeed they are often based on artifacts maintained in the Local Perceptual Space.

Every action is implemented as a Colbert activity, as described in Chapter 3. In particular, some of them have been directly implemented by goal-directed behaviors. In fact, the moving actions *FollowCorridor*, *EnterDoor*, and *ExitDoor* are implemented by Saphira built-in *FollowCorr* and *FollowDoor* behaviors.

Sensing actions instead are implemented by sensing behaviors. In particular *SenseOpenDoor* is a behavior the move the robot if front of a door and activate a data interpretation routine for sensing wheter the door is open. If interpretation fails, the behavior can move the robot to another "better" position. Notice that sensing behaviors depend on the features of the interpretation routine. For example the one based on sonar readings requires the robot to be perpendicular to the door.

Finally observe that, besides the described goal-directed behaviors, a number of reactive behaviors are always active in order to guarantee reactivity capabilities to the robot.

Conditions regarding robot position are directly derived from the Local Perceptual Space, by means of the geometric properties of artifacts: distance with respect to the robot and dimensions. The robot's position is continuously evaluated and the high-level state is updated when needed. This allows for monitoring the correct execution of plans.

The *OpenDoor* condition is activated, when needed, from the sensing behavior *SenseOpenDoor*. At the moment two interpretation routines have been tested. The first is very simple and it is based on range information provided by sonars. Because of sonar reflection problem, it requires the robot to be quite close and perpendicular to the door. Sonar readings are compared with the expected distance of a closed door and if they are significantly far away we consider the door to be open. This is not a very reliable technique since sonar readings are noisy and positioning the robot exactly in front of a door is not easy. A more reliable procedure is obtained by using the side sonars and relying on the surface construction. The sensing behavior should move the robot parallel to the door and analyse the sonar data buffer. Of course this requires a larger amount of time for sensing the state of the doors, and it could be not feasible in some situations with little space around the door.

The second interpretation routine is based on stereo vision analysis. Range data are again compared with the expected position of a closed door, however in this case data are much less noisy and it does not require the robot to be in a predefined position. Moreover we can choose to move the stereo camera instead of the robot, reducing odometric errors. In this way we obtain a much more reliable implementation of the sensing action *SenseOpenDoor*.

The process of implementing actions and conditions is concluded by a test phase aiming to prove the effectiveness of the correspondence between high-level actions and control activities, and between conditions and sensor data analysis. Specifically each activity execution and each condition verification have been tested with the robot in the environment under different conditions to evaluate reliability and robustness. This is an important task for two reasons: first it allows for detecting design errors in the first stages of the design process, second it allows for realizing robust procedures that can be easily ported to other similar environments.

8.1.3 High-level description of the environment

We can represent the environment shown in Fig. 8.1 and the robot's capabilities of acting in it through the following knowledge base:

$Corridor1 \quad \dot{\leq} \quad Corridor$

$$
\begin{array}{ll}
Corridor5 & \dot{\leq} \; Corridor \\
Corridor7 & \dot{\leq} \; Corridor \\
Room10 & \dot{\leq} \; Room \\
Room11 & \dot{\leq} \; Room \\
Room12 & \dot{\leq} \; Room \\
Room13 & \dot{\leq} \; Room \\
CloseToDoor2 & \dot{\leq} \; Corridor1 \\
CloseToDoor3 & \dot{\leq} \; Corridor1 \\
CloseToDoor4 & \dot{\leq} \; Corridor1 \\
CloseToDoor6 & \dot{\leq} \; Corridor5 \\
CloseToDoor8 & \dot{\leq} \; Corridor7
\end{array}
$$

$$
\begin{array}{lll}
Corridor1 & \mapsto & \exists_{\mathrm{K}} FollowC1ToC5 \sqcap \forall FollowC1ToC5.Corridor5 \\
Corridor5 & \mapsto & \exists_{\mathrm{K}} FollowC5ToC1 \sqcap \forall FollowC5ToC1.Corridor1 \\
Corridor7 & \mapsto & \exists_{\mathrm{K}} FollowC7ToC5 \sqcap \forall FollowC7ToC5.Corridor5 \\
Corridor5 & \mapsto & \exists_{\mathrm{K}} FollowC5ToC7 \sqcap \forall FollowC5ToC7.Corridor7 \\
Corridor1 & \mapsto & \exists_{\mathrm{K}} FollowC1ToD2 \sqcap \forall FollowC1ToD2.CloseToDoor2 \\
Corridor1 & \mapsto & \exists_{\mathrm{K}} FollowC1ToD3 \sqcap \forall FollowC1ToD3.CloseToDoor3 \\
Corridor1 & \mapsto & \exists_{\mathrm{K}} FollowC1ToD4 \sqcap \forall FollowC1ToD4.CloseToDoor4 \\
Corridor5 & \mapsto & \exists_{\mathrm{K}} FollowC5ToD6 \sqcap \forall FollowC5ToD6.CloseToDoor6 \\
Corridor7 & \mapsto & \exists_{\mathrm{K}} FollowC7ToD8 \sqcap \forall FollowC7ToD8.CloseToDoor8
\end{array}
$$

$$
\begin{array}{lll}
CloseToDoor2 \sqcap OpenDoor2 & \mapsto & \exists_{\mathrm{K}} EnterD2 \sqcap \forall EnterD2.Room10 \\
CloseToDoor3 \sqcap OpenDoor3 & \mapsto & \exists_{\mathrm{K}} EnterD3 \sqcap \forall EnterD3.Room11 \\
CloseToDoor4 \sqcap OpenDoor4 & \mapsto & \exists_{\mathrm{K}} EnterD4 \sqcap \forall EnterD4.Room12 \\
CloseToDoor6 \sqcap OpenDoor6 & \mapsto & \exists_{\mathrm{K}} EnterD6 \sqcap \forall EnterD6.Room13 \\
CloseToDoor8 \sqcap OpenDoor8 & \mapsto & \exists_{\mathrm{K}} EnterD8 \sqcap \forall EnterD8.Room12
\end{array}
$$

$$
\begin{array}{lll}
Room10 \sqcap OpenDoor2 & \mapsto & \exists_{\mathrm{K}} ExitD2 \sqcap \forall ExitD2.CloseToDoor2 \\
Room11 \sqcap OpenDoor3 & \mapsto & \exists_{\mathrm{K}} ExitD3 \sqcap \forall ExitD3.CloseToDoor3 \\
Room12 \sqcap OpenDoor4 & \mapsto & \exists_{\mathrm{K}} ExitD4 \sqcap \forall ExitD4.CloseToDoor4 \\
Room13 \sqcap OpenDoor6 & \mapsto & \exists_{\mathrm{K}} ExitD6 \sqcap \forall ExitD6.CloseToDoor6 \\
Room12 \sqcap OpenDoor8 & \mapsto & \exists_{\mathrm{K}} ExitD8 \sqcap \forall ExitD8.CloseToDoor8
\end{array}
$$

$$
\begin{array}{lll}
CloseToDoor2 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor2) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor2^{+} \sqcap \forall SenseOpenDoor2^{+}.OpenDoor2 \\
CloseToDoor2 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor2) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor2^{-} \sqcap \forall SenseOpenDoor2^{-}.ClosedDoor2 \\
CloseToDoor3 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor3) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor3^{+} \sqcap \forall SenseOpenDoor3^{+}.OpenDoor3 \\
CloseToDoor3 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor3) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor3^{-} \sqcap \forall SenseOpenDoor3^{-}.ClosedDoor3 \\
CloseToDoor4 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor4) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor4^{+} \sqcap \forall SenseOpenDoor4^{+}.OpenDoor4 \\
CloseToDoor4 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor4) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor4^{-} \sqcap \forall SenseOpenDoor4^{-}.ClosedDoor4 \\
CloseToDoor6 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor6) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor6^{+} \sqcap \forall SenseOpenDoor6^{+}.OpenDoor6 \\
CloseToDoor6 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor6) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor6^{-} \sqcap \forall SenseOpenDoor6^{-}.ClosedDoor6 \\
CloseToDoor8 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor8) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor8^{+} \sqcap \forall SenseOpenDoor8^{+}.OpenDoor8 \\
CloseToDoor8 \sqcap (\mathrm{not}_{\mathrm{K}}\ OpenDoor8) & \mapsto & \exists_{\mathrm{K}} SenseOpenDoor8^{-} \sqcap \forall SenseOpenDoor8^{-}.ClosedDoor8
\end{array}
$$

$$
\begin{array}{lll}
CloseToDoor2 & \mapsto & \forall SenseOpenDoor2^{+}.CloseToDoor2 \\
CloseToDoor2 & \mapsto & \forall SenseOpenDoor2^{-}.CloseToDoor2 \\
\ldots & & \\
OpenDoor2 & \mapsto & \forall FollowC1ToC5.OpenDoor2 \\
\ldots & &
\end{array}
$$

The above knowledge base describes the high-level knowledge of the agent about the environment and includes domain constraints, action precondition and effect axioms, and frame axioms. The designer is required to define such a knowledge by considering the features of the agent and

the environment and according to the actions and conditions defined and tested in the previous steps.

8.1.4 Description of robot's goals

Robot's goals are described by CLASSIC concepts denoting the properties that must be valid in the final state. For example to express that the final position of the robot must be Room 12, we use the concept *Room12*, that indicates the high-level position of the robot to be Room 12. Because of the domain constraints $RoomX \dot{\leq} Room$, the goal *Room* is achieved when the robot reaches any Room.

Information goals, that aim to acquire new knowledge, are also expressed by concepts. For example *OpenDoor4* requires the robot to detect if *Door4* is open (i.e. to reach a state in which *OpenDoor4* is valid). Obviously concept expressions can be used for declaring goals. For example, $Room12 \sqcap OpenDoor4$ indicates to reach $Room12$ with the knowledge that $Door4$ is open.

Given the description of the environment, the initial knowledge of the robot and its goal, the plan generation procedure is activated and a Colbert program is generated. For instance, consider the initial condition

$$Corridor1(init)$$

and the goal description

$$Room12$$

the following program is generated by the system

```
act COND_PLAN_FOR_Room12()
 {
   start FOLLOWC1D4;
   start SENSEOPEND4;
   if (SENSEOPEND4_T) {
     start ENTERD4;
   }
   else {
     start FOLLOWC1C5;
     start FOLLOWC5C7;
     start FOLLOWC7D8;
     start SENSEOPEND8;
     if (SENSEOPEND8_T) {
       start ENTERD8;
     }
     else {
       start FAIL;
     }
   }
 }
```

8.1.5 Experiments

We have performed many tests with real robots in several environments, obtaining good results as long as the environment is "friendly". Indeed the more the environment is "populated" by external objects (like obstacles, people, etc.), the more plans are difficult to execute successfully. We have successfully performed many experiments in our Department and in other Universities in Rome, and many demonstrations since the 1995 International Workshop on Description Logics.

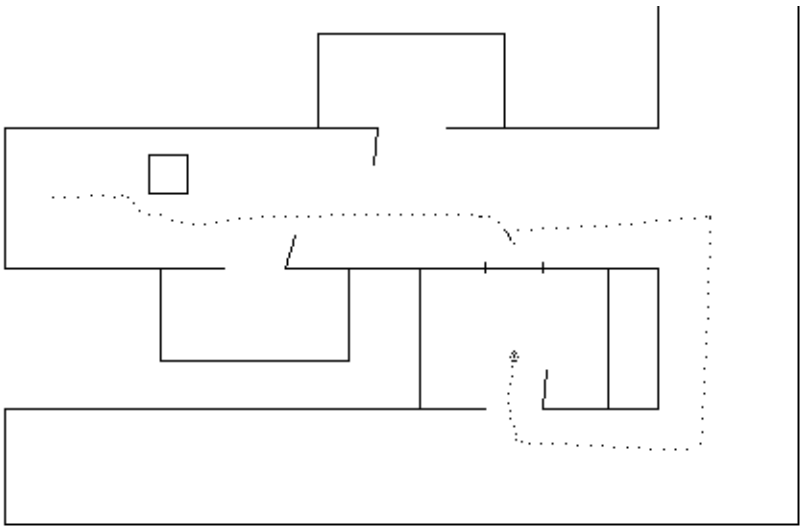


Figure 8.2: Plan execution for reaching Room12

Moreover, we sometimes make use of the Pioneer simulator for testing the robot's architecture and, in particular, for executing the robot's plans. The notable differences between using the simulator instead of the real robot are the following: (i) the environment is less dynamic, since it is possible to put in the environment unknown objects for the robot, but they cannot move; (ii) sensing procedures are much more reliable, e.g. odometric errors are very limited and sensing procedures for detecting open doors always succeed. However, the use of a simulator allows us for better analysing our methodology and above all for performing a large number of fast and cheap experiments.

We provide the robot with initial information about the position of artifacts in the environment by means of a topological map and we examine the execution of the plans generated by the reasoning system. In Figure 8.1.5 the execution of the plan `COND_PLAN_FOR_Room12` in the simulated environment is depicted. In this case data interpretation of open doors are either simulated or based on virtual sonars data provided by the Pioneer simulator. The example is interesting as it shows the basic capabilities of the robot for navigating in an office environment. Observe that control programs automatically generated by the system are actually executed by the control system. Moreover, reactivity is proved by the presence of unknown obstacles in the environment. The integration of deliberative and reactive capabilities has shown to be adequate for allowing the robot to achieve its goals.

8.2 A surveillance robot

In this section we describe a simple surveillance application for the robot. We want to generate a patrol activity in the environment described before, where the robot is in charge of repeatedly entering every room.

Suppose the robot's starting position is $Room10$. We generate the four plans corresponding to the following situations:

1. initial situation: $Room10$, goal: $Room11$,
2. initial situation: $Room11$, goal: $Room12$,
3. initial situation: $Room12$, goal: $Room13$,

4. initial situation: *Room*13, goal: *Room*10.

In this way we obtain from the system four plans: `COND_PLAN_FOR_Room11`,`COND_PLAN_FOR_Room12`, `COND_PLAN_FOR_Room13`, `COND_PLAN_FOR_Room10`, that are very similar each other. We thus show just the third one that involves the presence of two doors for exiting *Room*12.

```
act COND_PLAN_FOR_Room13()
{
  start SENSEOPEND4;
  if (SENSEOPEND4_T) {
    start EXITD4;
    start FOLLOWC1C5;
    start FOLLOWC5D6;
    start SENSEOPEND6;
    if (SENSEOPEND6_T) {
      start ENTERD6;
    }
    else {
      start FAIL;
    }
  }
  else {
    start SENSEOPEND8;
    if (SENSEOPEND8_T) {
      start EXITD8;
      start FOLLOWC7C5;
      start FOLLOWC5D6;
      start SENSEOPEND6;
      if (SENSEOPEND6_T) {
        start ENTERD6;
      }
      else {
        start FAIL;
      }
    }
    else {
      start FAIL;
    }
  }
}
```

Given the plans for navigating in the environment, that have been generated by the system, the patrol activity is defined by the user by composing in an infinite loop the four plans.

```
act PATROL()
{
  while (1) {
    start COND_PLAN_FOR_Room11;
    start COND_PLAN_FOR_Room12;
    start COND_PLAN_FOR_Room13;
    start COND_PLAN_FOR_Room10;
  }
}
```

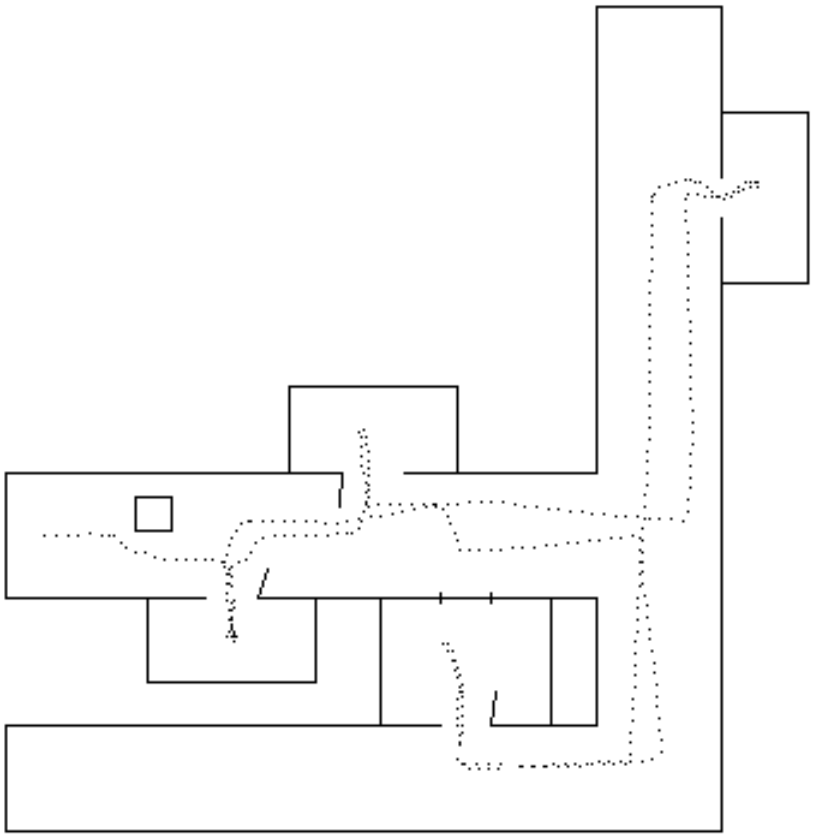


Figure 8.3: Patrol execution

The excution of the PATROL activity in the simulated environment in shown in Fig. 8.2.

Suppose now that we want the robot to check for light sources in the rooms. We can take advantage of incremental design by adding a new sensing action and a new condition and the corresponding axioms in the knowledge base defined before.

The sensing action *DetectLight* has been easily implemented by a sensing behavior that rotates the robot in the room and activates an interpretation routine for detecting lights by a threshold on the brightness of images. The condition *LightRoomXOn* denotes the presence of a light source in *RoomX* and it is verified by the execution of the sensing behavior.

The CLASSIC KB is augmented by the following axioms (one for each room X)

$$\begin{aligned} RoomX \sqcap (\text{not}_{\text{K}}\ LightRoomX) &\mapsto \exists_{\text{K}} DetectLight^{+} \\ RoomX \sqcap (\text{not}_{\text{K}}\ LightRoomX) &\mapsto \exists_{\text{K}} DetectLight^{-} \\ \top &\mapsto \forall DetectLight^{+}.LightRoomXOn \\ \top &\mapsto \forall DetectLight^{-}.LightRoomXOff \end{aligned}$$

and all the necessary frame axioms.

We want to generate a program for checking if all the lights in the environment are off Starting from Corridor1 we generate plans for the following goals:

1. $Corridor1 \sqcap LightRoom10Off$
2. $Corridor1 \sqcap LightRoom11Off$
3. $Corridor1 \sqcap LightRoom12Off$
4. $Corridor1 \sqcap LightRoom13Off$

The plan generated by the system for achieving the first goal is the following

```
act COND_PLAN_FOR_LightRoom10Off()
{
  start FOLLOWC1D2;
  start SENSEOPEND2;
  if (SENSEOPEND2_T) {
    start ENTERD2;
    start DETECTLIGHT;
    if (DETECTLIGHT_T) {
      start FAIL;
    }
    else {
      start EXITD2;
    }
  }
  else {
    start FAIL;
  }
}
```

The new patrol activity can be defined by a loop including the previous programs:

```
act LIGHTS_PATROL()
{
  while (1) {
    start COND_PLAN_FOR_LightRoom10Off;
    start COND_PLAN_FOR_LightRoom11Off;
    start COND_PLAN_FOR_LightRoom12Off;
    start COND_PLAN_FOR_LightRoom13Off;
  }
}
```

The simulated execution of this activity is very similar to the one described in the previous section. In fact, in this case, sensing actions for detecting lights are only simulated.

This patrol activity should be started by a main activity that is also responsible of activating reactive behaviors and monitoring plan failures. Indeed a failure of this plan is due to either a closed door that prevents the robot to enter a room, or the detection of a light source. By checking for the current high-level state maintained by the reasoning system is easy to provide explanations of the kind: "Door X is not open, I cannot check for light sources in Room Y" or "A light in room Y has been detected".

8.3 A mail delivery robot

A mail delivery application can also be implemented in a very simple way. Since package managing operations are only simulated, we can obtain a mail delivery implementation with a little modification of the robot's knowledge base described in the previous sections. We limit the example to a single package to be delivered and we do not consider priority among tasks.

In particular, we define new conditions denoting the position of a package in the environment, that are *PkgHeld, PkgRoom10, PkgRoom11, PkgRoom12, PkgRoom13*, and two simulated actions: *PickUp* and *PutDown*. We do not formalize any sensing actions on the package position, so we require the user to specify the initial position of the package.

The knowledge base is augmented by the following axioms:

$$
\begin{aligned}
Room &\mapsto \exists_{\mathrm{K}} PickUp \\
Room &\mapsto \exists_{\mathrm{K}} PutDown \\
Room10 \sqcap PkgRoom10 &\mapsto \forall PickUp.PkgHeld \\
Room11 \sqcap PkgRoom11 &\mapsto \forall PickUp.PkgHeld \\
Room12 \sqcap PkgRoom12 &\mapsto \forall PickUp.PkgHeld \\
Room13 \sqcap PkgRoom13 &\mapsto \forall PickUp.PkgHeld \\
Room10 \sqcap PkgHeld &\mapsto \forall PutDown.PkgRoom10 \\
Room11 \sqcap PkgHeld &\mapsto \forall PutDown.PkgRoom11 \\
Room12 \sqcap PkgHeld &\mapsto \forall PutDown.PkgRoom12 \\
Room13 \sqcap PkgHeld &\mapsto \forall PutDown.PkgRoom13
\end{aligned}
$$

We also have to add a number of frame axioms indicating that all the other conditions do not change when executing package operations.

A task for delivering a package that is *Room10* to *Room12* starting from *Corridor1* can be obtained by specifying the initial situation in the form

$$(Corridor1 \sqcap PkgRoom10)(init)$$

and the goal as

$$PkgRoom12$$

The plan generated by the system is the following.

```
act COND_PLAN_FOR_PkgRoom12()
{
  start FOLLOWC1D2;
  start SENSEOPEND2;
  if (SENSEOPEND2_T) {
    start ENTERD2;
    start PICKUP;
    start EXITD2;
    start FOLLOWC1D4;
    start SENSEOPEND4;
    if (SENSEOPEND4_T) {
      start ENTERD4;
      start PUTDOWN;
    }
    else {
      start FOLLOWC1C5;
      start FOLLOWC5C7;
      start FOLLOWC7D8;
      start SENSEOPEND8;
      if (SENSEOPEND8_T) {
        start ENTERD8;
        start PUTDOWN;
      }
      else {
        start FAIL;
      }
    }
  }
  else {
    start FAIL;
  }
}
```

Notice that all the needed knowledge about open doors is obtained during task execution. In this way it is possible to accomplish the task if at least one of the two doors leading to *Room12* is open. Moreover, if either the door for entering *Room10* (where the package is) or both of the doors in *Room12* are closed, then the plan will fail.

The execution of the plan is depicted in Fig. 8.3. Observe that the robot achieves its goal by entering the second door of *Room12* after detecting the the first one is closed. This is an interesting example because deals at the same time with conditional plan generation, reactive execution, and run-time knowledge acquisition for accomplishing a complex useful task.

The mail delivery robot has been often considered as an adequate experimental setting for cognitive robotics methods, even though real robots used for these experiments are usually not able to actually pick up packages. A main difference between the experiments we have described here and other experiments in this domain recently presented in [57, 82] is given by the formalization of sensing actions and the generation of conditional plans, that have not been addressed in the cited works. We believe that the ability of acquiring new knowledge from the environment during task execution and to generate conditional plans that take advantage of this new knowledge is an essential requirement for cognitive robots.

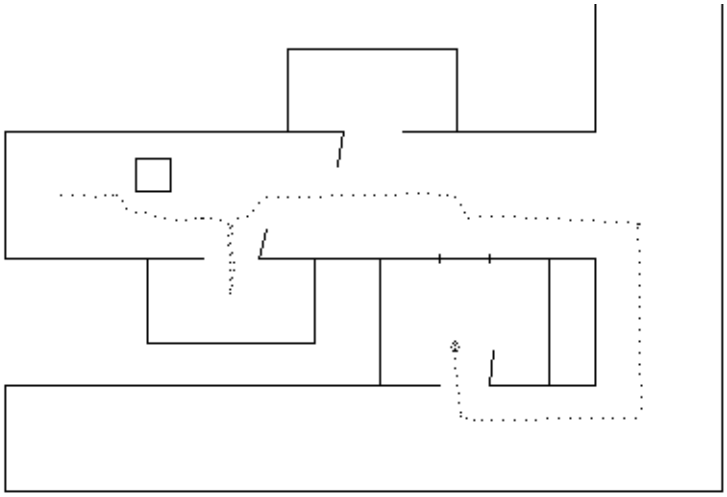


Figure 8.4: Mail delivery execution

On the other hand, we do not take into consideration here the high-level reactivity needed for managing the delivering schedule, that is managing tasks with different priorities, [57] that are also relevant in this scenario.

Chapter 9

Conclusions

The design and the development of cognitive robots that operate in a real world environment is a challenging problem for AI researchers, and a lot of recent work focuses on combining reasoning capabilities with the reactive functionalities of an actual mobile robot.

We believe that the integration of a logical reasoning framework within a robot architecture is a central problem in the design of cognitive agents. Moreover, the definition of the logical reasoning framework and the architecture for the agent should not be separated, but they should be integrated in a global design process. Such a view offers the advantage of considering at the same time and in the first stages of the development process all the issues concerning the integration of the reasoning framework within a reactive architecture. This is, in our opinion, the critical problem of Cognitive Robotics.

Following this viewpoint, in this thesis we have presented a new design methodology for the development of cognitive agents and we have shown its application on an actual mobile robot. Our methodology examines in the very first steps the interfaces between the cognitive level and the operative level. In this way the designer can individuate problems and detect design errors in the correspondences between the two levels at the early stages of the development process.

The key features presented by our methodology are:

1. *incremental design*: new capabilities can be added to the robot in a very easy way by repeating some of the proposed steps for the new features to be introduced, with minimal modification of the existing modules;

2. *reusability*: if the robot is moved to a new (similar) environment, the process of defining a new application domain involves only the last steps of the methodology, thus allowing for reusing the work done in the previous steps.

The effectiveness of the methodology has been proved by the small amount of time required for setting up the experiments that we have performed on our mobile robot. In fact, writing a new knowledge base for a new environment is an easy task; indeed we exploit the map generation features in Saphira and a semi-automatic translation from the Saphira representation of the LPS to the CLASSIC knowledge base in order to speed up this process. In this way it is possible to set up an experiment in a new environment in a few hours.

Within the realization of our cognitive robot we provide some other interesting contributions, that are summarized in the following points.

- A new heterogeneous, asynchronous, layered robot architecture that implements integration of reasoning and reactivity by representing information about the environment at different levels of abstraction.

- A new framework for representing dynamic systems that has been used for formalizing the robot's knowledge. The formalism is the basis of the CLASSIC planner we have implemented for generating plans that achieve the robot's goals.

- New multiresolution and calibration techniques for a stereo vision system that has been implemented in order to improve the sensing capabilities of the robot.

These features have played an essential role for the development of our cognitive robot and for showing the feasibility and effectiveness of our approach. Indeed we believe that, without making all the architecture components work together, it is not easy to understand what are the problems in developing real cognitive robots.

9.1 Work in progress

We are working on the application of the proposed design methodology for developing new cognitive agents in several domains. We describe in this sections two domains in which we have performed some preliminary experiments.

9.1.1 Information Access in the World Wide Web

We want to take advantage of the proposed methodology for the development of a software agent for accessing information in the World Wide Web. The goal of retrieving relevant information for the user can be achieved by analyzing Web sites and navigating through their pages for information gathering. We are working at a system [19, 27] that, starting from a description of the application domain and an information query, is able to autonomously navigate through a set of interesting Web sites (i.e. by deciding which links have to be followed) in order to retrieve the desired information.

A basic architecture has been defined [19] for implementing such an agent and some preliminary experiments have been performed in order to evaluate the effectiveness of data interpretation routines developed for acquiring knowledge from Web pages [27]. We are extending the system by adding plan generation procedures for allowing the agent to navigate through the links of a Web site and to search for the desired information.

9.1.2 The Robocup Challenge

The Robocup competition [6] is an event conceived for comparing robotic technologies in a common setting given by a soccer match between robotic teams. Our participation in Robocup 1998 [65] has been the basis for our current work on developing *cognitive robot soccer players*.

With respect to an office environment the Robocup setting is much more complex, in at least the following three aspects: (i) robots are more sophisticated, many different kinds of new sensors and actuators (e.g. color cameras, compasses, a kicker system) are needed; (ii) the environment is highly dynamic and tasks to be accomplished are much more complex; (iii) the system has to be considered in a multi-agent context, where tasks are usually accomplished by the combined behavior of several players.

We have developed a new mobile platform starting from the Pioneer base (we have called our robot soccer player "Ronaltino" see Fig. 9.1). It includes new hardware devices: a color camera, a compass, infrared sensors, a wireless network connection, a pair of independent kickers. The new sensor devices have been fundamental for recognizing and estimating the position of objects in the field. The two kickers have been used to shoot the ball in different directions: straight shots are obtained by activating both the kickers, while left or right angled shots by activating a single kicker.

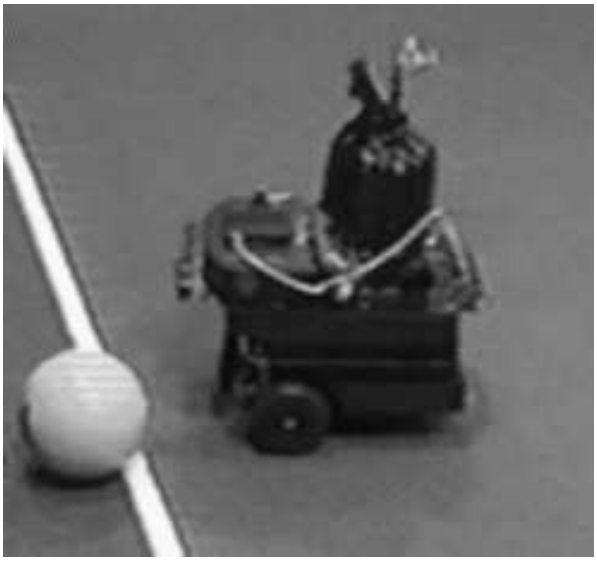


Figure 9.1: The robot soccer player Ronaltino

The main problem in realizing cognitive robot soccer players is the definition of the interface between the cognitive and the operative level. In other words, the first issue to be considered is which actions and conditions may be considered atomic at the cognitive level. In the past competition we tested the basic architecture including the management of the new hardware devices. In particular we analyzed the effectiveness and the reliability of the primitive actions and conditions.

With regard to the complexity of the environment and the goals to be achieved, we consider the use of reasoning capabilities essential. We want to program the robot soccer player in a declarative way. For example we want to write axioms like the following in order to decide which kickers to activate when attempting to score a goal.

$$\begin{aligned}
HasBall \sqcap PointToAdvGoal &\mapsto \exists_{\mathrm{K}} KickStraight \\
HasBall \sqcap PointToAdvGoal &\mapsto \exists_{\mathrm{K}} KickLeft \\
HasBall \sqcap PointToAdvGoal &\mapsto \exists_{\mathrm{K}} KickRight \\
GoalieNotInFront &\mapsto \forall KickStraight.Score \\
GoalieInFront \sqcap LeftSide &\mapsto \forall KickLeft.Score \\
GoalieInFront \sqcap RightSide &\mapsto \forall KickRight.Score
\end{aligned}$$

We are thus working on building the cognitive level of our robot soccer players.

Finally, in order to take advantage of the multi-agent scenario, we are working on devising a framework for representing not only the single robot's knowledge but also a team knowledge, and thus for reasoning about accomplishing global tasks.

9.2 Future Work

The design methodology we have presented in this thesis can be furtherly developed and extended in order to improve the actual realization of cognitive agents. To this end, we indicate here a collection of interesting points that should be addressed.

1. The application of the proposed design methodology for developing new cognitive agents in new domains will provide an important evaluation of the effectiveness of our approach. We consider this point very important for future development of actual cognitive agents.

2. We consider knowledge acquisition as a central problem in the development of cognitive agents. To this end, the designer should take advantage of a framework for programming in a declarative way the integration of information provided by different sources. Indeed we believe that high-level reasoning for information integration is a powerful and flexible tool for providing the robot with the ability of understanding complex situations.

3. In order to consider a multi-agent scenario and to build a *cognitive robotic team*, it is necessary to integrate within a distributed architecture a framework for representing and reasoning about concurrent actions and global goals.

We want to expand our work in these directions by always considering that the fundamental objective is to maintain a tight correspondence between the principled and theoretical formalization of the problems and the actual implementation and integration with the other components of the agent.

Bibliography

[1] Agre, P. E., and Chapman, D. Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)* (1987).

[2] Allen, J. Maintaining knowledge about temporal intervals. *Coomunication of the ACM 26*, 1 (1983), 832–843.

[3] Allen, J. Towards a general theory of action and time. *Artificial Intelligence 23* (1984), 123–154.

[4] Arkin, R. C. Just what is a robot architecture anyway? turing equivalency versus organizing principles. In *AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents* (1995).

[5] Artale, A., and Franconi, E. A computational account for a description logic of time and action. *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)* (1994).

[6] Asada, M. The RoboCup physical agent challenge: Goals and protocols for Phase-I. In *Robocup-97: Robot Soccer World Cup I* (1998), H. Kitano, Ed.

[7] Bacchus, F., and Kabanza, F. Using temporal logic to control search in a forward chaining planner. In *Proocedings of the Third European Workshop on Planning* (1995).

[8] Baral, C. Relating logic programming theories of actions and partial order planning. *Annals of Math and AI 21* (1997).

[9] Baral, C., Gelfond, M., and Provetti, A. Representing actions: Laws, observations and hypotesis. *Journal of Logic Programming* (1996).

[10] Barnard, S. T. Stereo matching by hierarchical, microcanonical annealing. Tech. Rep. 414, AI Center, SRI International, 1987.

[11] Blum, A. L., and Furst, M. L. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)* (1995).

[12] Borgida, A., Brachman, R. J., McGuinness, D. L., and Alperin Resnick, L. CLASSIC: A structural data model for objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1989), pp. 59–67.

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

[14] Buchheit, M., Donini, F. M., Nutt, W., and Schaerf, A. Terminological systems revisited: Terminology = schema + views. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* (Seattle, USA, 1994), pp. 199–204.

[15] Buchheit, M., Donini, F. M., and Schaerf, A. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research 1* (1993), 109–138.

[16] Buffa, M., Faugeras, O., and Zhang, Z. A stereovision-based navigation system for a mobile robot. Tech. Rep. 1895, INRIA, 1993.

[17] Bylander, T. Complexity results for planning. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)* (1991).

[18] Calvanese, D. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)* (1996), W. Wahlster, Ed., John Wiley & Sons, pp. 303–307.

[19] Calvanese, D., Giacomo, G. D., Iocchi, L., Lenzerini, M., and Nardi, D. Knowledge-based access to the Web. In *Proc. of 5th AI*IA Symposium* (1996).

[20] Cimatti, A., Giunchiglia, E., Giunchiglia, F., and Traverso, P. Planning via model checking: a decision procedure for $\mathcal{AR}$. In *Proc. of the 4th European Conference on Planning (ECP'97)* (1997).

[21] De Giacomo, G., Iocchi, L., Nardi, D., and Rosati, R. Moving a robot: the KR&R approach at work. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)* (1996).

[22] De Giacomo, G., Iocchi, L., Nardi, D., and Rosati, R. Planning with sensing for a mobile robot. In *Proc. of 4th European Conference on Planning (ECP'97)* (1997).

[23] De Giacomo, G., and Lenzerini, M. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* (1994), AAAI Press/The MIT Press, pp. 205–212.

[24] De Giacomo, G., and Lenzerini, M. Enhanced propositional dynamic logic for reasoning about concurrent actions (extended abstract). In *Working Notes of the AAAI 1995 Spring Symposium on Extending Theories of Action: Formal and Practical Applications* (1995), pp. 62–67.

[25] De Giacomo, G., and Lenzerini, M. PDL-based framework for reasoning about actions. In *Proceedings of the Fourth Conference of the Italian Association for Artificial Intelligence (AI*IA-95)* (1995), no. 992 in Lecture Notes In Artificial Intelligence, Springer-Verlag, pp. 103–114.

[26] De Giacomo, G., Lesperance, Y., and Levesque, H. J. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)* (1997).

[27] De Rosa, M., Iocchi, L., and Nardi, D. Knowledge representation techniques for information extraction on the Web. In *Proceedings of Webnet 98* (1998).

[28] Devambu, P., and Litman, D. Plan-based terminological reasoning. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)* (1991), J. Allen, R. Fikes, and E. Sandewall, Eds., Morgan Kaufmann, Los Altos, pp. 128–138.

[29] Devernay, F., and Faugeras, O. Automatic calibration and removal of distortion from scenes of structured environments. In *Proc. of SPIE'95* (1995).

[30] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. Adding epistemic operators to concept languages. In *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR-92)* (1992), Morgan Kaufmann, Los Altos, pp. 342–353.

[31] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. Queries, rules and definitions. In *Foundations of Knowledge Representation and Reasoning* (1994), Springer-Verlag.

[32] Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. Reasoning in description logics. In *Foundation of Knowledge Representation*, G. Brewka, Ed. Cambrige University Press, 1996. To appear.

[33] Donini, F. M., Nardi, D., and Rosati, R. Autoepistemic description logics. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)* (1997), pp. 136–141.

[34] Duda, R., and Hart, P. Use of the Hough Transformation to detect lines and curves in the pictures. *Communications of the ACM 15*, 1 (1972).

[35] Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., and Williamson, M. An approach to planning with incomplete information. *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR-92)* (1992).

[36] Faugeras, O., and Toscani, G. The calibration problem for stereo. In *Proc. of CVPR'86* (1986).

[37] Fikes, R., and Nilsson, N. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2* (1971).

[38] Firby, R. J. An investigation into reactive planning in complex domains. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)* (1987).

[39] Fischer, M. J., and Ladner, R. E. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences 18* (1979), 194–211.

[40] Gat, E. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)* (1992).

[41] Gelfond, M., and Lifschitz, V. Representing action and change by logic programs. *Journal of Logic Programming 17* (1993), 301–322.

[42] Green, C. Theorem proving by resolution as basis for question-answering systems. In *Machine Intelligence*, vol. 4. American Elsevier, 1969, pp. 183–205.

[43] Gutmann, J., and Schlegel, C. AMOS: Comparison of scan matching approaches for self-localization in indoor environments. In *Proc. of 1st Euromicro Workshop on Advanced Mobile Robots (EUROBOT)* (1996).

[44] Hanks, S., and Firby, R. J. Issues and architectures for planning and execution. In *Proc. of Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990).

[45] Iocchi, L., and Konolige, K. A multiresolution stereo vision system for mobile robots. In *Proceedings of the AI*IA98 Workshop on New Trends in Robotics Research* (1998).

[46] Kaelbling, L., and Rosenschein, S. Action and planning in embedded agents. *Robotics and Autonomous Systems 6* (1990), 35–48.

[47] Kanade, T., Yoshida, A., Oda, K., Kano, H., and Tanaka, M. A stereo machine for video-rate dense depth mapping and its new applications. In *Proc. of CVPR'86* (1996).

[48] Kartha, G. N. *A Mathematical Investigation of Reasoning about Actions.* PhD thesis, University of Texas at Austin, 1995.

[49] Koehler, J. An application of terminological logics to case-based reasoning. *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)* (1994).

[50] Konolige, K. Erratic competes with the big boys. *AAAI Magazine Summer* (1995), 61–67.

[51] Konolige, K. COLBERT: A language for reactive control in Saphira. In *Proceedings of German Conference on Artificial Intelligence* (1997).

[52] Konolige, K. Small vision systems: Hardware and implementation. In *Proc. of 8th International Symposium on Robotics Research* (1997).

[53] Konolige, K., Myers, K., Ruspini, E., and Saffiotti, A. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence 9*, 1 (1997), 215–235.

[54] Kozen, D., and Tiuryn, J. Logics of programs. In *Handbook of Theoretical Computer Science – Formal Models and Semantics* (1990), J. V. Leeuwen, Ed., Elsevier Science Publishers (North-Holland), Amsterdam, pp. 789–840.

[55] Lenz, R., and Tsai, R. Techniques for calibration of the scale factor and image center for high accuracy 3-d machine vision metrology. *IEEE Trans. on Pattern Analysis and Machine Intelligence 10*, 5 (1988).

[56] Lesperance, Y., Levesque, H., Lin, F., Marcu, D., Reiter, R., and Scherl, R. A logical approach to high-level robt programing. In *AAAI FAll Symposium on Control of the Physical World by Intelligent Systems* (1994).

[57] Lesperance, Y., Tam, K., and Jenkin, M. Reactivity in a logic-based robot programming framework. In *Proceedings of AAAI98 Fall Symposium on Cognitive Robotics* (1998).

[58] Levesque, H. J. What is planning in presence of sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)* (1996), A. P. M. Press, Ed., pp. 1139–1149.

[59] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31 (1997), 59–84.

[60] Lifschitz, V. Minimal belief and negation as failure. *Artificial Intelligence Journal 70* (1994).

[61] Lobo, J. COPLAS: A COnditional PLAnner with Sensing actions. In *Proceedings of AAAI98 Fall Symposium on Cognitive Robotics* (1998).

[62] Lobo, J., Mendez, G., and Taylor, S. Adding knowledge to the action description language $\mathcal{A}$. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)* (1997), pp. 454–459.

[63] McAllester, D., 1991. Unpublished Manuscript.

[64] McCarthy, J., and Hayes, P. Some philisophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4* (1969), 463–502.

[65] Nardi, D., Clemente, G., and Pagello, E. ART: Azzurra Robot Team. In *Proceedings of 2nd RoboCup Workshop* (1998).

[66] Nebel, B. Terminological cycles: Semantics and computational properties. In *Principles of Semantic Networks*, J. F. Sowa, Ed. Morgan Kaufmann, Los Altos, 1991, pp. 331–361.

[67] Nilsson, N. J. Shakey the robot. Tech. Rep. 323, SRI Artificial Intelligence Center, 1984.

[68] Penberthy, J., and Weld, D. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR-92)* (1992).

[69] Piaggio, M. Classifying robot software architecture. *AI*IA Notizie 4* (1998).

[70] Quam, L. H. Hierarchical warp stereo. Tech. Rep. 402, AI Center, SRI International, 1986.

[71] Reiter, R. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press, 1991, pp. 359–380.

[72] Reiter, R. Proving properties of states in the situation calculus. *Artificial Intelligence 64* (1993), 337–351.

[73] Reiter, R. *Knowledge in action: Logical foundations for describing and implementing dynamical systems*. 1998. Draft monograph available at `http://www.cs.toronto.edu/~cogrobo`.

[74] Rosenschein, S. J. Plan synthesis: A logical perspective. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)* (1981), pp. 331–337.

[75] Saffiotti, A., Konolige, K., and Ruspini, E. A multivalued logic approach to integrating planning and control. *Artificial Intelligence 76* (1995), 481–526.

[76] Sandewall, E., and Shoham, Y. Non-monotonic temporal reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming* (1995), vol. 4, Oxford Science Publications, pp. 439–498.

[77] Scherl, R., and Levesque, H. J. The frame problem and knowledge producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)* (1993), pp. 689–695.

[78] Schild, K. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)* (Sydney, 1991), pp. 466–471.

[79] Schmidt-Schauss, M., and Smolka, G. Attributive concept descriptions with complements. *Artificial Intelligence 48*, 1 (1991), 1–26.

[80] Schoppers, M. Universal plan for reactive robots in unpredictable environments. In *Proc. of the Int. Joint Conf. on Artificial Intelligence* (1987).

[81] Shanahan, M. Event calculus planning revisited. In *Proc. of the 4th European Conference on Planning (ECP'97)* (1997).

[82] Shanahan, M. Reinventing Shakey. In *Proceedings of AAAI98 Fall Symposium on Cognitive Robotics* (1998).

[83] Simmons, R. An architecture for coordinating planning, sensing and action. *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)* (1992).

[84] Tsai, R. An efficient and accurate camera calibration technique for 3D machine vision. In *Proc. of CVPR'86* (1986).

[85] Tucakov, V., Sahota, M., Murray, D., Mackworth, A., Little, J., Kingdon, S., Jeggins, C., and Barman, R. Spinoza: A stereoscopic visually guided mobile robot. In *Proc. of HICSS'97* (1997).

[86] Woods, W. A., and Schmolze, J. G. The KL-ONE family. Tech. Rep. TR-20-90, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1990. To appear in *Computers & Mathematics with Applications.*

[87] Wooldridge, M. J., and Jennings, N. R. Agent theories, architecrures, and languages: A survey. In *Intelligent Agents ECAI-94 Workshop on Agent Theories, Architectures, and Languages* (1994).

[88] Wooldridge, M. J., and Jennings, N. R. Intelligent agents: Theory and practice. *Knowledge Engineering Review 2*, 10 (1995), 115–152.

Università *La Sapienza*
Dottorato di Ricerca in Ingegneria Informatica
Collana delle tesi
Collection of Theses

V-93-1 Marco Cadoli. *Two Methods for Tractable Reasoning in Artificial Intelligence: Language Restriction and Theory Approximation.* June 1993.

V-93-2 Fabrizio d'Amore. *Algorithms and Data Structures for Partitioning and Management of Sets of Hyperrectangles.* June 1993.

V-93-3 Miriam Di Ianni. *On the complexity of flow control problems in Store-and-Forward networks.* June 1993.

V-93-4 Carla Limongelli. *The Integration of Symbolic and Numeric Computation by p-adic Construction Methods.* June 1993.

V-93-5 Annalisa Massini. *High efficiency self-routing interconnection networks.* June 1993.

V-93-6 Paola Vocca. *Space-time trade-offs in directed graphs reachability problem.* June 1993.

VI-94-1 Roberto Baldoni. *Mutual Exclusion in Distributed Systems.* June 1994.

VI-94-2 Andrea Clementi. *On the Complexity of Cellular Automata.* June 1994.

VI-94-3 Paolo Giulio Franciosa. *Adaptive Spatial Data Handling.* June 1994.

VI-94-4 Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* June 1994.

VI-94-5 Andrea Sterbini. *2-Thresholdness and its Implications: from the Synchronization with PVchunk to the Ibaraki-Peled Conjecture.* June 1994.

VII-95-1 Piera Barcaccia. *On the Complexity of Some Time Slot Assignment Problems in Switching Systems.* June 1995.

VII-95-2 Michele Boreale. *Process Algebraic Theories for Mobile Systems.* June 1995.

VII-95-3 Antonella Cresti. *Unconditionally Secure Key Distribution Protocols.* June 1995.

VII-95-4 Vincenzo Ferrucci. *Dimension-Independent Solid Modeling.* June 1995.

VII-95-5 Esteban Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging.* June 1995.

VII-95-6 Michele Flammini. *Compact Routing Models: Some Complexity Results and Extensions.* June 1995.

VII-95-7 Giuseppe Liotta. *Computing Proximity Drawings of Graphs.* June 1995.

VIII-96-1 Luca Cabibbo. *Querying and Updating Complex-Object Databases.* May 1996.

VIII-96-2 Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* May 1996.

VIII-96-3 Marco Cesati. *Structural Aspects of Parameterized Complexity.* May 1996.

VIII-96-4 Flavio Corradini. *Space, Time and Nondeterminism in Process Algebras.* May 1996.

VIII-96-5 Stefano Leonardi. *On-line Resource Management with Application to Routing and Scheduling.* May 1996.

VIII-96-6 Rosario Pugliese. *Semantic Theories for Asynchronous Languages.* May 1996.

IX-97-1 Paola Alimonti. *Local search and approximability of MAX SNP problems.* May 1997.

IX-97-2 Tiziana Calamoneri. *Does Cubicity Help to Solve Problems?.* May 1997.

IX-97-3 Paolo Di Blasio. *A Calculus for Concurrent Objects: Design and Control Flow Analysis.* May 1997.

IX-97-4 Bruno Errico. *Intelligent Agents and User Modelling.* May 1997.

IX-97-5 Roberta Mancini. *Modelling Interactive Computing by exploiting the Undo.* May 1997.

IX-97-6 Riccardo Rosati. *Autoepistemic Description Logics.* May 1997.

IX-97-7 Luca Trevisan. *Reductions and (Non-)Approximability.* May 1997.

X-98-1 Gianluca Battaglini. *Analysis of Manufacturing Yield Evaluation of VLSI/WSI Systems: Methods and Methodologies.* April 1998.

X-98-2 Piergiorgio Bertoli. *Using OMRS in practice: a case study with Acl-2.* April 1998.

X-98-3 Chiara Ghidini. *A semantics for contextual reasoning: theory and two relevant applications.* April 1998.

X-98-4 Roberto Giaccio. *Visiting complex structures.* April 1998.

X-98-5 Giampaolo Greco. *Dimension and structure in Combinatorics.* April 1998.

X-98-6 Paolo Liberatore. *Compilation of intractable problems and its application to artificial intelligence.* April 1998.

X-98-7 Fabio Massacci. *Efficient approximate tableaux and an application to computer security.* April 1998.

X-98-8 Chiara Petrioli. *Energy-Conserving Protocols for Wireless Communications.* April 1998.

X-98-9 Giulio Balestreri. *Algebraic Semantics of Shared Spaces Coordination Languages.* April 1999.

XI-99-1 Luca Becchetti. *Efficient Resource Management in High Bandwidth Networks.* April 1999.

XI-99-2 Nicola Cancedda. *Text Generation from Message Understanding Conference Templates.* April 1999.

XI-99-3 Luca Iocchi. *Design and Development of Cognitive Robots.* April 1999.

XI-99-4 Francesco Quaglia. *Consistent checkpointing in distributed computations: theoretical results and protocols.* April 1999.

XI-99-5 Milton Romero. *Disparity/Motion Estimation For Stereoscopic Video Processing.* April 1999.