Priorities and data abstraction in hierarchical control architectures for autonomous robots

Adelardo A. D. Medeiros
Universidade Federal do Rio Grande do Norte
Departamento de Engenharia Elétrica
59072-970 Natal RN Brazil
adelardo@leca.ufrn.br

Raja Chatila
LAAS/CNRS
7, Ave. du Colonel Roche
31077 Toulouse Cedex France
raja@laas.fr

Resumo

Para robôs com um largo grau de autonomia, bons resultados podem ser obtidos adotando-se um modelo hierárquico para a arquitetura de controle. Esse artigo expõe nossa definição de arquiteturas hierarquisadas e uma estratégia eficiente para a implementação em tempo real de dois aspectos importantes nesse tipo de sistema: a execução de uma política inteligente de resolução de conflitos e a abstração progressiva dos dados. Mostra-se como a introdução de um executivo *no sistema de controle permite tratar de forma adequada esses aspectos e apresenta-se alguns resultados experimentais obtidos com o robô móvel* Hilare II.

Abstract

In the context of autonomous robots, good results can be obtained using a hierarchical model for the control architecture. This paper exposes our definition of an hierarchical architecture and an efficient approach for the real-time implementation of two important aspects of this kind of system: the execution of an intelligent policy for resolving conflicts and the progressive abstraction of data. We show how an executive*, introduced in the control system, can adequately accomplish these tasks and we present some experimental results obtained with the* Hilare II *mobile robot.*

1 Introduction

The present state of computing technology requires complex robots to be implemented as concurrent systems. The existing research experience has not yet produced an ultimate paradigm for the distribution and/or coordination of the functionalities required for intelligent robotic systems subject to real-time constraints. The main reason, in the context of programmable autonomous mobile robots, is that a software control architecture has to fulfill several objectives:

Reactivity – The robot should be capable to take into account external events with time bounds that are compatible with the correct, efficient and safe execution of its tasks.

Intelligent behavior – The reactions of the robot to external stimuli must be guided by the objectives of its main task.

Robustness – The architecture should be able to exploit the redundancy of the processing functions and the limited accuracy, reliability and applicability of individual sensors must be compensated by the integration of several complementary sensors.

Programmability – A useful robot cannot be designed for only one precise task. It should be able to achieve multiple tasks which are described at some abstraction level. The functions should be easily combined according to the task to be executed.

Autonomy and adaptability – The robot is designed to accomplish its tasks by itself, despite of possibly changing external circumstances. This includes the capacity to refine the description of a task to adapt it to the actual conditions of execution.

These behavior specifications have some implications on the design requirements:

Modularity – As required for all complex systems, the control architecture of autonomous vehicles should be divided into smaller subsystems that can be designed and implemented separately.

Flexibility – Experimental robotics requires continuous changes in the design during the implementation. Therefore, flexible control structures are required to allow the design to be guided by the demonstrated success or failures.

Expandability – Since a long time is required to design, build and test the individual components of the control system, the architecture should have the capacity of integrating new functions, in order to be able to build the control system incrementally.

2 Main approaches

The well known sense-model-plan-act paradigm has been used as the basis of many traditional robot control architectures since the early works in mobile robotics. The main common distinctive feature in these *strictly deliberative* systems is that the control problem is divided into progressive levels of abstraction. Data flows from the sensors to a series of perception processes, to a decision-making process, through the series of action processes to the actuators. One application of this concept was the NASREM [1] architecture (fig. 1).

The primary drawback of this approach is that the series of stages through which all sensor data must pass places an unavoidable delay in the loop between sensing

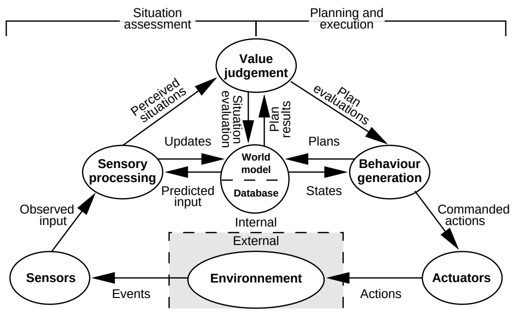

Figure 1: The NASREM architecture

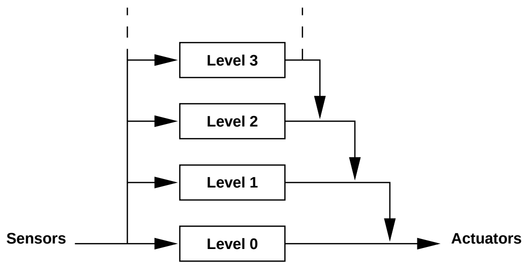

Figure 2: The subsumption architecture

and action. Unless some component is bypassed at times, thereby undermining the generality of the approach, the delay can only be reduced by increasing the throughput of the modules. At times it may occur that specialized circumstances or goals provide sufficient constraints to allow direct disambiguation of sensor data for control, but the specialized functions needed to make this possible may not be easily added if the architecture has not already been organized to accommodate them.

Recently, the use of behavioral-based control systems, inspired by the subsumption [2] architecture (fig. 2), has received a great deal of attention and has attracted many supporters as well as detractors. The success of this *strictly reactive* approach has been because many tasks can be implemented with a collection of simple, interacting behaviors. Each behavior can be computed in parallel with the other behaviors and extract from the environment just the information needed to perform its task.

Brooks has stated [3] that this style of computation can break the Von Neuman bottleneck that throttles computation of information-rich sensory data. He feels that his approach is scalable to the level of higher cognitive functions. But is it unequivocal that this approach is truly scalable? Tsotos [14] argues, with a counter-example, that the *strict* computational behaviorist position for the modeling of intelligence does not scale to human-like problems and performance.

This is accomplished by showing that the task of visual search can be viewed within the behaviorist framework and that the ability to search images (or any other sensory field) of the word to find stimuli on which to act is a necessary component of any behaving, intelligent agent. If targets are not explicitly known and used to help optimize search[1], he shows that the search problem is NP-hard. Experimental results seem to agree with this opinion, since most of the works with strictly behavioral robots [9, 4] are not based on complex sensors and do not go beyond navigation tasks (like avoiding obstacles and collisions).

Many authors [6, 7, 12, 13] advocate the combination of the deliberative and reactive approaches, combining the task of analyzing data from all sensors and constructing a composite view of the world with the immediacy needed for real-time response. Hence, the control architecture must include both decision-making and real-time execution processes, which correspond respectively to the programmable part and to the reactive part of the system.

[1]Knowledge of the target is explicitly forbidden in the strict interpretation of the published behaviorist dogma.

3 Hierarchical architectures

The hierarchical architectures[2], presented in figure 3, are designed to overcome problems inherent in strictly deliberative or reactive architectures. The reactive subsystems, which runs continuously, parallel to, and independent from the deliberative subsystem, can respond quickly to outside events (such as approaching obstacles). And a global high-level reasoning system ensures an intelligent overall behavior. Such an organization decouples the assimilation and reasoning actions of the deliberative agent from the low-level responses of the reactive agents.

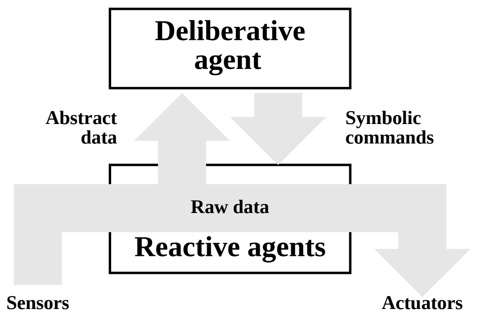

Figure 3: The hierarchical architecture

The strategic decision-making agent must have access to all the available high-level knowledge in order to reason about the global state of the environment. Furthermore, the intelligent control unit must be independent of the other system components' organization and interactions. It should act more as an intelligent symbolic information processor rather than as a command generator. This advocates that the generation of the required tactical control actions should be delegated to the activities. Only the output of one such activity is selected, according to the control strategy adopted by the decision element, and is routed to the actuators to physically drive some behavior of the robot.

To ensure system reactivity and responsiveness to changes in the real-time environment, the information traffic between the perception/action modules and the reasoning agent should be kept to a minimum in order to avoid communication delays. This suggests that the exchanged information should be of a high level of abstraction, that is, processed data as opposed to raw sensory information. This requires the perception modules to process the raw data and to extract the relevant environment features. These "feature extractors" should provide the high-level information in real-time to guarantee that the control decisions are based on the most recent knowledge and to ensure the reactivity.

3.1 The LAAS architecture

The LAAS architecture, adopted in this work, has been used in several real-time applications of autonomous mobile robotics. It decomposes the robot's control system of a robot into two levels:

- a *functional level*, which includes processing functions and control loops; and

[2]What we define as an *hierarchical architecture* is sometimes called an *hybrid architecture*, because some authors associate the term hierarchical with what we call a strictly deliberative architecture.

- a *decisional level*, possibly organized in layers, embedding the capacities of planning and decision-making. Since planning requires an amount of time usually longer than the dynamics imposed by the environment, each layer is composed of a *planner*, which produces the sequence of actions necessary to reach a given objective, and a *supervisor*, which controls the execution of the plans and reacts to incoming events in a bounded time.

One of the most generic implementations of this principle organizes the global system architecture into three levels, representing two decisional layers and the functional level (fig. 4). The decomposition of the decisional level into two layers is heuristic, in order to improve performance.

Activities on the mission level correspond to long term strategic planning as well as eventual plan adaptations. This layer relies on very abstract knowledge and sophisticated reasoning techniques. It is the typical domain of AI planners, able to make use of extensive application domain knowledge. An instance of the mission level, based on the temporal planning system IxTeT (Indexed Time Table), is presented in [5].

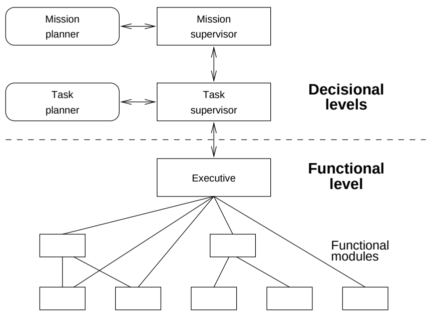

Figure 4: The LAAS architecture

The task level uses procedures, or plan templates, for task decomposition. Procedures are pre-written ordered sets of actions, rich in structure (disjunction and conjunction of actions, recursions, real-time choices and parallelism). The task level receives tasks from the mission level, chooses the appropriated sequence of actions and supervises their execution. As it is based on a context dependent selection of predefined procedures (no general planners are used), its response time can be bound. In [8] is presented an implementation of a task level using C-PRS, a well suited set of tools for real-time systems, based on procedural reasoning.

The functional level includes the basic functionalities of the robot. It is composed of an executive and a set of modules embedding the functions for sensing and acting. The instantiation of the task level procedures finally leads to activation and termination of activities in the modules, which involves monitoring and recognition of termination conditions and execution failures. These tasks are accomplished by the executive [10].

A module usually encapsulates similar functions or functions that manipulate the same data or resources. The modules react to external requests from a client and, after processing, send back a report to the client, using a client/server communication protocol (with requests and replies). If necessary, while executing a function, a module can dispatch requests to other modules. Hence, the organization of the modules is not fixed: their interactions are dynamic and depend on the task being executed and on the perceived environment [7, 11].

The functions of a given module are pre-defined at the system design stage. Some functions permit several instances in parallel, but certain other ones can not be executed simultaneously, mainly because they use the same non-sharable resource.

One important feature of this organization is its flexibility. The robot behavior is not fixed by any association of behaviors: all the functions are available and can be combined to achieve a given task or a desired activity.

4 Abstraction and conflicts

The LAAS architecture combines deliberation and reactivity to endow the robot with an autonomous and intelligent behavior. This combination, however, requires the definition of two supplementary aspects, not present in some other architectures:

- a policy to resolve conflicts; and
- a strategy to abstract low-level data.

4.1 Priorities policy

To guarantee the reactivity of the robot, the low-level activities in the modules run in parallel with the high-level planning. In consequence, a new plan can decide to start activities that are not compatible for simultaneous execution with some other previous activities. In this case:

- starting the new activity can be refused or delayed; or
- the previous activity can be interrupted.

The definition of an intelligent priority policy is not trivial, because who can correctly arbitrate the conflicts (the decisional level) is not fast enough to do it in real-time. Hence, some possibilities must be excluded, because they induce the robot to manifest some undesirable behaviors:

- The priorities could be hard-coded, associating with each function a strategy (refuse, delay or interrupt) *vis-à-vis* each other one. This policy is not adequate because, as the functional level does not know the actual plans, the robot behavior would not be intelligently guided by the global task. For example, two navigation activities are obviously incompatible, but to determine the must important we have to know the general mission of the robot.
- The decisional level could maintain a list of activities in execution and use this information to plan and, if necessary, interrupt the conflicting activities. This strategy can not be used because the temporal characteristics of the decisional level are not fast enough to follow the real-time execution of the functions, and the reactivity of the robot would be compromised.

4.2 Data abstraction

Our ambition is to design programmable robots, and not only machines conceived to accomplish an *a priori* known task. This implies that the functions in the modules must be generic, with enough input parameters to be used in different contexts.

The desired final position to reach, for example, must be an input parameter of a locomotion function. The global mission of the robot can require several movements, executed by the same locomotion function with different input parameters.

The problem is that the decisional level, which orders these movements, only works with abstract data. Thus, somewhere we have to convert tasks like `(GOTO (CENTER ROOM))` or `(GOTO (DOOR ROOM HALL))`, which are so different from the point of view of the planner, to calls to the same locomotion function. This mapping, and also the real-time implementation of the priority policy, will be done introducing the concept of *services*.

5 The services

Services are the primitives of the communication protocol between the decisional and functional levels (fig. 5). The executive converts the requests and replies of the decisional level (concerning services) to requests and replies recognized by the modules (concerning services).

The executive contains a static list of all the valid services, described by:

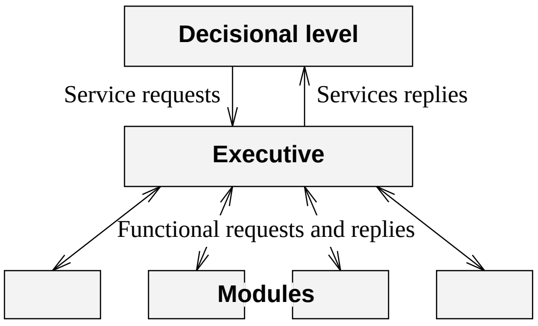

Figure 5: The communication protocols

1. the corresponding function;
2. the input and output parameters, which are local executive's variables;
3. its behavior when in conflict with each one of the other services.

More than one service can use the same function. This arrives when changing an input parameter of a function alters the meaning of the corresponding services to the decisional level. For example, `GOTO_DOOR` and `GOTO_GOAL` are two services that points to the same locomotion function.

The same function can also be called by different services to resolve conflicts. The executive offers services with different priorities, and the decisional level chooses one of them according to the actual global task, as explained in section 5.2.

5.1 Services and data abstraction

The services allow the functions of the modules to be generic without overloading the communication channel between the functional and decisional level with numerical data. This is possible because the parameters associated with the services transfer the required informations.

For example, a call to the `GOTO_DOOR` service must be preceded by a call to some other service which determines the door position, like `FIND_DOOR`. In this case, the system will work correctly because the same executive's variable is an output parameter of `FIND_DOOR` and an input parameter of `GOTO_DOOR`.

As the services do not have input parameters, we could think the programmability of the robot would be compromised, as the services depend on the actual mission.

This is not right because the services are only interfaces, and the underlying functions maintain all their generality. Changing the global mission of a robot always requires some adjustments, but this approach reduces the modifications to a minimum. Adding a new service only demands the inclusion of a new element into the list of services, without altering the modules. This good evolutivity was verified during the implementation of the experience described in section 7

5.2 Services and priorities

To illustrate the role of the services in the resolution of conflicts, we consider the example of a mobile robot with a manipulator arm. There are three possibilities[3]:

1. the arm can be used while the robot is moving (there is no conflict).
2. the arm cannot be used while the robot is moving:
    (a) the movement must be interrupted.
    (b) the utilization of the arm must wait the end of the movement.

These possibilities can be implemented by three different services: the first one (`USE_ARM_1`) is not incompatible with robot movements, the second one (`USE_ARM_2`) interrupts all the conflicting services and the third one (`USE_ARM_3`) waits the end of eventual movements.

The most important feature of this strategy is that the decisional level only decides the priority policy to use, but does not execute it. The planner decides, for example, that using the arm in a certain context is more important than an eventual movement of the robot, and inserts an `USE_ARM_2` into the plan. During execution, the task level sends the `USE_ARM_2` request to the executive. If some of the running services occasions a movement of the robot, it will be interrupted before starting the new service. Hence, the intelligent choice of the priority policy can be done off-line, using the knowledge of the global task, and efficiently executed on-line by the executive.

The executive needs to know all the incompatibilities between services. If this knowledge is coded in a procedural way, the expandability of the system would be compromised, because to add a new service we have to explicitly include its incompatibilities with all the previous ones. This task becomes difficult and subject to errors when the number of services increases.

The incompatibilities between services can be more easily expressed using a rule-based system. With `IF–THEN` rules, adding new services is not difficult, but many rule-based environments have two main drawbacks which prevent their utilization:

- The position occupied by the executive in the control architecture imposes a bounded (and small) execution time, impossible to obtain with most of the rule-based developing systems.

[3]A fourth possibility is to refuse the request to use the arm if the robot is moving. We do not consider this possibility because of the reactivity paradigm adopted in the LAAS architecture [11]. But, if desired, it can be included in other hierarchical architectures.

- Usually, there is no formal verification of the rule basis, and we could not be sure that all the incompatibilities between systems are recognized.

One possible solution to overcome such limitations is the use of Kheops.

6 The KHEOPS system

The Kheops system is a development environment for rule-based systems devoted to real-time applications. Its knowledge representation relies on multi-valued propositional logic and monotonic reasoning. Kheops tackles the usual poor temporal properties of rule-based declarative programming by pre-compiling the knowledge. It rewrites a set of declarative rules into a deterministic network, the traversal of which gives an upper-bound on the time response of the system.

Kheops relies on a finite set of propositional input and output variables (or *attributes*) which take their values over finite domains and on a knowledge base which defines a consistent mapping from the input space to the output space. The compiler produces a decision tree, the nodes of which are tests on the input attributes values, and the leaves of which characterize the resulting situation (complete determination of the output attributes, incompleteness or inconsistency).

Kheops permits to ensure some important properties of the executive:

- The execution cycle of the executive will have a finite (and *a priori* measurable) execution time, because of the characteristics of the decision tree.
- As the compilation is made during the developing phase, the knowledge base can be refined until we are assured the output attributes are completely and unequivocally determined to all possible values of the input attributes. This assures that the conflict resolver provides one and only one solution for each possible combination of services.
- The compilation converts the knowledge base into an equivalent C source that can be compiled and linked to the other subsystems. No trace of Kheops (like inference machines, usually necessary when executing rule-based programs) will be present in the executable version of the executive.

Figure 6 shows an example of Kheops rule, selected if an `USE_ARM_2` request arrives when the robot's locomotion module is executing a circular trajectory.

```
Takeimage1WhenLocostateCircling:
   function == "USE_ARM_2", operation == "START",
   @armstate == "IDLE",    @locostate == "CIRCLING"
==>
   do sendrequest("LOCO","STOP"), do setexecstate("LOCOCIRCLE","ZOMBIE"),
   restrict locostate == "IDLE",  restrict armstate == "ARM_USING",
   restrict status == "EXEC"
```

Figure 6: A rule of the executive

7 Experimental results

The experience was carried out with the robot Hilare II (fig. 7). This experimental robot is equipped with a belt of 32 ultrasound sensors, 2 driving wheels, 2 odometer wheels and a rotating platform, supporting 2 color cameras and a 3D scanning laser range-finder.

In our experience, the robot explores an environment, looking for some objects. As soon as an object is detected, the robot must go toward its position, take it and put it on the robot itself. This cycle continues until a given number of objects is found. A simple but efficient approach to this problem is given by the parallel algorithm presented in figure 8.

Paralleling the activities optimizes the utilization of resources but creates potential conflicts that must be resolved by the executive:

- Simultaneous execution of two trajectories is of course impossible. As the ultimate robot's goal is to take the objects, the activities of the "taking object" cycle must have priority against the activities of the other two cycles (see fig. 8).
- To take an object (but not to put it) requires the immobility of the robot.

The algorithm was implemented dividing the environment in cells and calculating the probability to find an object in each cell. Initially, the probability is identical for all cells. The camera continuously obtains images around the robot and the probability of the analyzed cells, if an object is not found, is reduced according to the distance and angle between the cell and the camera (the probability of detecting an object is greater for near and directly sighted cells). To explore the environment, the robot goes to the cell where, obtaining images all around this position, the maximal reduction of the probabilities summation is obtained.

As there is no high-level planning in this experience, the mission level is not necessary. The three cycles of the parallel algorithm correspond to three parallel procedures in the task level, implemented with PRS. Eight modules (fig. 9) are used:

PLAN: computes free paths in cluttered environments, taking into account the non-holonomic constraints of the robot.

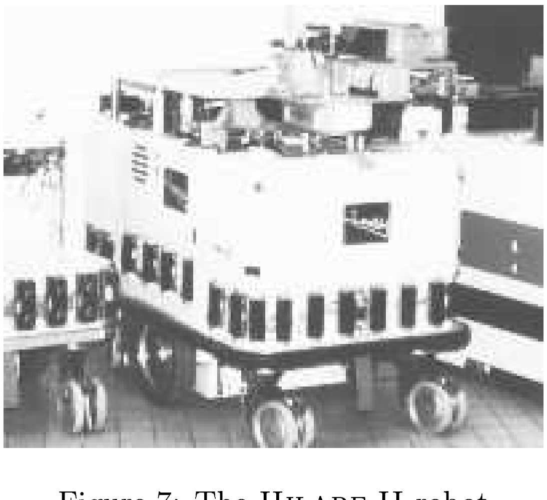
Figure 7: The Hilare II robot

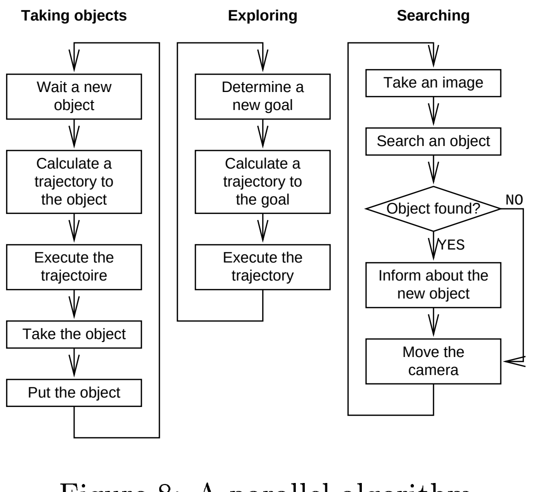

Figure 8: A parallel algorithm

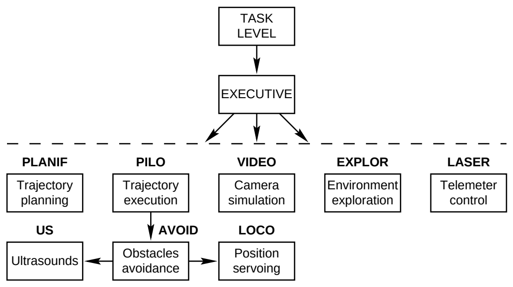

Figure 9: The implemented architecture

Service	Incompatibilities	Action
EXEC-TRAJ-GOAL	CALC-TRAJ-OBJ	Wait
	EXEC-TRAJ-OBJ	Wait
	CALC-OBJ ARM-OBJ	Wait
EXEC-TRAJ-OBJ	EXEC-TRAJ-GOAL	Interrupt
CALC-TRAJ-GOAL	CALC-TRAJ-OBJ	Wait
CALC-TRAJ-OBJ	CALC-TRAJ-GOAL	Interrupt
	EXEC-TRAJ-GOAL	Interrupt
ARM-OBJ	ARM-BACK	Interrupt
	EXEC-TRAJ-GOAL	Interrupt
ARM-BACK	ARM-OBJ	Wait
OPEN-GRIP		
CLOSE-GRIP		
TURN-CAMERA		
TAKE-IMAGE	SEARCH-OBJ	Wait
SEARCH-OBJ		
GET-NEAR-OBJ		
CALC-GOAL	EXEC-TRAJ-OBJ	Wait
CALC-OBJ	EXEC-TRAJ-GOAL	Interrupt

Figure 10: The implemented services

PILO: generates dynamic trajectories (a set point moving along the computed path).

AVOID: filters, using proximity data, the reference points to locally avoid obstacles.

US: configures and activates the sonars and exports proximity data.

LOCO: controls the motor wheels with a feedback control on position and exports the information from the odometers.

EXPLOR: calculates new optimal positions to explore the environment.

LASER: uses the range-finder to simulate an arm seizing objects. The rotating platform is moved until the laser beam points to a certain position on the robot and the system waits until an object (placed by an human) intercepts the beam. Then, the platform returns to the neutral position.

VISUAL: updates the probabilities of the cells and simulates the detection of objects. Virtual objects are randomly disposed and are possibly detected when the robot is near enough and points the virtual camera in the right direction.

The required services to implement this experience, except for some initialization services, are presented in figure 10, with the associated policy to resolve conflicts.

We present in figure 11 an hypothetical execution of the mission. The robot must collect three objects and there are five ones in the hall (A to E). Solid and broken lines indicate planned and executed trajectories, respectively.

While going to the first exploration position (1), an obstacle is locally avoided (t1) and, before arriving (t2), an object is detected. The actual movement is interrupted and the robot goes to the object (A). The execution continues until the three objects (A, B and C) are collected and the mission finishes.

In the real experience, realized in a large hall in LAAS, the environment was divided in 30×30cm cells and HILARE2 found all the three virtual objects disposed in the hall. The modules and the executive run in on board VxWorks cards, while the PRS-based task level and the graphical interface run in a remote workstation, linked to the robot by a radio modem.

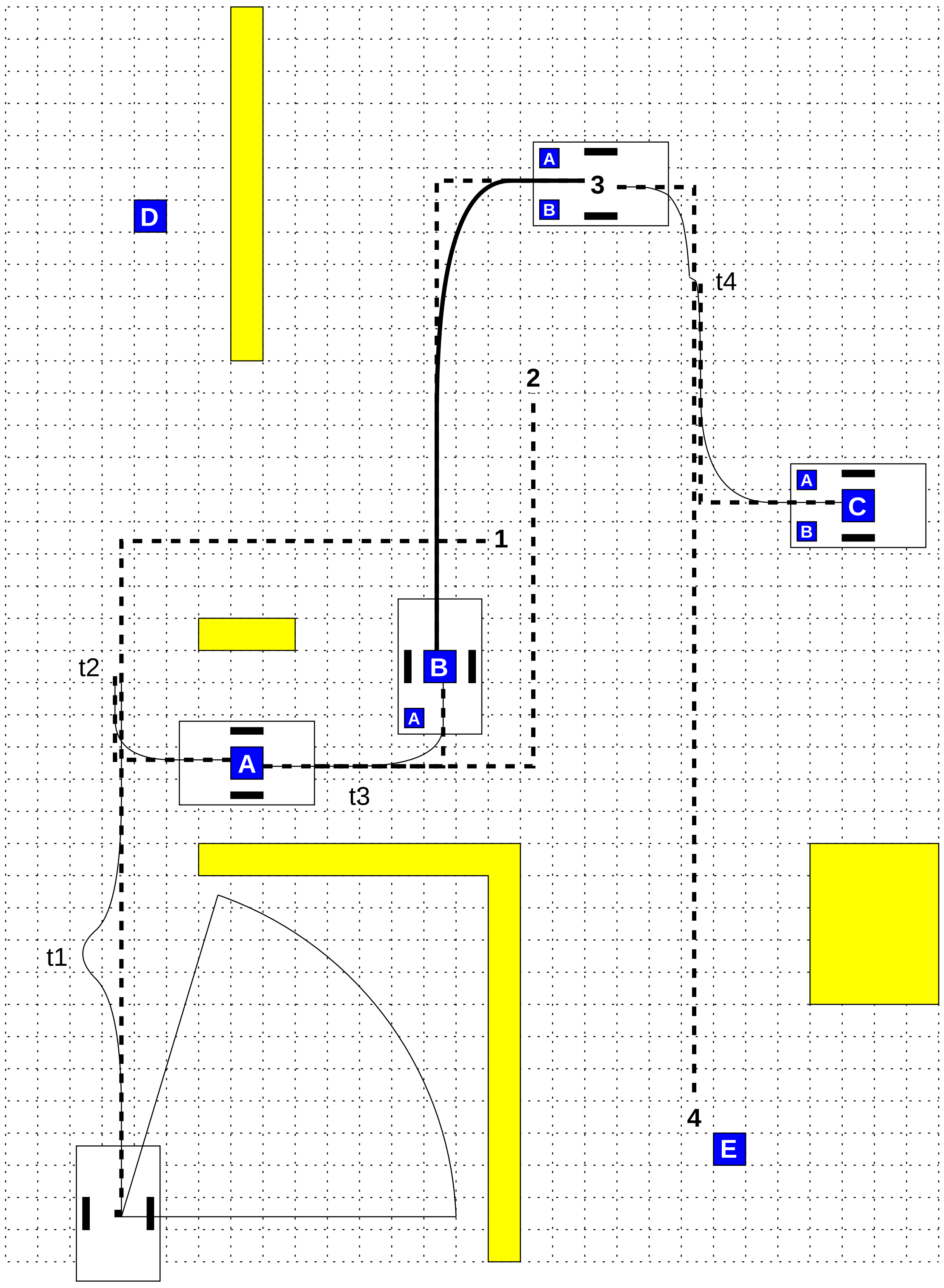

Figure 11: A scenario of execution

8 Conclusion

This experience, even if the robot mission is not complex, illustrate some advantages of introducing an executive and the associated services in a hierarchical architecture: an increased programmability of the robot, a better utilization of non-sharable resources and the facility to introduce parallelism in the plans.

Programmability: Using services, the modules can be developed without no association with a particular task, since all required adaptations are done while defining the new eventual specific services.

Our experience, for example, utilizes different sensors, actuators and algorithms, what would surely require more than one year to correctly develop and integrate from scratch. But this approach allowed to use all the pre-existent modules, and the complete experience was finished in three months.

Resources utilization: The experience could be based on a sequential algorithm:

- Determine an observation position.
- Calculate and execute a trajectory to this position.
- Search an object.
- If an object is found, execute a trajectory to the object and take it.
- Repeat this cycle until the desired number of objects if found.

But in this case the robot's resources are not efficiently used (we do not search objects while moving, for example). The executive facilitates the utilization of more efficient plans because parallelism is allowed even if conflicts can appear.

Easier parallelism: Without services, the procedures of the task level would have to be more complex to anticipate all the possible conflicts and the corresponding solutions. If the execution of the priority policy is delegated to the executive, the decisional level can specialize in the high-level decision-making, without wasting processing time on other problems.

There are some possibilities to improve the ideas presented in this article. One of them consists of semi-automating the generation of the Kheops rules that describe the behavior of the executive's conflict resolver.

The functional modules are assembled using GeNoM[7], a module generator. A formal description of some of their characteristics must be furnished, which includes a listing of the resources required by each function. This knowledge could be used to generate Kheops rules describing the behavior of the module in case of conflict. These rules, if complemented with specific rules to manage the inter-modules conflicts, would be a good starting point when developing the conflict resolver.

References

[1] J. S. Albus. **Outline of a theory of intelligence**. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):473–509, May 1991.

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

[3] R. A. Brooks. **Intelligence without reason**. In *Int. Joint Conf. on Artificial Intelligence*, vol. 1, pages 569–595, Sydney, Australia, Aug. 1991.

[4] A.-H. Cai et al. **Hierarchical control architecture for cellular robotic system – simulations and experiments**. In *IEEE Int. Conf. on Robotics and Automation*, vol. 1, pages 1191–1196, Nagoya, Aichi, Japan, May 1995.

[5] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. **Integrated planning and execution of autonomous robot actions**. In *IEEE Int. Conf. on Robotics and Automation*, vol. 3, pages 2689–2695, Nice, France, May 1992.

[6] R. E. Fayek, R. Liscano, and G. M. Karam. **A system architecture for a mobile robot based on activities and a blackboard control unit**. In *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pages 267–274, Atlanta, Georgia, USA, May 1993.

[7] S. Fleury, M. Herrb, and R. Chati la. **Design of a modular architecture for autonomous robots**. In *IEEE Int. Conf. on Robotics and Automation*, vol. 4, pages 3508–3513, San Diego, California, USA, May 1994.

[8] F. F. Ingrand, R. Chatila, R. Alami, and F. Robert. **Embedded control of autonomous robots using procedural reasoning**. In *ICAR - Int. Conf. on Advanced Robotics*, vol. 2, pages 855–861, Sant Feliu de Guixols, Catalonia, Spain, Sept. 1995.

[9] M. A. Lewis, A. H. Fagg, and G. A. Bekey. **The USC autonomous flying vehicle: an experiment in real-time behavior-based control**. In *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pages 422–429, Atlanta, Georgia, USA, May 1993.

[10] A. A. D. Medeiros and R. Chatila. **Execution control in autonomous mobile robots**. In *SIRS - Int. Symposium on Intelligent Robotic Systems*, pages 87–94, Lisboa, Portugal, July 1996.

[11] A. A. D. Medeiros, R. Chatila, and S. Fleury. **Specification and validation of a control architecture for autonomous mobile robots**. In *IROS - IEEE Int. Conf. on Intelligent Robots and Systems*, pages 162–169, Osaka, Japan, Nov. 1996.

[12] R. G. Simmons. **Structured control for autonomous robots**. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, Feb. 1994.

[13] C. E. Thorpe and M. Hebert. **Mobile robotics: Perspectives and realities**. In *ICAR - Int. Conf. on Advanced Robotics*, vol. 1, pages 497–506, Sant Feliu de Guixols, Catalonia, Spain, Sept. 1995.

[14] J. K. Tsotos. **Behaviorist intelligence and the scaling problem**. *Artificial Inteligence*, 75:135–160, 1995.