Around the Lab in 40 days...*

Rachid Alami†, Raja Chatila, Sara Fleury,
Matthieu Herrb, Felix Ingrand, Maher Khatib‡,
Benoit Morisset, Philippe Moutarlier§, Thierry Siméon
LAAS/CNRS,
7 avenue du Colonel Roche, F-31077 Toulouse Cedex 04, France
{*first-name.last-name*}*@laas.fr*

Abstract

In previous papers [1] we have argued that the LAAS architecture[1] is one of the most suited for mobile robot control. This statement may seem over-optimistic, not to say pretentious and unverifiable. After all, can we compare architectures? can we set up benchmarks? or can we measure how good an architecture is compared to another?
An architecture defines organization principles, integration methods and supporting tools. Comparing those tools, methods and principles may sometime end up in sterile controversies. However, we think there are means to measure the overall quality (or interest) of an architecture. Development time is for example one relevant criterion [5]. Basically, using a specific architecture, how long does it take to integrate a complete demonstration, including non trivial decisional capabilities, from the low level functional modules up to the supervisory level? This may seem a rather weak measure of an architecture quality, however, it encompasses properties such as: genericity and adaptability, ease of design and programming, extensibility and robustness.
In this paper we describe our recent experience in integrating a complete demonstration from scratch in 40 days using the LAAS architecture.

1 Introduction

Our laboratory is thoroughly examined and evaluated every three years by an external scientific committee. For its last three years evaluation in September 1999, the Robotics and AI group of LAAS/CNRS had decided to demonstrate, among other things, a robust indoor navigation task using a newly acquired XR4000 Nomadic robot.

Our goal, in this paper, is to show, and hopefully to convince the reader, that the architectural concepts and the software tools we have developed for years, constitute a powerful development environment for building robots and integrating their various functions into a coherent system. We are convinced that without them, it would have been impossible, at least for us, to construct such a demo in such a short period.
The reader may refer to [1] for a state of the art discussion. What we propose here is to report on an experiment and to share some results on the process of building an advanced robotic demo.
In the next section, we present the objectives and the constraints of the experiment that we decided to build and demonstrate. Section 3 summarizes our robot control architecture and its associated tools. We then show how we have used it to build a robot experimentation and integrate its various components (Section 4). Section 5 discusses the obtained results and concludes.

2 Objectives and Constraints

The real work on this demonstration started at the end of August, roughly 40 days before the big show. We started to meet at that time to decide what we wanted to demonstrate, and to make an inventory of what we had in stock.
The objectives of the experiment and the constraints linked to it were numerous... The most stringent one being the time we had to set the demo up.
The main objective for this first demonstration on this robot was to set up some robust indoor navigation in our lab environment. The robot should be able to continuously localize itself and navigate to subgoals avoiding known or unknown obstacles. Some interactions with the user should be possible such as giving high level missions (to go to some predefined locations), or to control the robot in a "compliant" mode (the robot moves away from the user, who can therefore control its motion). Symmetrically, the robot could request cooperation from the user, such as loading or unloading some

*This paper has been accepted for publication in IEEE ICRA 2000, San Francisco, USA.
†Authors list in alphabetical order.
‡Light Corporation (http://www.light.fr).
§Nomadic Technologies.
[1]LAAS: LAAS' Architecture for Autonomous Systems.

objects on it, or guiding it in through the "compliant" mode to some particular locations.

The robot we wanted to use was new to us. We acquired it in May 1999, and from then on, we mostly used it to port and test the generic tools we had been already using under Solaris or VxWorks. We had also studied and tested the XRDEV software[11] provided by Nomadic, i.e. the `nRobot` server, which allows clients to command (locomotion, speech, sensors states) and access various sensor data.

Another constraint we had was the number of people involved. Although the list of authors of this paper may suggest that nine persons worked on this demo, most of them were involved upstream and/or in other demonstrations, i.e. providing some of the ideas, algorithms or tools used. We roughly evaluate the amount of works spent during the 40 days to 3 men.months.

Figure 1: Diligent: our Nomadic XR4000.

The demonstration involved a standard XR4000 Nomadic robot (Fig. 1), called Diligent, equipped with a SICK laser range finder (LRF). Diligent has 2 Pentium II 266 MHz boards with 128 Mega each and a BreezeNet Ethernet radio[2] to access our Local Area Network.

As far as the software is concerned, we can distinguish between the algorithmic components, specific to a particular functionality we want to endow the robot with, and the generic tools which allow to implement the architecture itself [1].

The software tools we use are:

- GenoM, a programming environment used to program and set up the modules of the functional level [4] (§ 3.2.1).
- ComLib, a communication library providing a client/server protocol and shared memory primitives [3].
- Propice (formerly C-PRS), a procedural reasoning system, used to program the supervisor [6] (§ 3.2.2).
- TransGen, a tool used to generate automatically the Propice kernel which can interact with all the functional modules produced with GenoM (§ 3.2.3).
- GDHE, a 3D OpenGL/Tcl based system for 3D display of the robot in its environment(see Fig. 7).

All these software systems have been ported under Linux (XR-4000 Operating System) beforehand.

The algorithmic components we wanted to integrate in this demonstration are:

- A mapping and localization software to build a segment map from the LRF values and to perform localization based on segment matching [10].
- A 2D motion planner [9], which searches for a trajectory between two configurations in a polygonal environment
- A navigation based on an elastic band approach, which executes the trajectory produced by the motion planner, while avoiding local obstacles (detected by the LRF) [12].

Note that most of these systems were already implemented and "running" in other environments: for a different type of robot (e.g. The Hilare non holonomic robots), or on screen display only. Therefore we had to adapt them to this particular robot and in this specific environment.

3 A Software Architecture

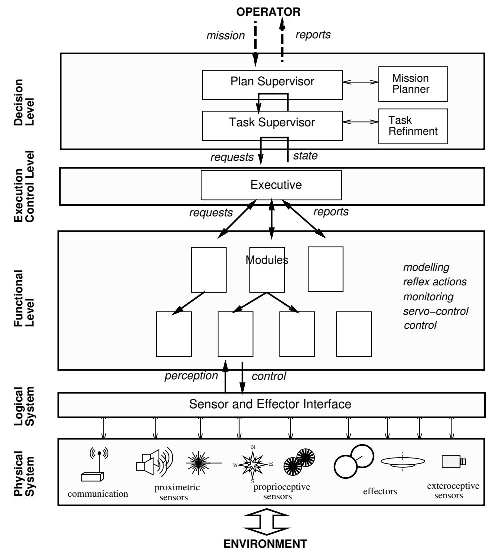

Figure 2: LAAS Software Architecture.

An architecture expresses organization principles and control methods. In this section, we briefly recall the main ideas underlying the LAAS architecture and the

[2] This connection is in fact, only used for display purposes, all the processing are made onboard the robot.

tools we have developed to implement it. The reader is referred to [1] for a complete discussion.

3.1 The LAAS Architecture

The LAAS architecture has three hierarchical levels (Fig. 2), having different temporal constraints and manipulating different data representations. From bottom up, the levels are:

- **A functional level.** It includes all the basic built-in robot action and perception capacities. These processing functions and control loops (image processing, motion control, *etc.*) are encapsulated into controllable communicating modules (§ 4).
- **An execution control level,** or Executive. Just above the functional level, it controls and coordinates dynamically the execution of the functions, distributed in the modules, according to the task requirements specified by the next level (§ 4.7).
- **A decision level.** This higher level includes the capacities for producing the task plan and supervising its execution, while being at the same time reactive to events from the other levels (§ 4.7). It may embed several layers, according to the application; their basic structure is a planner/supervisor pair that enables to integrate deliberation and reaction.

In the navigation demonstration, the decision level consists of only one layer including several activities. The instanciated architecture is shown on Fig. 3.

3.2 Software Tools

The software architecture of each of our robots is a specific instance of the generic architecture presented above. Its integration is achieved thanks to a set of tools that we present briefly in this section.

3.2.1 The Generator of Modules GenoM

The Generator of Modules GenoM is a tool to design and build real-time distributed software systems. It allows to easily encapsulate operational functions (algorithms broken down into elementary code chuncks called "codels") on independent communicanting modules. The functions can be dynamically started, interrupted or (re)parameterized upon asynchronous *requests* sent to the modules by any other element in the system, including other modules, the execution controller or the supervisor. A final *reply* that qualifies how the request has been executed by the module is sent back.
Thus the modules generated using GenoM are "standardized" servers. A module can integrate several synchronous or asynchronous functions (related to a same resource or functionality), and can execute them in parallel. In such a way, one can build modules specialized in camera control (image production with different parameters/filters), or actuator control (with several servo-control algorithms), or trajectory computation, or map making (with various modalities), etc.
During the execution of functions a module can export data in public structures called *posters*. Therefore, images, maps, trajectories, positions, laser or sonar data, etc., can be made available as they are computed.
Programming a module does not require to know about the on-board operating system, nor communication or other real-time procedures (synchronization...). Modules are automatically produced by GenoM using a template consisting in predefined fields. The fields of a request are related to its input parameters, its returned results, its possible failure reports, its conflicts with other services, its temporal constraints (period, if any).... The fields of a poster describe the data exported by the posters and the frequency of the poster update. Such systematic specification guarantees a coherence in module design (usually different modules are programmed by different persons) and allows to automatize their integration.
From this formal description GenoM produces:

- a module running under various operating systems (VxWorks, Solaris, Linux)
- interface libraries (services and data access) in various programming languages (C, Propice, Tcl)
- an interactive program for preliminary tests.

3.2.2 Propice

We encode the decision level using Propice (formerly C-PRS). Propice [7, 6] is composed of a set of tools and methods to represent and execute plans and procedures. Procedural reasoning differs from other commonly used knowledge representations (rules, frames, ...) as it preserves the control information (*i.e.* the sequence of actions and tests) embedded in procedures or plans, while keeping some declarative aspects. Propice is composed of:

- **A database** which contains facts representing the system view of the world and which is constantly and automatically updated as new events appear. In our robot architecture, the database contains symbolic but also numerical information such as the position of the robot, the status of the various sub-systems, posters names to retrieve their data, the currently used resources, *etc.*
- **A library of procedures/plans,** each describing a particular sequence of actions and tests that may be performed to achieve given goals or to react to certain situations. The content of this procedure library is application dependent and it may include

predefined plans to perform robot tasks.

In Propice, each plan/procedure is self-contained: it describes in which conditions it is applicable and the goals it achieves. This is particularly well adapted to context based task refinement and to incremental robot tasks.

3.2.3 TransGen

TransGen generates a Propice kernel dedicated to the supervisor. Basically, it takes a description list of the modules produced with GenoM and generates the code, the Propice data files (basic procedures) and the makefile to produce and run a Propice kernel to control the involved modules.
The result is a Propice kernel augmented with basic actions corresponding to all module requests (with input/output encoding/decoding) and with mechanisms to put request reports and posters values in the database.

4 How the Experiment was set up

We present in this section the different functionalities used in the demo that were integrated into six modules controlled by a supervisor (Fig. 3), and the progress toward demo day.

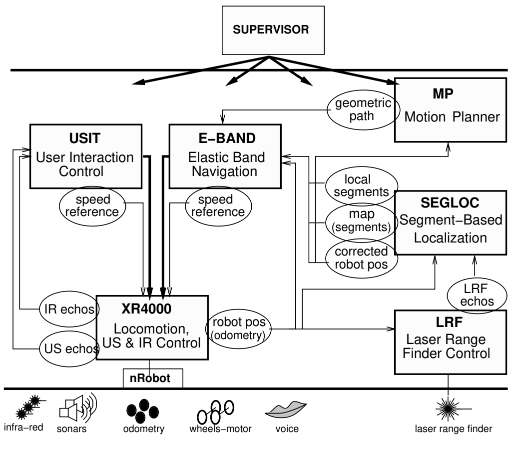

Figure 3: Diligent's Software Architecture: *The Robot Supervisor controls 6 modules (rectangles). 10 posters (ellipses) are regularly updated. Thin arrows indicate poster readers.*

4.1 XR4000

The XR4000 module implements the interface between the LAAS module-based architecture and the *XDREV* software environment provided by Nomadic Technologies on the XR4000 (Nrobot). This module accepts requests to control the robot basic motions (speed or position control of the three individual axes) and exports sensor data (odometry, sonars, infrared sensors) in three posters.

4.2 Laser Range Finder (LRF)

The LRF module controls a Sick laser range finder. It provides in a poster the raw sensor data (polar distances for each measurement point - every 0.5 degree over 180 degrees) and the robot's position corresponding to the middle point of a data set. The data are acquired with a period of 200 milli-seconds.

4.3 Segment-based Localization (SEGLOC)

This module is in charge of maintaining an accurate position of the robot. The localization method relies on a map of obstacle edges incrementally built by the robot itself from data in the 'Laser Echos' poster. Map building and localization procedures use an Extended Kalman Filter to match the local perception with the previously built model [10]). Three main modes of localization are provided by the module:

- The Re-Location Mode is used when the robot is lost.
- The Continuous Tracking Mode is used when a good probabilistic estimation of the robot position is available. It relies both on robot position variance and the predicted local aspect which is automatically computed from the estimated current robot's position. The result is merged with the current robot position.
- The Simple Tracking Mode is used when there is not enough information in the environment to produce a new position (e.g. in a corridor).

Fig. 4 shows a localization experiment in a map of about 400 segments previously built by the robot. The localization process in this relatively large model is performed is less than 120 milli-seconds. We run it at 1 Hertz. The trajectory is a large loop.

4.4 The Motion Planner (MP)

The main purpose of this module is to plan feasible paths for the robot. It is able to compute trajectories for holonomous as well as non-holonomous robots [9].
One of its peculiarities, is that it accepts requests which allow the supervisor to control and to adapt motion planning activities to the current task needs. Indeed, the supervisor can select the level of discretization, the type and the source of obstacles to be taken account by

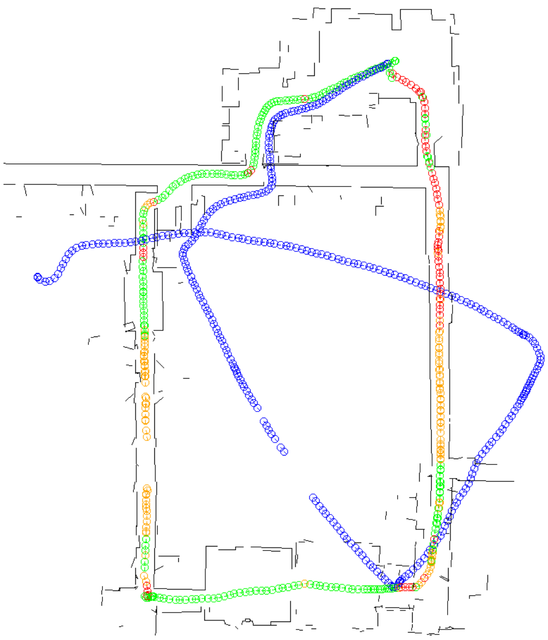

Figure 4: Comparison of position estimation obtained from XR4000 odometry alone (dark circles - missing positions are due to temporarily communication break down) and from the SEGLOC module (grey circles). Dimensions : $40x60m^2$.

the planner (from models built by the other modules), the shape and the kinematic constraints of the robot.
The MP module is used in conjunction with the E-BAND module described next: it produces an initial path which is dynamically adapted by E-BAND.

4.5 Elastic Band Navigation (E-BAND)

The *elastic band* is a method to dynamically modify a trajectory in order to take into account variations in the obstacle layout between the model used during path planning and the actual sensor data acquired during path execution. The method used here was first proposed by S. Quinlan and O. Khatib [12] and further developed and implemented by M. Khatib [8]. The principle is to build a flexible path between the current robot position and the goal, described by a sequence of configurations in free space. Connexity between these configurations is maintained by a set of internal forces that also optimize the global shape of the path. External forces are associated with obstacles and are applied to all configurations in order to maintain the path away from obstacles. The MP module is invoked again if the elastic has been broken. MP then tries to produce a path with a different homotopy class.
The computation time of an elastic band is proportional to its length. In order to maintain the real-time capabilities of the robot, the E-BAND module limits the portion of the trajectory that is elastic. If the band retracts beyond a minimum length, new points are added toward the goal along the trajectory computed by MP. Once it has reached the goal, the band retracts until it only contains a single configuration, indicating that the execution is completed.
In order to take into account moving obstacles, if, during execution, the path is blocked (that is, the distance between the next configuration of the band and the perceived obstacles is smaller than the radius of the robot plus some safety threshold) the execution of the band is stopped and a watchdog timer is armed. When the watchdog expires, a failure status is reported to the supervisor. If the next configuration becomes free again before the expiration of the timer (because the obstacle moved), the execution resumes.

4.6 User Interaction (USIT)

This module embeds the basic functions that are controlled by the supervisor for sensory human/robot interaction. For instance, it implements a safe compliant motion mode in which the robot moves as if it was "pushed" by a human; the direction of motion is detected using the ring of infra-red sensors while the ultrasonic sensor are used to create repulsive forces in order to avoid contact with obstacles. Another function (not used in the demo) is the detection and motion estimation of moving objects using data from the LRF.

4.7 The Robot Supervisor (RS)

The robot supervisor is programmed in Propice. It provides an interface with all the robot modules thus allowing to program robot control. All supervisor activities (task refinement, control, display, dialog...) are programmed using goal directed and/or situation driven procedures.
The main activities, performed by the Robot Supervisor, in its current version are: Mission Management, Robot Localization Control, Robot Navigation Control, Human/Robot Interaction Control. These different activities interact depending on the current context.

Mission management This activity involves the management of the set of missions that can be incrementally allocated to the robot through several means. In the current implementation, missions involve essentially navigation tasks. Mission management allows to add or delete missions through textual or through direct human/robot interaction. It is able to postpone, restart or re-plan missions depending on execution contingencies.

Robot Localization Control This activity runs and controls the localization system (see 4.3). It permanently monitors the "quality" of robot localization and implements different levels of reaction depending on the threshold reached by the robot position uncertainty. The first level induces a change in the re-localization modalities. The second level induces an active strategy (e.g. a momentary stop with rotations in order to find landmarks). The third level corresponds to the inability of the robot to solve the problem by itself; it postpones any autonomous motion tasks and asks for human help.

Robot Navigation Control Robot navigation tasks are implemented through the use of a "classical" plan-and-execute paradigm based on a previously learned map. However, this paradigm is enhanced by several features. First, trajectory execution, based on the control of an elastic band, allows an effective robustness to contingencies, from local obstacle avoidance, to adaptation to moving and transient obstacles as well as adaptation to significant re-localization updates. Second, if the band is completely blocked for a period of time, a new plan is searched, taking into account an update of the learned map. The loop is repeated iteratively until the robot reaches its goal or the planner finds no path, or an "external" event entails the postponement or the cancellation of the navigation mission.

Human/Robot Interaction Control This aspect will progressively take an important part in the project (indeed, Diligent is dedicated to the development and illustration of human/robot interaction in the context of Personal Robotics[2]). The activity is currently limited to the interaction between human and robot described in section 4.6.
The robot supervisor offers (1) a programmer interface through the Propice programming environment, (2) a textual user interface through a TCP/IP connection and (3) a user and public interface through the compliant motion.
Interfaces (2) and (3) allow the user to add or delete navigation missions (e.g. "GO-TO Station-N"). Besides, interface (3) allows the user or any person from the public to set the robot in a compliant mode. The human can then "push" the robot in a compliant but safe (no collision) manner. This mode is used in two situations: (1) when a human encountering the robot wants to move it aside and (2) to provide to an operator a comfortable and fast mean to guide the robot to a given place for example for allowing a robust re-localization.
Another interface feature is the use of the speech synthesis system to indicate the main changes in robot state.

4.8 Chronology

We now describe how the experiment was set up and made to run within 40 days.
D-40: The task force was composed of 5 persons. We met and decided which demo we wanted to show, what functional modules would be necessary to include, how they would interact, and what would be the role of the supervisor.
D-30: All the modules are described in GenoM. Requests, input/output data structures are defined. Of course, data structures will be refined as the development goes on, but the way all the modules and the supervisor are organized is such that any change in one module is automatically propagated throughout the rest of the software.
D-20: Most modules are now running on their own, and are being debugged stand alone. The first TransGen/Propice supervisor is produced and runs on the robot. The map of the environment is acquired by the robot itself (manually guided). Whoever moves a bookcase or a desk is doomed for two generations (or has to redo the mapping procedure).
D-10: The motion planner is now able to plan a path in the environment, although we still need to tune it. The elastic band module is able to take this trajectory to initialize its band and then execute it. The localization is still flaky, but usable. Interaction with the user, i.e. compliance and mission request starts to work.
D-3: Robots are real machines... and sometimes, you wished you had picked up a different research domain, such as software robots, multi agent simulation... any domains in which s... does not happen. Anyway, when you arrive early (after a short night), you start the robot, and after a little "plock" you see a little bit of smoke raising and you smell this characteristic odor which tells you that some electronic components have just decided to make your day... you know you are in it deep. Just to make a long story short, we were able to get some replacement parts from our colleagues at ENI of Tarbes (200 km far from Toulouse) and the robot (renamed Franck Einstein) was back working at D-2. We just lost one days... Thanks God, there are nights between days.
D-2: Our boss has doubled its coffee consumption and is starting to call travel agencies to find some far destination to expatriate. All the modules are now working "fine", and we are now testing (thus debugging) the supervisor for good. Diligent localizes most of the time, but still says "I'm lost" too often to achieve robust motions: the supervisor has to monitor when it gets lost and to plan appropriate action to re-localize. The motion planner and the elastic band locomotion get together pretty well, although we are having some problems with obstacles too close to starting position, which prevent then the locomotion to start.

D-1: Diligent navigates in the laboratory, plans paths from one point to another, executes them with its elastic band locomotion... but still gets lost or blocked unexpectedly by evanescent obstacles. We discover a bug in our low level LRF library which limited its range to 16 meters (thus creating "fake" obstacles modulo 16 meters for those who were further). Still debugging the supervisor which however shows robustness and good recovery.
D Day: Apart from a little glitch with the Ethernet communication (to raise the level of adrenaline), the demo goes fine, nobody got hurt. Diligent is asked by users to go from one point to another, it plans and executes its trajectories, avoiding great density of local obstacles (mostly reviewers testing our approach ;-)) and re-planning an alternative path to avoid them if necessary. Users are able to interrupt its task, move it in compliant mode and let it resume its suspended task. We display the whole demo on a screen (see Fig. 7).

5 Results of the Experiment

The demonstration for the review was successful, and we can already ensure that the robot is capable of long run robust navigation. Systematic and intensive tests have yet to be performed.
We give here a short description of the type of missions which where given to Diligent and how it performed them.
The basic function that we wanted to demonstrate is the ability of the robot to navigate safely and repeatedly in the laboratory environment from one place to another. Indeed, a short term goal is to allow its effective use by the lab members, through very simple and very intuitive commands.
We can say that we have succeeded in this first step:

1. the robot navigated safely in the environment, avoiding obstacles not present in the learned map[3]. The elastic band demonstrated good performance as well as a great flexibility in conjunction with the localization activity. Fig. 5 shows a path produced by the planner as well as the elastic band which drives the robot avoiding obstacles not present in the map.
2. the robot demonstrated effective autonomy in planning and executing its navigation tasks, end re-planning in case of permanent obstacle or "non-cooperative" human (we call "non-cooperative" human, a human who blocks completely the path of the robot and who does not leave place after a fixed amount of time - several seconds). Fig. 6 shows the robot after a while; it has been blocked at the door and found itself in a situation where it had to re-plan its path, finding a path which passes trough another door.
3. the robot behaved correctly in presence of humans in the environment. The compliant motion mode provided a very convenient way to move the robot aside when it bothers people.

[3]Obstacle avoidance is performed using only the LRF.

Figure 5: A navigation task (1).

Figure 6: A navigation task (2).

6 Conclusion

Even though the demonstration was successful and illustrative, our aim here is to draw conclusions not from the demonstration but from the process of building that demonstration. The LAAS architecture has effectively helped us considerably in this process. The concepts and the methodology served as a guide for the design and the structuring of the overall system.
The GenoM and TransGen tools allowed us to gener-

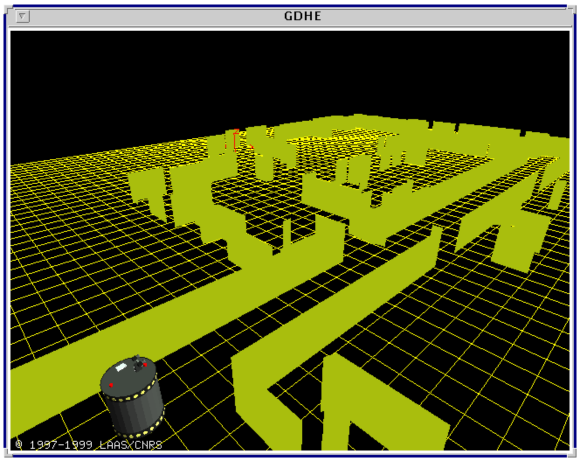

Figure 7: A display of the world as seen by Diligent.

ate automatically all the interfaces, the modules interpreters (shells) and the scripts for running and initializing the various robot components. The modules generated by GenoM and TransGen being independent from the underlying operating system, we were able to run and debug some modules (LRF, SEGLOC,...) using another robot platform (Hilare-2).

The benefit was really substantial: during all the programming and debugging process, we never had problems with process communication, real-time multi-process management, coding/encoding and transmission of data between different system components, "bugs" in the modules interpreters or in the interaction between the supervisor and the modules. All these aspects were dealt with by our software tools. Consequently, we concentrated our efforts on the performance of the functional aspects only.

For the high level, i.e. the Robot Supervisor programming, our experience in programming robot tasks using a context-based task refinement activity allowed us to develop autonomous and adaptative task achievement abilities. The Propice programming environment facilitates such programming style which embeds planning and execution monitoring.

For the future, we envisage to make a step further toward a more comprehensive and more rigorous approach for building autonomous user-friendly mobile robots supervisors. This will lead us to the development of generic programming tools for building *Robot Supervisors* and reducing as much as possible "manual" re-programming when one builds new robot software, adds or modifies robot functions, designs a new demonstration. Another challenging key issue, which is not present in our system, is the integration of learning capabilities at the Robot Supervisor level.

On the other hand, we will continue the development of Diligent toward long run human-friendly navigation in structured environments open to public. The development will involve alternative and/or complementary modalities for robot navigation and a substantial effort in designing advanced ingredients for human/robot interaction.

Acknowledgment

We wish to thank the Ecole Nationale d'Ingénieurs of Tarbes (ENIT) for lending us parts of their XR4000 robot so we could run the demonstration on due time.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[2] J. Canny and R. Alami. Workshop on Personal Robotics. In *IEEE ICRA, Detroit, (USA)*, 1999.

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

[4] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation, San Diego California, (USA)*, 1994.

[5] G. Giralt. Robot Autonomy and Machine Intelligence: Trends and Critical Issues. A post-Symposium Report. In *Proc. of the 6th International Symposium on Robotics Research, Pittsburgh, PA*, 1993.

[6] F. F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE ICRA, St Paul, (USA)*, 1996.

[7] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992.

[8] M. Khatib. *Contrôle du mouvement d'un robot mobile par retour sensoriel*. PhD thesis, Université Paul Sabatier, Toulouse, December 1996.

[9] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.

[10] P. Moutarlier and R. G. Chatila. Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modelling. In *Proc. International Symposium on Robotics Research, Tokyo*, 1989.

[11] Nomadic Technologies, Inc., Mountain View, CA. *Nomad XRDEV Software Manual*, March 1999.

[12] S. Quinlan and O. Khatib. Towards real-time execution of motion tasks. In R. Chatila and G. Hirzinger, editors, *Experimental Robotics 2*. Springer Verlag, 1992.