An Architecture for Dependable Autonomous Robots

Félix Ingrand, Raja Chatila, Rachid Alami
LAAS/CNRS
7 Avenue du Colonel Roche
31077 Toulouse Cedex 4, France
{felix,raja,rachid}@laas.fr

Introduction

Autonomous robots are complex machines embedding:

- software modules implementing basic functions such as servo loops to follow a path, or to compute stereo correlation of a pair of images.
- software to take into account the interactions between the robot components.
- programs which deal with the supervision and the planning of the mission.

These components are integrated within an architecture which defines an organization and a methodology for robot operation. This architecture, which determines the robot's capacities to achieve tasks and to react to events, should possess the following properties:

a. Programmability: The robot shouldn't be designed for a specific environment or for achieving a single task, and programmed in detail to do it. It should be able to achieve multiple tasks in different situations, described at an abstract level. Its functions should be easily combined according to the task to be executed.

b. Autonomy and adaptability: the robot should be able to carry out its actions, to refine or modify the task and its own behavior according to its goals and to the execution context as perceived by its sensors.

c. Reactivity: the robot has to take into account events with time bounds compatible with the correct and efficient achievement of its goals (including its own safety, which is a permanent goal).

d. Consistent behavior: the (re)actions of the robot must be guided by the objectives of its tasks.

e. Robustness: an individual action has well defined operating conditions. To deal with contingencies out of the scope of these conditions, the control architecture should be able to exploit the redundancy of robot functions.

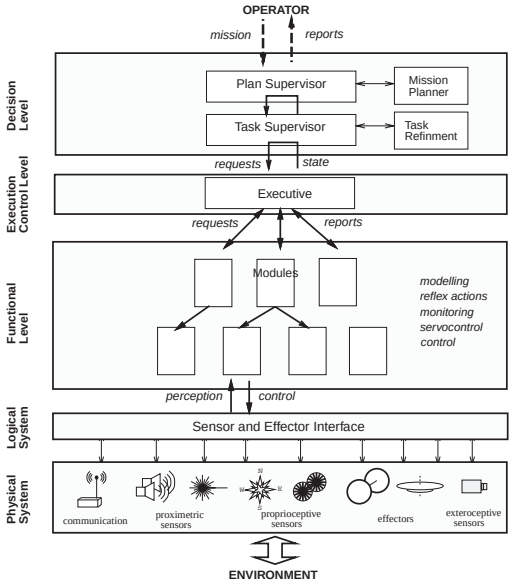

Figure 1: The LAAS Architecture

f. Extensibility: integration of new functions and definition of new tasks should be easy. Learning capabilities are important to consider here: the architecture should make learning possible.

Dependability as such is not often explicitly considered in such a system, except in very critical applications (*e.g.*, assisted surgery). However, issues (b., c., d., and e.) above deal with several aspects directly related to dependability and robust behavior. As robots are to interact more closely with humans (to assist, to communicate and to program), these issues become more central. We contend that they cannot be considered independently from the decision-making processes of the system. Indeed in its nominal operation, the robot control system should exhibit some robustness, *i.e.*, it should include within its decision-making processes the capacity to act in different contexts and to react to unexpected events. These decision-making processes include those that are classically considered as related to dependability and safety issues. Therefore the validation of the architecture itself and of its components becomes a central

issue for dependability. The architecture should provide tools to help the designer not only to integrate and develop the global robot system, but also to validate it both in its logical properties and temporal operations. The architecture we propose (called **LAAS**[1]) and use on our robots addresses some of these issues.

The LAAS Architecture

The **LAAS** architecture [Alami et al., 1998] has three hierarchical levels (Fig. 1), 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 with no fixed hierarchy.
- 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.
- 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. It may embed several layers, which integrate deliberation and reaction, according to the application.

Approaches and Tools

The software architecture of each of our robots is a specific instance of the generic architecture presented above. Its integration is achieved using approaches and tools which we briefly present, with an emphasis on the dependability aspects.

Automatic Generation of Functional Modules: GenoM

The Generator of Modules GenoM [Fleury et al., 1994] 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 communicating 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. A module operation can be modeled by a finite state automaton.

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.

Programming a module does not require knowing about the on-board operating system, nor communication or other real-time procedures (synchronization...). Modules are automatically produced by GenoM using a template. Such systematic specification guarantees a coherence in module design (usually different modules are programmed by different persons) and allows to automate their integration. Moreover, all the modules being based on a template, only the codels and the particular activity states automata need to be specified and eventually proven to improve the robustness and dependability.

The Executive: Kheops

The Executive is certainly one of the most critical components of the architecture, dependability wise. It is a purely reactive system, with no deliberation capability, which acts as a time bounded filter between the decisional level and the functional level. It is thus responsible for catching poor or inadequate decision which could come from the decisional level or the user.

It receives a flow of requests to be executed from the decision level. Based on the request to transmit, on the current state of the set of modules, and on a finite state automaton, it selects, parameterizes and synchronizes dynamically the adequate requests which are passed to the functional level.

The state of the set of modules is maintained by the Executive, according to the ongoing activities and to the output of previous processing, *i.e.* according to requests sent and to replies returned by modules once an activity is over.

The finite state automata is computed beforehand, off line, from a set of propositional rules and encodes the desirable or undesirable interactions between the functional modules. We currently use Kheops [Alami et al., 1998], a rule based system that meets the compiling

[1] **L**AAS **A**rchitecture for **A**utonomous **S**ystems.

requirement of a finite number of possible rule chaining and involves basically a monotonic deduction on a propositional logic. Such an approach allows us to prove a number of properties thus improving the overall dependability of the system.

Supervision and Plan Execution Control: Propice

The supervisor and plan executor of the decision level is a reactive system which controls and refines the execution of the plan provided by the user or the planner. It monitors the reports, in response to the requests it sent, and takes corrective actions when they are specified. Moreover, it recognizes critical situations and executes predefined operational procedures to take care of them. It is implemented in Propice [Ingrand et al., 1996] which is composed of a set of tools and methods to represent and execute plans and procedures. 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, 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.

The supervisor and plan execution control components provides mechanisms to improve the overall dependability of the system (monitoring situations, reports analysis and error recovery, etc). However, if the plans provided by the planner are logically sound, those provided by the user practically remain unverifiable, hence the need for the executive which provides safe but conservative mechanisms.

Planning/Scheduling: IxTeT

Several planners are needed in a robot. Some of these, such as path and trajectory and manipulation planners, are modules of the functional level. The task and mission planner belongs to the decision level. It is a system queried by the supervisor. It has to deal explicitly with time, not only in task duration, but also in parallel activities within compound tasks, and in various temporal constraints between conditions (before or while a task proceeds) and effects of a task. It should deal, at planning time, with the predictable part of a dynamic environment, e.g., contingent change not under the robot control such as day/night cycles, resource availability profiles, or expected events. It has to manage the resources of the robot, preferably while planning, not as a subsequent resource allocation and scheduling step, after the tasks have been chosen and constrained.

We developed and use the IxTeT planner, which is based on a reified logic formalism. In IxTeT, attributes are temporally qualified by the predicate *hold*, which asserts the persistence of an attribute value over an interval; and the predicate *event*, which states an instantaneous change of values. Resources are expressed by the predicates *use*, which represents a borrowing of a quantity of a resource over an interval, *consume* and *produce* which state the consumption or production of a quantity of a resource.

IxTeT *time-map manager* relies on time-points as the elementary primitives. Time-points are seen as symbolic variables on which temporal constraints can be posted. A *time-map manager* propagates *numeric* and *symbolic constraints* (precedence, simultaneity) to ensure the global consistency of the network. It answers queries about the relative position of time-points. Management of atemporal variables is achieved through a *variable constraint manager*. We consider variables ranging over finite sets and propagate *domain restriction*, *equality* and *inequality constraints*. Constraint propagation on atemporal variables is achieved through classical CSP techniques.

The initial plan is a particular task that describes a problem scenario, that is: (i) the initial values for the set of instantiated attributes (as a set of explained events); (ii) the expected changes on some contingent attributes that are not controlled by the planner (as a set of explained events); (iii) the expected availability profile of the resources ; (iv) the goals that must be achieved (usually, as a set of assertions); and (v) a set of temporal constraints between these elements.

The use of a logically based planner dramatically improves the overall dependability of the system.

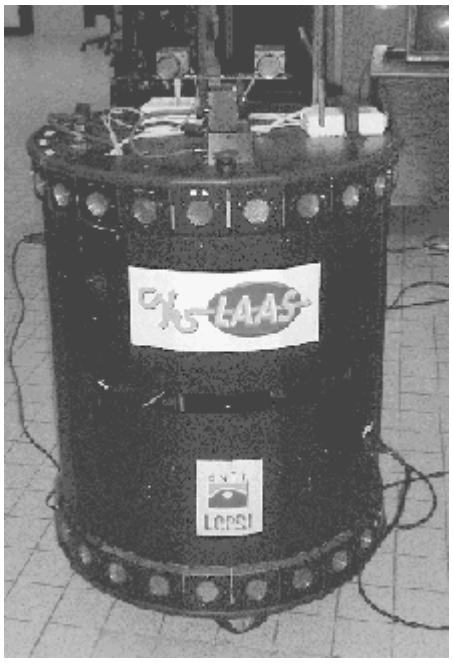

Figure 2: Diligent is based on XR4000 mobile platform

Indeed, such planner relies on logical and formal foundations and, providing the actions and the states representation is correct and sound, the plans produced are also correct and sound.

Robust Robot Operation

Rationale

In order to achieve dependable behavior, robot modules are designed so that they can cope with some contingencies and variations in their operating context, but within well defined limits. Out of these limits, It is at the architectural level, *i.e.*, in the organization of the system, that dependability is achieved, through the decision-making processes that analyze the situation and select the most adequate actions. We illustrate this by the following example of an indoor navigation task implemented on the mobile robot *Diligent*. We use a classical plan-and-execute paradigm based on a previously learned map. However, this paradigm is enhanced by several features to achieve dependable behavior. A geometric path planner provide the trajectory to reach a specified goal. 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 path 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.

A localization module using the known map enables to cope with odometry errors, providing the robot with a accurate positioning.

Hence each module has its own capacity to cope with errors (localization uses odometry or model-based localization, trajectory execution includes obstacle avoidance), but a the supervisory level, decision to replan when the individual modules are unable to complete the task, enables to achieve a more global dependable behavior.

Robot Modules and Architecture

Diligent's current architecture consists of a supervisor and six modules: we briefly summarize their main functions.

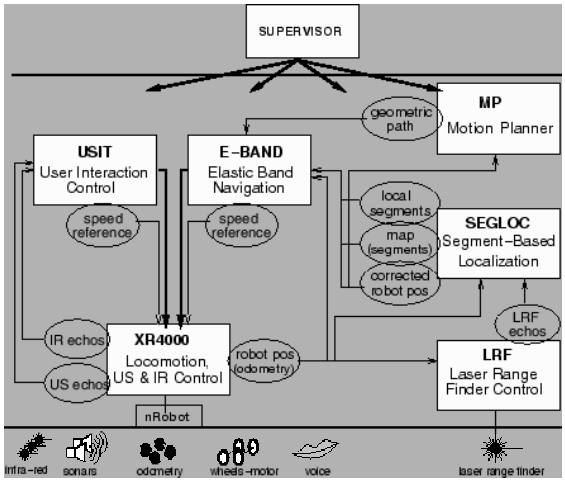

Figure 3: Diligent's Software Architecture: *The Robot Supervisor controls 6 modules (rectangles).*

The XR4000 module implements the interface with the basic robot primitives provided by the robot manufacturer.

The LRF module controls a laser range finder.

The Segment-based Localization (SEGLOC) module is in charge of maintaining an accurate position of the robot. Map building and localization procedures use an Extended Kalman Filter to match the local perception with the previously built model. Fig. 4 shows a localization experiment in a map of about 400 segments previously built by the robot. The trajectory is a large loop.

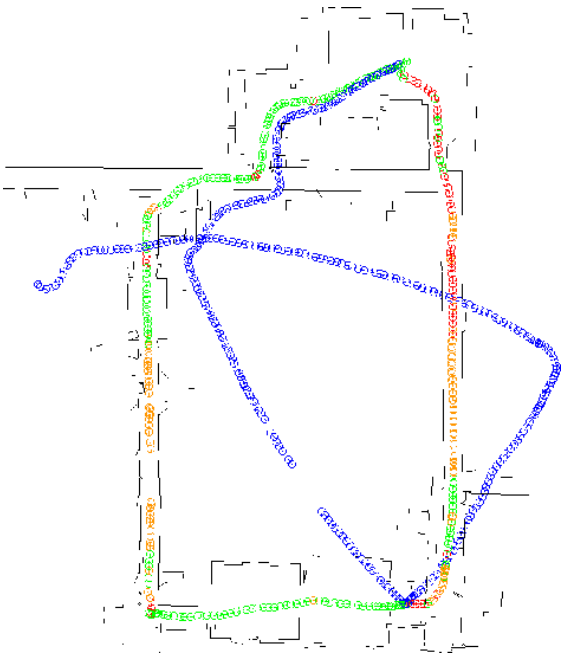

Figure 4: Comparison of position estimation obtained from XR4000 odometry alone (dark circles) and from the SEGLOC module (gray circles).

Dimensions: 40x60m2.

The main purpose of the Motion Planner (MP) module is to plan feasible paths for the robot. Depending on the current task needs, the supervisor adapts the motion planning activities: it selects the level of discretization, the type and the source of obstacles to be taken account by the planner, the shape and the kinematics constraints of the robot.

The E-BAND module uses an *elastic band* 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 principle is to build a flexible path between the current robot position and the goal, described by a sequence of configurations in free space. Connectivity 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 robot is intended to navigate in human environments, and safety issues are therefore of concern. The USIT module embeds the basic functions that are controlled by the supervisor for a safe sensory human/robot interaction. For instance, it implements a 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 ultra-sonic sensor are used to create repulsive forces in order to avoid contact with obstacles.

Robot Supervisor

The robot supervisor is programmed in Propice. 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, are: Mission Management, Robot Localization Control, Robot Navigation Control, Human/Robot Interaction Control. These different activities interact depending on the current context.

Fig 5. shows a path produced by the planner as well as the elastic band that drives the robot avoiding obstacles not present in the map.

Figure 5: A navigation task : the planned path and its execution.

Figure 6: A navigation task: re-planning in case of failure.

Fig.6 shows a situation where the robot has been blocked at the door. The supervisor decided to re-plan its path, to pass trough another door.

The robot demonstrated effective autonomy in planning and executing its navigation tasks, end re-planning in case of permanent obstacle, thus achieving a dependable behavior.

Conclusion and Future Dependability Issues

We believe the presented architecture provides already a number of features and characteristics that implement an effective overall robustness and some ingredients of dependability for robots. Issues of dependability and trustworthiness of robots will become more and more important and will deserve more efforts. Indeed, applications such as robots for surgery and robots in direct interaction with the public impose very severe dependability constraints.

The presented architecture and associated tools can serve as a basis for future work in this domain. Indeed, a number of issues remain to be addressed:

to provide mechanisms similar to those based on synchronous languages (*e.g.*, those present in Orccad [Orccad, 1998] to define the activity states automata in the GenoM modules.

- to provide mechanisms (*e.g.,* timed automata and such [S. Bornot and J. Sifakis, 1998]) to prove temporal properties of the functional modules.
- to provide an action representation which remains consistent over the various components of the architecture.
- better integration and a tighter loop between the planning/scheduling system and the execution, including anticipation mechanisms.
- specific procedures to deal with error analysis and recovery (e.g., error trees) for non-nominal situations.
- specific robot state evaluation procedures to improve redundancy in action selection and in dealing with physical failures in sensors and effectors.

We are also convinced that architectural design is only a partial answer to the question. The overall robot design should of course take benefit of the most recent generic advances in dependability for computer systems. And last, but not least, the robot functional and algorithmic components should also be designed with improved robustness, enlarging their capabilities in various operational contexts by embedding a learning component.

Concerning personal robots in direct interaction with humans, one has also to address all topics related to user-acceptability issues as well as human objective and subjective confidence on the robot capabilities.

References

[Alami et al., 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, “An Architecture for Autonomy”, *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming,* Vol. 17, No. 4, pp. 315-337, April 1998.

[Bornot and Sifakis, 1998] S. Bornot and J. Sifakis. “An Algebraic Framework for Urgency”. *In Calculational System Design, NATO Science Series, Computer and Systems Science 173, Marktoberdorf, July* 1998. Long version to appear in Information and Computation, 2000.

[Fleury et al., 1994] 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.

[Ghallab et al., 1994] M. Ghallab and H. Laruelle. “Representation and Control in Ixtet, a Temporal Planner”. In *Proceedings AIPS-94*, pages 61-67, 1994.

[Ingrand et al., 1996] 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'96, St Paul, (USA)*, 1996.

[Orccad, 1998] The Orccad Team. “The Orccad Architecture”, International Journal of Robotics Research, Special issues on Integrated Architectures for Robot Control and Programming, vol 17, No 4, pp 338-359, April 1998.