A simple architecture for modular robots

A. Godin

DGA/ETAS
Route de Laval
BP 60036 - Montreuil-Juigné
49245 AVRILLE Cedex

Abstract

Different ongoing robotics projects financed by DGA [1] aim at developing modular demonstrators with advanced autonomous functionalities: in order to support these developments, state services carry out studies on both stable and emerging concepts, especially concerning architectures issues.

The ArMoR architecture, presented in this article, is an effort to build a structure, based on well-known proven solutions, to demonstrate the viability of robotic systems. At the same time, it aims at being a test platform in which to validate and capitalize new functionalities. Such a work is often conducted by trainees, not always students in robotics, and the underlying architecture should therefore be simple to apprehend and offer a minimalist programming interface. Modularity and simplicity were thus part of the initial requirements.

Section 1 presents a brief analysis of previous work, that will point out the main trends in architectural design. The interest of hybrid structures, that is the choice we have retained, will be pointed out. Hence, the following sections will first focus on the modular reactive layer (section 2) and, in a second step, on the minimalist controller, inspired by Petri nets approaches, section 3. A set of tools, coming alongside with the framework to build the modular layer and the mission plan, and still in development at that time, is then presented in section 4. The results of our first tests, conducted in simulation with open source SLAM algorithms and mentioned in section 5, tend to prove the viability of the implementation. Finally, section 6 will conclude on the limitations of this work and on some promising approaches that we hope to integrate as soon as they prove robust and ready for operational use.

Keywords

Architecture; Robot; Modularity.

1 PREVIOUS WORK

For the last twenty years now, scientists have been looking for more autonomous systems and many solutions have been proposed. Among them, two main trends appeared.

Autonomy is often considered as a synonym of "intelligence": the robot becomes autonomous as soon as it can analyse the environing world, plan actions in consequence and execute them. In such an approach, a high level process imposes its decisions, that take into account both the environment and the initial goals that were given to the system. For this reason, the resulting actions are often quite optimal in the long term. So-called deliberative

[1] DGA is the procurement agency within the French Department of Defence

architectures are especially relevant in cases where a fault during execution can be catastrophic. For instance, in space exploration missions, for which no human can interfere in case of deadlocks, a mobile robot should not execute a specific move, in reaction to a local obstacle, without checking the safety of the new path. In the 80s', the NASA developed an architecture, NASREM [1], following the previous paradigm. A global memory, storing the state of the environment, is updated by three main communicating functionalities, task decomposition, world modelling and sensory processing. In the resulting highly structured conception, it is worth noting that effectors commands are only issued by the "task decomposition" block: each action can only be taken after a planning step, no immediate reflex in response to sensory data is allowed. The TCA framework [2] is another example of a mainly deliberative architecture based on task decomposition and a strict respect of the resulting plan.

However, reflexive hierarchic architectures make it difficult to build very fast systems. Besides, when these systems evolve in a dynamic world, sometimes while moving themselves, safety is not always guaranteed by long-term planning and it should be possible to apprehend upcoming unpredicted events. A possible solution to solve this reactivity problem is to associate a predefined action to a given perceptive stimulus: a short term loop is hence inserted between perception and effectors control. Obviously, such approaches are minimalist and it has quickly been proposed that more advanced data processes be implemented, often called "behaviours" in the literature. One of the most famous architectures of this kind was introduced by Brooks [3] in the 80s'. The robot is organized in layers; each module is allowed to inhibit the outputs or suppress the inputs of a module belonging to a lower layer and to provide its own output in place (i.e. higher layers subsume lower ones). Especially, different working layers concurrently imply that the robot is able to pursue multiple goals so that it is not only restrained to simple tasks. The complexity of the overall structure is however a probable drawback of the "subsumption" design. More flexible frameworks, based on the behavioural approach, have been further proposed. In DAMN [4], for instance, navigation modules all deliver their outputs to a central arbiter which generates the final driving command. Modularity is increased but the framework implies that all behaviours provide the same kind of data, in this case for navigation purpose only. For a system that should execute more diverse tasks, a generalisation of this principle is necessary, such as the one proposed in the reactive layer of AuRA [5]. Behaviours (here called "motor schemas") also send their results to processes that use a potential-fields-like method to merge the different contributions and generate the controls; but, compared to DAMN, multiple arbiters are allowed, each dedicated to a particular task. At this point, the architecture has all the cumulated advantages of Brooks' subsumtpion and Rosenblatt's DAMN ones: multiple goals pursuing, flexibility, generalizability and reactivity.

Reactive structures are nevertheless known to produce dead-end situations, especially for navigation tasks. Consequently, many authors have proposed to mix deliberative approaches and behavioural ones so that a high reactivity is achievable while anticipatory capabilities are added. Most of the time, "hybrid" architectures are based on a layered design, where the reflexive upper part controls the execution of the reactive one. The LAAS architecture [6], for instance, respects this structure: it has been successfully implemented in very diverse systems, from very autonomous planets-exploration robots to automatic museum guides with continuous interaction with human beings. Many research teams have since proposed different variations: CLARAty [7] introduced a two-layered framework, where planning and control are tightly linked in the upper level; Ranganathan an Koenig [8] worked on an architecture in which replanning is triggered on demand when behaviours execution fails; Albus et al. [9] presented a design where layers are replaced by a hierarchy of nodes, each one gathering deliberative and reactive capabilities. But all approaches confirm the current interest for this kind of hybrid

structures, which have besides proved very competitive: for instance, [10] illustrates the performances of a military demonstrator running 4-D/RCS for an autonomous navigation task.

From an implementation point of view, most recent works in robotic applications have proposed object-oriented approaches, some implicitly, like Carmen [11] that uses the C language, others taking directly benefit of OO-languages inheritance properties, like Player/Stage [12]. Some architectures (CLARAty [13]) explicitly make the assumption of an OO-hierarchy as the base of their design. Advances in computer science have also brought more formal models, through the introduction of component-based approaches [14]. Besides, the latter provide a whole framework that also proposes a methodology to generate and deploy the components [15], [16]. Finally, it has been demonstrated that software has reached sufficient robustness to undertake real long term missions, as shown by Stanley, the 2005 DARPA Grand Challenge winner [17].

ArMoR project

The project began in 2006, while we decided to attend, on our own, to the first European Land-Robot Trial (Elrob'06 [18]). At that time, the need was a very simple modular software that could embed teleoperation, vision and basic cartography modules. Even if hardware issues prevented us to go further, the developed program really revealed easy to use. However, some limitations had shown up, such as robustness. At the same period, a couple of internships gave very promising results but most students, learning computer science, had difficulties to apprehend existing robotic frameworks and the underlying concepts during their three months of presence in the robotics department. Therefore, we were in search of a tool that could gather these individual programs and offer, for later works, an easy to understand, simplified, interface. Provided some evolutions, the Elrob software was considered a relevant base.

The following of this article thus describes the simple Architecture for Modular Robots (ArMoR) that was developed taking into account both these constraints and the state of the art presented above. Therefore, the main requirements for the framework were:

- Robustness. The Stanley [17] and Carmen [11] paradigms were retained, that is running independent processes, whatever local or distributed, so that the failure of some of them does not imply the failure of the whole system.
- Simplicity. The software must present a minimal programming interface (a minimal set of usable functions) so that as little prerequisites in robotics as possible be needed. On another hand, we are convinced that this simplicity can only be attained if the architecture itself comes with appropriate tools, possibly graphical ones, to help in the conception of the system.
- Representativity. Ease of use does not mean low performance. Our robots are periodically implied in demonstrations for military representatives and should be able to run state-of-the-art algorithms and prove that they can be driven by (kind of) military plans. This naturally conducted us to retain a hybrid structure and, thus, to add a control layer that was missing in the original software.
- Modularity. Evaluating systems belongs to our daily activities. This means that our students works should be easily integrated in an overall structure that then enables us to carry out tests and comparisons. This is only possible with a modular design. In particular, the addition of a specific module must not imply the re-compilation of the whole architecture.

Solutions chosen to answer these four main requirements, both the operational ones and those that are currently under development, are detailed below.

2 A MODULAR DESIGN

The current ArMoR is based on a two-layer model. The lower level gathers modules which provide the functional capabilities to the system: navigation, mapping, environment observation are examples of what could be inserted. As in the LAAS architecture [6], the functional module notion embraces a large number of different behaviours. For instance, for navigation task, it covers from pure reactive obstacle avoidance to short term trajectory planning. In brief, modules in this level can implement every function that only needs local information provided by instantaneous perception of the environment.

2.1 Module model

Each module of the functional layer can link to every other module to retrieve the data it needs. Links establishment is realized through a publish/subscribe mechanism which resembles in some ways the strategy used by ECA society within its robots [19]. But contrary to the latter, in which the central agent forwards messages to the subscribers, in ArMoR the central module is only compulsory during the initialisation of the system. Each module declares its publications, then opens a socket connection and listens to it. The central agent, that keeps trace of published data, when receiving a subscription request, only provides the reference of the server module (i.e. its IP address and the port on which it listens). The two modules then initiate a direct connection between each other. In case of the central agent failure, the link remains valid and functionalities, especially those in charge of preserving the robot integrity, keep on running. It should be noticed that modules can detect when a connection is closed, which happens when the data provider crashes. In that case, the concerned input of the module is also closed and a subscription request is reemitted to the central agent, so that the connection can be re-established if the missing module is started again.

As will be seen later, modules also initialise connections with a configuration database. On the whole, each module can thus be modelled as a component which is dedicated to a specific task (i.e. which runs its own process) and manages three types of flow: control, configuration and data. For each of the first two types, the module opens a bi-directional port, while data ports are differentiated into outputs and inputs. Figure 1 gives the external representation of a module.

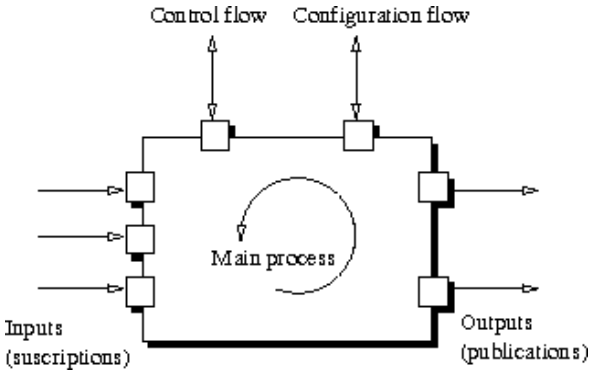

FIG. 1 – Global model of a module.

2.2 Flow Model

In the current implementation, all data and control flows respect a "push" model: the provider asynchronously sends the message, without request from the target module. This is indeed a simple way of insuring that all modules receive and work on the updated information. However, as discussed in section 6, this mechanism is sometimes problematic for data management and, in a next version, should be complemented with a "pull" mode.

2.3 Configuring the modules

Each output, and so each link, is associated with a specific data, which is identified by a unique name, initially read in a configuration file (report to section 4 for its creation). Different modules can subscribe to the same data and, consequently, connect to a same output. Inversely, a given input can only receive data from one output. These connections are initially described in the same file. Finally, in general, the core process of a module needs to be tuned depending, at least, on external conditions and on the system type (for instance dimensions and dynamics for a mobile ground vehicle). All parameters necessary for the module to run are also noted in the configuration file.

A usual way to configure algorithms is to associate each one with a particular file. Most of the time, from one author to another, naming conventions and file format vary drastically. In ArMoR, the choice was made to gather all configuration parameters in a server, that reads the unique -XML- configuration file mentioned above. At start-up, the server reads and stores all the information: for a given module, it includes the names of data served by the outputs, the names of data the component subscribes to and, optionally, all parameters needed to run.

During its initialisation stage, each module automatically asks the configuration server for the outputs and inputs related information so that it can emit the associated publication and subscription requests to the central agent. If parameters have to be retrieved, equivalent messages can be sent; in parallel, an internal mechanism increases a counter for each request and decreases it each time a value for a given parameter is received. A specific command, available to the developer for the module code, enables to wait for this counter to return to zero before continuing execution: this guarantees that the module is completely configured before starting its actual task, in the case when this condition is needed.

Finally, the current implementation takes benefit from the "inotify" Linux-kernel daemon to monitor the configuration file and notify the server if it has been modified. The configuration is then automatically refreshed. However, changes are not immediately sent to the concerned modules. Contrary to data or control, configuration flow is initiated by the component: subsequent requests from a module will actually result in receiving the correct updated values.

2.4 Overview of the functional layer API

The current implementation, following all the above principles, offer an programming interface limited to 12 functions. The first two ones are used to declare publications and subscribe to data (given the name of the data and its type). As multiple instances of a same component can be used, the system gives an internal unique name to the data, that will differ from the "id" argument. For further operations, the user can get this unique name through the two latest functions.

```
void provide(std::string id, int type);
void suscribe(std::string id, int type);
std::string getProvidedData(std::string id);
std::string getSuscribedData(std::string id);
```

If necessary, a user can define some specific parameters in the configuration file in order for the module to run correctly. These parameters can be retrieved through a set of three functions. The last one generally follows calls to "getParameter()": it enables to wait for the reception of all values so that the main loop of the component cannot start before it is completely configured.

```
void getParameter(std::string paramName);
virtual void parameterReceived(std::string paramName, std::string paramValue);
void synchronize();
```

Manipulation of data flows is enabled with the three following functions, the last one providing the current time in a common referential for all the modules. This time can especially be used, in the case of a component producing a data, to retrieve the instant of creation to be included in the "header".

```
int sendData(data_t header, unsigned char * data);
virtual void dataReceived(data_t header, unsigned char * data);
double getTimestamp() const;
```

Data handling is most often managed in the main loop of the module that is implemented in the virtual following function.

```
virtual void exec();
```

Finally, a special function is provided in order to tell the system that this particular component can not be commanded by the execution controller. This is especially useful for module that provide a man-machine interface so that it cannot be freezed.

```
void setNonControllable();
```

We have found that this API is enough for controlling all aspects of a component but still is very convenient to use.

3 EXECUTING THE MISSION

As explained above, our mobile robots participate to exhibitions with officials. In such circumstances, they aim at demonstrating credibility of robotics concepts and adequation to military needs.

3.1 Military missions and planning

Every military mission is based on a plan, which is then declined to each level of the hierarchy. This plan specifies both time constraints and geographical information such as dedicated action area, dangerous zones, enemy and friendly troops location. It is also a dynamic element since it must be updated while units are moving and achieving their tasks. A view of a characteristic plan is shown on figure 2.

We argue that robotic systems should be able to work with such a plan since they (will) belong to military units that affect, feed and use it. The control level of our robots should thus include mechanisms to handle planning information: synchronization schemes to conform to imposed time limits; planning models that allow to specify and pursue multiple goals and respect geographical constraints; possibilities to receive events and updated upper objectives; processes to repair or re-compute the current plan when new pieces of information contrast with it.

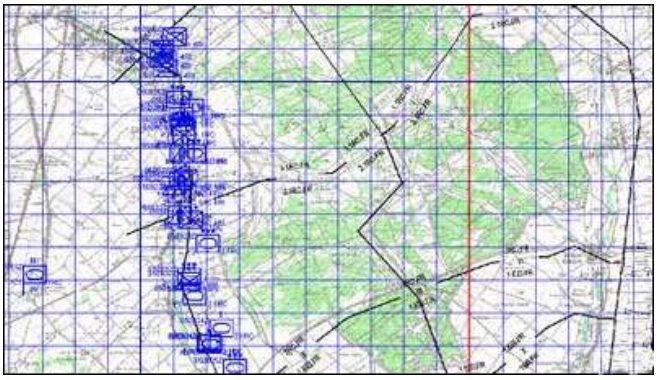

FIG. 2 – Aspect of a military-like plan. Blue symbols on the left indicate friendly units; black horizontal lines delimitate the action area; black vertical lines are used to represent time appointments (synchronization of the action).

Our department maintains tight relations with French research laboratories. While working with them, we have been convinced by a specific tool, Petri nets, that inherently handles the above-quoted mechanisms. Especially, the recent thesis [20] or article [21] emphasize that the use of this mathematical approach is well-suited to assign tasks to robot, monitor them, replan when necessary, even for teams of heterogeneous agents. Time constraints are themselves transparently managed, either implicitly (transitions in the net work as a synchronisation mechanism), or explicitly (time conditions can be added to the transition firing rules). As a consequence, we have decided to implement a "Petri net-like" feature in ArMoR to control the activities of the modules.

A word on task planning: we here made the hypothesis that a quite detailed plan was known and that we only needed to translate it into a Petri net form. But in the general case, only high level goals are specified and they need to be "broken" into a sequence of unitary tasks which, taking into account the system capacities, can be transferred to the robot.

Many approaches have been proposed for planning, that primarily differ on the way the mission is described. A classical approach uses the STRIPS language [22], to enunciate the goals of the mission and the rules to achieve them. A variety of algorithms exists to solve the resulting problem, GraphPlan [23] being a famous example that inspired many other planners. Constraints solvers also provide an elegant solution to the planning problem that can inherently handle time-based equations; an example of such planners can be found in [24]. Other approaches can also rely on heuristics [25], satisfactory problems [26] or Markov processes.

In a military context, we think that planning with constraints satisfaction techniques presents many advantages: time is handled, geographic limits can be easily expressed, resources limits are directly translated into constraints. Moreover, [20] has shown possible to directly use plans produced by such solvers into a Petri net-based controller. In a further version of ArMoR, we envision to embed such a planner, for example the free open-source Choco tool [27].

3.2 Controlling the robot

The mission plan in ArMoR is specified as an XML file that lists all the functional modules that should be activated during the mission. More specifically, each module is seen as a place of a simple Petri net. Transitions between places are also listed in this configuration file.

At the initialisation of the system, the file is read and, for each place marked as "initial", the corresponding module is activated. Then, a one second-periodic process checks if the different

transitions in the net are activable and, if it is the case, fires them; input places are commanded to stop their execution, output places are activated. However, not all necessary mechanisms are implemented yet: for example, the controller does not take into account modules reports (for instance, alerts that their task has ended) even if this capacity should be added soon. Currently, the user can only associate timeouts to the transitions: if one is activable and its timeout is non-zero, it is fired after the delay is reached. This enables the current active modules to run for a specific amount of time. This first basic version of the controller, at least, enables to autonomously execute simple missions.

4 BUILDING THE SYSTEM

As mentioned in section 1, ArMoR is associated with a set of graphical tools to enable the user to easily apprehend the architecture. A first one is currently under development and is used to build the functional layer of the robot; a second one, that will offer an interface to prepare the mission plan, is envisioned.

4.1 Organizing the modules

While developping ArMoR, we were looking for a way to easily build the functional layer of the system: specifying the components to be inserted, configuring and linking them. Some verification mechanisms should also prevent the user from making mistakes. Scientific softwares editors have already been confronted to the issue and, for long, have proposed the use of block diagrams: for instance, Simulink or RT-Maps [28] have retained this concept which drastically decreases the learning time on the software.

Our tool offers a similar interface: the current version is shown on figure 3. It lists all the components that are stored in a specific directory and executes each one in a configuration mode to retrieve its inputs, outputs and parameters. Once the component is dragged to the workspace, the user can access these properties by double-clicking on it. In order for the developer not to forget specifying them, the number in the block indicates how many parameters are needed; anyway, when asking for saving the diagram, the tool checks whether a value has been assigned to each of them. Linking the modules is also verified. It is actually forbidden to connect two inputs, two outputs or, more generally, two ports that do not handle the same type of data. When creating a link and dragging the mouse over a specific destination port, the user is advised that the action is allowed by the colour of the port, that turns red. These very basic mechanisms make it very fast to build the diagram representing the functional layer and ensures that the resulting configuration file holds no mistake.

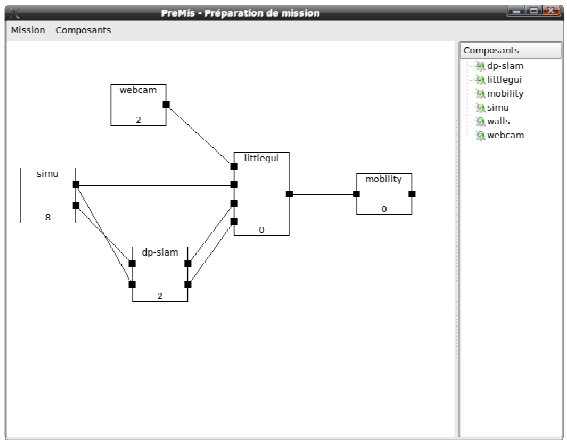

FIG. 3 – View of the graphical tool to build the robot internal organization.

4.2 Creating the plan

While the creation of the functional layer is of the responsibility of a developer, specifying the tasks that the system should achieve is the role of the end-user, in our case the soldier. As a consequence, the tool dedicated for mission preparation should not look like a scientific tool and, on the contrary, should take advantage of the users reflex.

With the introduction of geographical information systems (GIS) at all levels of the military hierarchy, people have become used to handling such devices. Actually, section 3 emphasized the tight similarity between the graphical form of the plan (the map with tactical information, as shown on figure 2) and the description of the mission with Petri nets. We thus argue that a relevant tool should simply offer a map view to the user, on which to enter waypoints, time constraints and active modules, and translate the resulting plan into the corresponding Petri net.

This tool does not exist yet in the current implementation of ArMoR but we at least envision to offer the following functionalities:

- enter the legs of the trajectory. Each leg could be described by two waypoints and the action that should be pursued, both on its extremities and on the leg itself. We here call "action" a set of modules that should be active.
- associate time constraints to the legs. It can be a waiting period (for observation phase for example), in which case a duration is entered, or an absolute date for synchronization purpose or if the overall mission schedule imposes it (appointment with other units).
- use standard military symbols and graphical conventions.

Ideally, corresponding work should be conducted in coordination with DGA teams specialized in GIS so that the mission preparation tool should look like a conventional military one and offer all the necessary functions.

5 PRELIMINARY RESULTS

As of the first tests, three properties were checked: ability of the framework to handle large amounts of data; ability to run processes consuming many resources; ease of deployment. Our choice fell on SLAM algorithms as they meet the two first properties. Actually, no one in our department is a real mapping expert, so we were unable to modify deeply such an algorithm to tune it: achieving the port to ArMoR should then prove that the framework is flexible. Finally, military units are actually interested and convinced by the utility of automatic mapping process so the feature is worth implementing quickly in our robot.

Some open-source SLAM algorithms can be found on the Internet [29], some of them already implemented in C with the Carmen API[2]. Two algorithms, DP-SLAM [30] and GridSLAM [31] were thus retained. The work consisted in removing all calls to the Carmen functions so that both algorithms became self-executable programs, then locating all points in the code were data were consumed or produced (odometry and laser measurements, maps) and replacing the corresponding lines with calls to ArMoR API. No modification in the SLAM process was brought at all. At the moment, DP-SLAM is entirely ported and GridSLAM is "on the way".

A whole system was tested with the resulting algorithm. Odometry and raw laser data is provided by a simulation component which integrates error models, so that the SLAM process

[2] ArMoR itself is written in C++ so we were first searching for C or C++ pieces of software

is fed with realistic measurements. A preliminary HMI and a webcam components are also available. A last module, that aims at controlling the mobility of our real robot (an ATRV Mini), is ready too but still needs to be validated. The block diagram corresponding to the tested system is the one shown on figure 3 and results are presenting on figure 4.

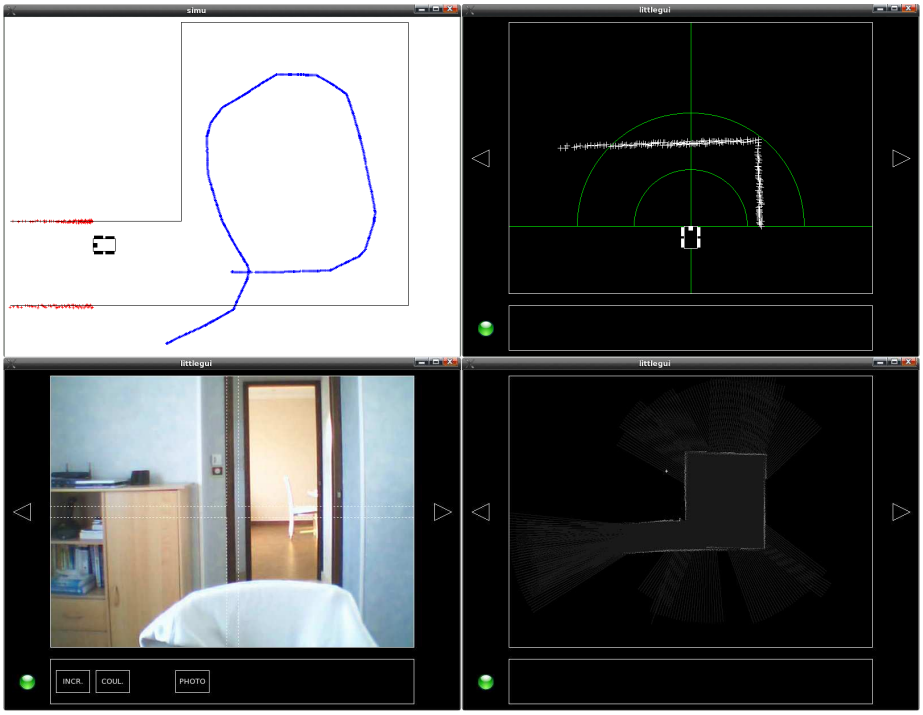

FIG. 4 –Views of the system used of the tests. The upper left window is the simulator, where the blue dots are the odometry measurements (with a well visible drift) and the red dots are the laser impacts on the walls. The upper right window is the user interface presenting raw laser measurements, the lower left receives the video flow (in the real system, for teleoperation task). Finally, the lower right window gives an overview of the DP-SLAM mapping result. All processes run and exchange data correctly as shown here.

6 DISCUSSION AND CONCLUSION

If DGA tries to support the scientific effort in robotics, through multiple research programs, its role also implies to promote technologies towards military units, as soon as they reveal robust and ready for operational use. This has guided our work on ArMoR which has been built as the aggregation of well-known principles. Its minimal API is also a quality and answers to the constraint of making it accessible to the largest number of people. It thus provides a relevant framework for capitalising available algorithms and students works. But this must not conceal the fact that it remains a "minimalist" architecture too, without all the possibilities of modern ones, developed in laboratories. Hereafter, we discuss on some functionalities that still lack in our own work and would deserve to be integrated.

a) Data links: currently, components establish direct links using TCP protocols. But it is known that, in some cases such as remote control, UDP can reveal more efficient. Depending on the performance of the current implementation, this second protocol will be studied and proposed as an alternative to link modules.

b) Data flow: it was highlighted that information exchanges in ArMoR respect a "push" model: producers send their data as soon as it is available. In the case when a consumer component runs at a slower rate, data processing can not be started on each reception as it will not be finished on the next data arrival. Some specific mechanisms have to be implemented, so that the data is ignored until the current iteration of the main process loop is finished. In such cases, it would be actually more efficient to work with a "pull" model, that is the consumer requests the latest available data when needed. This model has to be added in further versions of our framework.

c) Robustness: at the moment, mechanisms presented in section 2 prevent the whole system from crashing if one component accidentally disappears. They also enable to re-establish the connections if the dead module is launched again. However, no automatic fault detection and component reactivation are currently included. In a near future, we plan to implement a watchdog to overcome this issue. Some previous works have proposed solutions: the Stanley vehicle [17], for instance, was equipped with a safety mechanism, that was later released in the Carmen toolkit [11]. Besides, our colleagues in Arcueil successfully managed to add this functionality to their own architecture [32], demonstrating the viability of the solution.

d) Control: the controller currently implemented is actually simplistic. A complete system that could accept more features of the Petri nets has to be developed alongside with the graphical tool that will translate the mission plan into a correct net. It must also be noted that, at the moment, the human user cannot interfere in the control process. This issue of man-machine interaction, that is probably one of the most difficult problems raised in the robotics community today, has to be addressed in the future. Ongoing works, in the DGA as well as in laboratories, tend to show that adjustable autonomy is a very promising way. In the near future, if some consensus is reached on this concept, we envision to implement it.

References

[1] J. S. Albus, R. Quintero, and R. Lumia, "*Overview of NASREM: The NASA/NBS standard reference model for telerobot control system architecture*", NASA STI/Recon Technical Report N, vol. 95, April 1994.

[2] R. Simmons, "*Concurrent planning and execution for a walking robot*", Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-90-16, July 1990.

[3] R. Brooks, "*A robust layered control system for a mobile robot*", IEEE Journal of Robotics and Automation, vol. 2, no. 1, pp. 14–23, 1986.

[4] J. Rosenblatt, "*DAMN: A Distributed Architecture for Mobile Navigation*", Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1997.

[5] R. C. Arkin and T. R. Balch, "*AuRA: Principles and practice in review*", Journal of Experimental and Theoretical Artificial Intelligence (JETAI), vol. 9, no. 2/3, pp. 175–188, April 1997.

[6] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "*An architecture for autonomy*", International journal of robotics research, vol. 17, no. 4, 1998.

[7] I. A. D. Nesnas, "*CLARAty: a collaborative software for advancing technologies*", in NASA Science Technology Conference, June 2007.

[8] A. Ranganathan and S. Koenig, "*A reactive robot architecture with planning on demand*", in Proceedings of the Intelligent Robots and Systems Conference (IROS), vol. 2, October 2003, pp. 1462–1468.

[9] J. Albus, H.-M. Huang, E. Messina, K. Murphy, M. Juberts, A. Lacaze, S. Balakirsky, M. Schneier, T. Hong, H. Scott, F. Proctor, W. Shackleford, J. Michaloski, A. Wavering, T. Kramer, N. Dagalakis, W. Rippey, K. Stouffer, S. Legowik, J. Evans, R. Bostelman, R.

Norcross, A. Jacoff, S. Szabo, J. Falco, R. Bunch, J. Gilsinn, T. Chang, T.-M. Tsai, A. Meystel, A. Barbera, M. L. Fitzgerald, M. del Giorno, and R. Finkelstein, "*4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0*", National Institute of Standards and Technology, Tech. Rep. NISTIR 6910, August 2002.

[10] A. Lacaze, K. Murphy, and M. DelGiorno, "*Autonomous mobility for the DEMO III Experimental Unmanned Vehicle*", in Proceedings of AUVSI, July 2002.

[11] CARMEN Robot Navigation Toolkit. [Online]. Available: http://carmen.sourceforge.net

[12] The Player Project: Free Software tools for robot and sensor applications. [Online]. Available: http://playerstage.sourceforge.net

[13] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "*CLARAty: Coupled Layer Architecture for Robotic Autonomy*", Joint Propulsion Laboratory, Tech. Rep. D-19975, December 2000.

[14] C. Côté, Y. Brosseau, D. Létourneau, C. Raïevsky, and F. Michaud, "*Robotic software integration using MARIE*", International Journal of Advanced Robotic Systems, vol. 3, no. 1, pp. 55–60, March 2006. [Online]. Available: http://marie.sourceforge.net

[15] The Orocos Project: Smarter control in robotics & automation! [Online]. Available: http://www.orocos.org

[16] R. Passama, D. Andreu, C. Dony, and T. Libourel, "*Overview of a new robot control development methodology*", in Proceedings of the First National Workshop on Control Architectures of Robots, Montpellier, April 2006.

[17] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "*Stanley: The robot that won the DARPA Grand Challenge*", Journal of Field Robotics, vol. 23, no. 9, pp. 661–692, 2006.

[18] European Land-Robot Trial. [Online]. Available: http://www.elrob.org

[19] C. Riquier, N. Ricard, and C. Roussel, "*DES (Data Exchange System), a publish/suscribe architecture for robotics*", in Proceedings of the First National Workshop on Control Architectures of Robots, Montpellier, April 2006.

[20] O. Bonnet-Torrès, "*Replanification locale pour une équipe d'agents hétérogènes*", Ph.D. dissertation, ONERA, decembre 2007.

[21] M. Barbier, J.-F. Gabard, D. Vizcaino, and O. Bonnet-Torrès, "*ProCoSA: a software package for autonomous system supervision*", in Proceedings of the First National Workshop on Control Architectures of Robots, Montpellier, April 2006.

[22] R. Fikes and N. Nilsson, "*STRIPS: A New Approach to the Application of Theorem Proving*", Artificial Intelligence, vol. 2, 1971.

[23] A. Blum and M. Furst, "*Fast planning through planning graph analysis*", Artificial Intelligence, no. 90, pp. 281–300, 1997.

[24] P. van Beek and X. Chen, "*CPlan: A constraint programming approach to planning*", in AAAI/IAAI, 1999, pp. 585–590.

[25] B. Bonet and H. Geffner, "*Planning as heuristic search*", Artificial Intelligence, vol. 129, no. 1–2, pp. 5–33, 2001.

[26] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "*Chaff: Engineering an efficient sat solver*", in 39th Design Automation Conference (DAC 2001), June 2001.

[27] Choco constraint programming system. [Online]. Available: http://choco.sourceforge.net

[28] N. Dulac, "*Real time, multisensor, advanced prototyping software*", in Proceedings of the First National Workshop on Control Architectures of Robots, Montpellier, April 2006.

[29] OpenSLAM. [Online]. Available: http://www.openslam.org

[30] A. Eliazar and R. Parr, "*DP-SLAM 2.0*", in IEEE International Conference on Robotics and Automation (ICRA), 2004.

[31] D. Haehnel, D. Fox, W. Burgard, and S. Thrun, "*A highly efficient Fast-SLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements*", in Proceedings of the Conference on Intelligent Robots and Systems (IROS), 2003.

[32] R. Jaulmes and E. Moliné, "*HNG: A Robust Architecture for Mobile Robots Systems*", in European Robotics Symposium 2008, ser. Tracts in Advanced Robotics. Springer, 2008, vol. 44, pp. 121–131.