Acromovi architecture: A framework for the development of multirobot applications

Patricio Nebot & Enric Cervera

Robotic Intelligence Lab, Jaume-I University
Castellón de la Plana, Spain

1. Introduction

The presented agent-based framework (Acromovi - an acronym in Spanish which stands for Cooperative Architecture for Intelligent Mobile Robots) was born from the idea that teamworking is an essential capability for a group of multiple mobile robots (Jung & Zelinsky, 1999; Mataric, 1998).
In the last years, there is an increasing interest in the development of systems of multiple autonomous robots, so that they exhibit collective behaviour. This interest is due to the fact that having one single robot with multiple capabilities may waste resources. Different robots, each one with its own configuration, are more flexible, robust and cost-effective. Moreover, the tasks to achieve may be too complex for one single robot, whereas they can be effectively done by multiple robots (Arai et al., 2002).
In this work is described the design and implementation of a distributed architecture for the programming and control of a team of coordinated heterogeneous mobile robots, which are able to collaborate among them and with people in the accomplishment of tasks of services in daily environments, the Acromovi architecture.
Acromovi architecture is a framework for application development by means of embedding agents and interfacing agent code with native low-level code. It addresses the implementation of resource sharing among all the group of robots. Cooperation among the robots is also made easier in order to achieve complex tasks in a coordinated way.
Moreover, in this work is emphasized the reusability of software, allowing the programmer to seamlessly integrate native software components (vision libraries, navigation and localization modules) , by means of agent wrappers, providing the programmer with a set of tools for mobile robots. Also, it allows sharing the robots' resources among the team and an easy access to the robots' elements by the applications. Other important characteristics of the Acromovi architecture are the scalability and ease-of-use.
Though robot programming has been extensively done in C or C++ languages, a Java-based multiagent development system was chosen to develop the architecture of our team of robots (Nebot & Cervera, 2005a). Among Java strengths, those

particularly pursued were the high-level communication capabilities, and the native interface to existing C/C++ code.
On the other hand, Acromovi architecture is a distributed architecture that works as a middleware of another global architecture for programming robots. Though any common agent architecture can be used, it has been implemented by means of the JADE (Java Agent DEvelopment Framework), a tool for the development of multiagent systems, implemented in JAVA, that fulfils with the FIPA specifications.
The embedded agents that constitute the Acromovi architecture work as interfaces between the applications and the physical elements of the robots. Some agents also make easier the handle of these elements and provide higher-level services. The embedding agents are the key for powerful distributed applications being rapidly developed.
Multiagent systems are an important tool for the development of new robotic applications. They are the natural environment for the development of applications which consist of more than one robot. And they make possible the fast implementation of powerful architectures for the specification and execution of tasks for those teams of robots.
In Section 2, some related works in the literature are compared. Next, Section 3 describes the different robots that make up the team. In Section 4, a description of the Acromovi architecture is given, from the design to the implementation. Some examples of applications implemented with the Acromovi architecture are presented in Sections 5. These applications demonstrate the rapid development capabilities and ease-of-use of the agents implemented with this framework. Finally, Section 6 concludes and presents some final remarks.

2. State of the art

The amount of research in the field of cooperative mobile robotics has grown steadily in recent years (Parker, 2000; Cao et al., 1997). Some examples of these works are presented below, emphasizing their potentials, and drawbacks, and whether they are related (or not) with embedded agents in mobile robotics.
CEBOT (CEllular roBOTics System) is a hierarchical decentralized architecture inspired by the cellular organization of biological entities. It is capable of dynamically reconfigurating itself to adapt to environment variations (Fukuda & Iritani, 1995). It is composed by "cells", in the hierarchy there are "master cells" that coordinate subtasks and communicate among them.
On the other hand, *SWARM* is a distributed system made up of a great number of autonomous robots. It has been called the term "SWARM intelligence" to define the property of systems of multiple non-intelligent robots to exhibit a collective intelligent behaviour. Generally, it is a homogeneous architecture, where the interaction is limited to nearest neighbours (Johnson & Sugisaka, 2000).
Other interesting project, *MICRobES*, is an experiment of collective robotics that tries to study the long time adaptation of a micro-society of autonomous robots in an environment with humans. Robots must survive in these environments as well as cohabit with his people (Picault & Drogoul, 2000).
In an attempt to use traditional IA techniques, the *GOPHER* architecture was thought to resolve problems in a distributed manner by multirobots in internal environments

(Caloud et al., 1990). A central tasks planning system (CTPS) communicates with all the robots and disposes a global vision of the tasks that has been done and of the availability of the robots to perform the rest of tasks.
An example of behaviour-based architecture is *ALLIANCE*, which is oriented to the cooperation of a small-medium team of heterogeneous robots, with little communication among them (Parker, 1998). It assumes that robots are relatively able to discern the effects of their actions and those of the rest of agents, either through its perception or through communication. Individual robots act based on behaviours or sets of behaviours to make their tasks.
ABBA (Architecture for Behaviour Based Agents) (Jung & Zelinsky, 1999) is another behaviour-based architecture for mobile robots, whose purpose is to design an architecture to model the robot's behaviour so that it can select reactive behaviours of low level for his survival, to plan high level objectives oriented to sequences of behaviours, to carry out spatial and topological navigation, and to plan cooperative behaviours with other agents.
Dynamic physical agents are considered in an architecture called *DPA*, being well suited for robotics (Oller et al., 1999). A physical agent is the result of integrate a software agent in an autonomous hardware. This hardware is frequently a mobile robot. DPA architecture shows the agent like a modular entity with three levels of abstraction: control module, supervisor module and agent module. The architecture's operation is based on two basic processes for each module: the process of introspection and the one of control, and in a protocol of intermodular dialogue which allows the exchange of information between modules.
In order to reuse native software components, agents can be embedded in an interface level or middleware. *ThinkingCap-II* (Cáceres et al., 2003) is an architecture developed, in a project of distributed platform based on agents for mobile robots. It includes intelligent hybrid agents, a planning system based on a visual tracking, vision components integration, and various navigation techniques. Furthermore, it has been developed over a real-time virtual machine (RT-Java), implementing a set of reactive behaviours.
Miro is a middleware for create autonomous mobile robot applications from the University of Ulm. It is based on the construction and use of object-oriented robot middleware to make the development of mobile robot applications easier and faster, and to foster portability and maintainability of robot software (Enderle et al., 2001). This middleware also provides generic abstract services like localization or behaviour engines, which can be applied on different robot platforms with virtually no modifications.
The previous works implement architectures and systems so that teams of mobile robots can make cooperative tasks. Our work consists of the implementation of an architecture of such type, emphasizing software reuse, and adding the versatility and power of the multiagent systems, for the resolution of cooperative tasks for a group of heterogeneous robots.

3. Description of the team of robots

In order to understand the purpose and working environment of the presented agent architecture, their physically embedding instances are described in this section: a

team of mobile robots. For the development of this project, a team consisting of six Pioneer robots and one Powerbot robot is used, all of them manufactured by ActivMedia Robotics, as can be seen in Fig. 1.

Fig. 1. Team of robots used in the project.

Though sharing the same platform, the robots exhibit different features, constituting therefore a heterogeneous group. Within the group, dimensions of robots may vary due to the different accessories added to robots (cameras, laser, grippers), or its own size (Pioneer vs. Powerbot).
Particularly, only one Pioneer has a laser rangefinder system and two Pioneers carry camera systems; moreover, the Powerbot has front and rear bumpers and front IR sensors. All the Pioneer robots have a gripper and a front ring with 8 sonar discs, except the robot with laser that also includes a rear sonar ring. The Powerbot has 28 sonar discs distributed 14 at front and 14 at rear, but it does not include a gripper. Instead, a 7 d.o.f. arm is mounted on its top, as can be seen in the last section of this work.
All robots of the team maintain the same logical design based on a microcontroller which transfers the readings of the standard elements of the robot (ultrasound sensors, wheel encoders) via RS232, by cable or radio, to a computer client where the applications are executed. This computer can be located on board of the robot or not, in which case the communication is made by radio-modem. In our team, onboard computers are only available on three Pioneer robots (those that also carry either a camera or the rangefinder) and the Powerbot.
Finally, the robots with onboard computing power are connected by means of a Wireless Ethernet network, which, through several base stations, can access to the local network of the university, and to the Internet.

4. Acromovi architecture

To establish mechanisms of cooperation between robots implies to consider a problem of design of cooperative behaviour: given a group of robots, an environment and a task, how the cooperation must be carried out. Such problem implies several challenges, emphasizing among them the definition of the architecture of the group.

The multiagent systems are the natural environment for such groups of robots, making possible the fast implementation of powerful architectures for the specification and execution of tasks.

4.1 Design of the architecture

The robots that constitute the team have already a programming architecture with native (C/C++) libraries for all their accessories. In the Pioneer robots, this architecture is splitted in two layers. The lower layer is called ARIA, and it takes care of the requests of the programs to the robot components. The upper layer -called Saphira- provides services for range sensor interpretation, map building, and navigation.

However, the existing native code is oriented to both local and remote processing from only one controller, without defining collaboration mechanisms between robots or applications. Acromovi architecture tries to overcome this problem by the introduction of a new level over this architecture that allows an easy collaboration and cooperation.

Also, the Acromovi architecture is able to subsume any other extra software, like the ACTS (a colour tracking library) and the Vislib (a framegrabbing library). ACTS is a native (C/C++) module which, in combination with a colour camera and a framegrabber, allows the users to select regions of colour in an image. ACTS is an external program to the architecture that works as a server, and the applications work as clients that communicate by means of an ARIA library. Vislib is a library that provides a fast and flexible image processing and single-camera machine vision. The image processing functions that Vislib offers are generic convolution (2D kernel), filtering and morphological tools. Vislib also is written in the C language.

Subsuming existing C/C++ libraries is a powerful and quick method for rapidly developing Java code and thus embedding agents with all the functionality of native systems. Furthermore, the new code adds seamless integration with other distributed agents, either running on remote applications, or even as applets in web pages.

Acromovi, in Fig. 2, is a middleware between the native robot architecture and the applications, which allows sharing the resources of each robot among all the team of robots. This middleware is based on an agent-embedding approach.

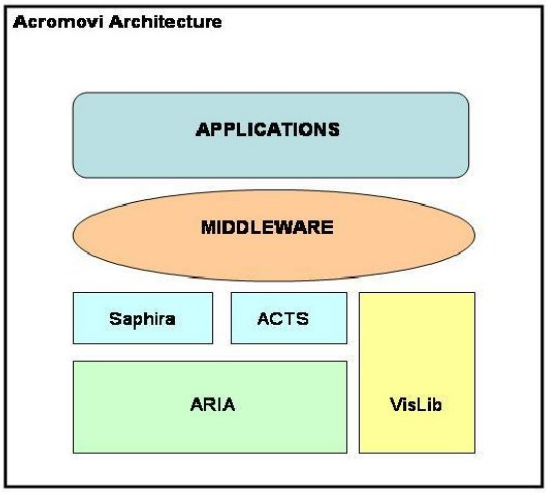


Fig. 2. General diagram of the Acromovi architecture.

This middleware level has also been divided into two layers. In the lower layer, there are a set of components, such as the sonar, the laser, the base, and so on. The others are special components that offer services to the upper layer, as the vision, the navigation or the localization.
These components only perform two kinds of functions: requests processing and results sending. Thus, they could be implemented like software components. But because of reasons of efficiency, implementation facility and integration with the whole structure, they have been developed as agents that encapsulate the required functionality. But one should take into account that this implementation may vary in the future if needed.
The upper layer of the middleware comprises a great variety of embedded and supervising agents, e.g. agents that monitor the state of the agents/components of the lower layer, agents that provide services of subscription to a certain component, agents that arbitrate or block the access to a component, and any other type of agent which might offer a service that can be of interest for the whole architecture or for a particular application. These agents act as links between the applications and the agents that access the components of the robot.
The structure of the entire middleware layer can be seen in the Fig. 3.

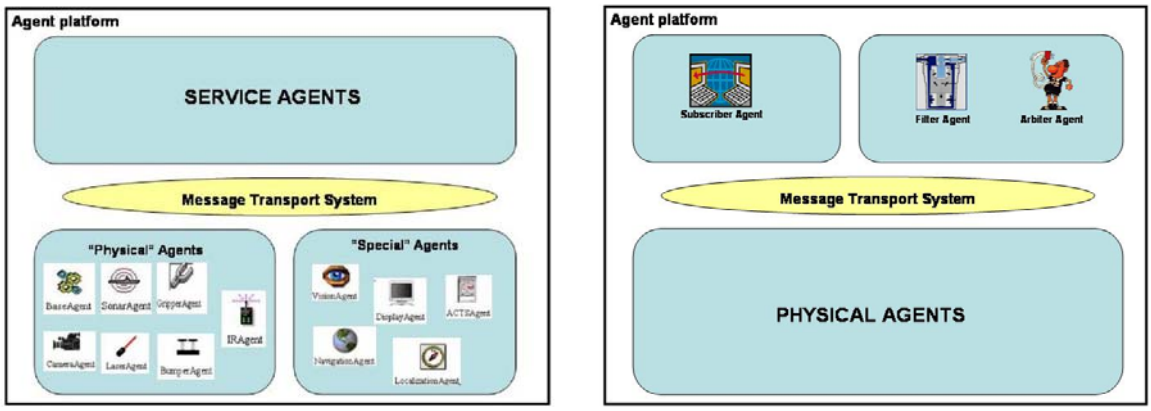
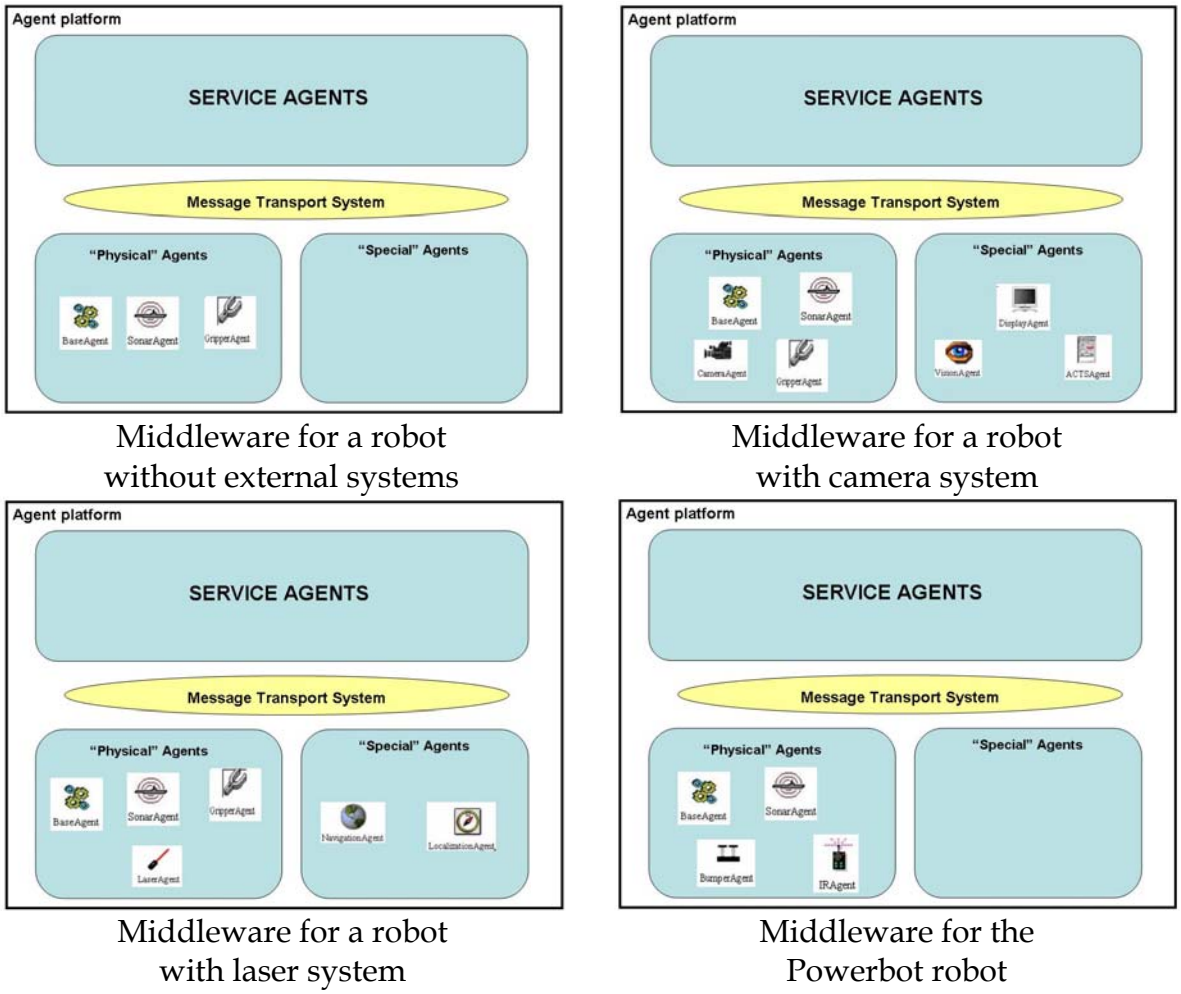


Fig. 3. The middleware layer: lower & upper layers.

The presented middleware has made feasible the change from an abstraction based on library functions for the handling of the robot -that are available in compilation and not consider multiplicity of systems- to an abstraction based on embedded agents -that are available in execution and natively distributed. One major design guideline was the distribution requirements and the communication between various platforms.
Because of the heterogeneous character of the team, there are different middleware layers for the robots, depending on the configuration of each robot of the team. These middleware layers are generated in execution time according to the elements that each robot has active at the moment of its execution. Therefore, each middleware layer will have active those agents that permit the access to the own components of the robot. In the Fig. 4, several examples of different configurations of the middleware layer are depicted. Only the agents of the lower layer are presented in

each configuration, due to the agents of the upper layer depends on these agents.



Middleware for a robot without external systems

Middleware for a robot with camera system

Middleware for a robot with laser system

Middleware for the Powerbot robot

Fig. 4. The different middleware layers of the Acromovi architecture.

The applications layer is on top of the middleware layer. These applications can be implemented as agents too. The applications, in order to access to the components of the robots, must communicate with the agents of the middleware, which then access to the bottom layer that controls the robot.

Once an application has been tested, and if it is useful for the whole architecture, it can be converted in a new agent of the upper layer of the middleware. Thus, each application done can increase our system to make applications more difficult and more interesting, following a bottom-up design.

4.2 Implementation of the architecture

Due to the fact that the middleware is just a set of embedded agents, a multiagent systems programming tool has been selected for its implementation.

The multiagent tool chosen was JADE (Java Agent DEvelopment Framework). It is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. It is fully implemented in Java language and simplifies the implementation of multiagent systems through a middle-ware and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across

machines (which not even need to share the same OS). The JADE middleware implements an agent platform for execution and a development framework. It provides some agent facilities as lifecycle management, naming service and yellow-page service, message transport and parsing service, and a library of FIPA interaction protocols ready to be used.
The agent platform can be distributed on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is basically a container of agents that provides a complete runtime environment for agent execution and allows several agents to concurrently execute on the same host.
The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent. The transport mechanism is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol.
JADE supports also scheduling of cooperative behaviours, where JADE schedules these tasks in a light and effective way. The runtime includes also some ready to use behaviours for the most common tasks in agent programming, such as FIPA interaction protocols, waking under a certain condition, and structuring complex tasks as aggregations of simpler ones
Having all in mind, the Acromovi architecture is implemented. Each robot of the team is a main container, thus, there are seven main containers, one in each robot that works as host of the distributed network. In each container, there is a group of agents. These agents are created in execution time depending on the robot configuration. Each of these agents represents one of the elements that in that moment has active the robot. So, in that way, the heterogeneity mentioned in the generation of the middleware layer of Acromovi is implemented. All this, can be seen in the Fig. 5.

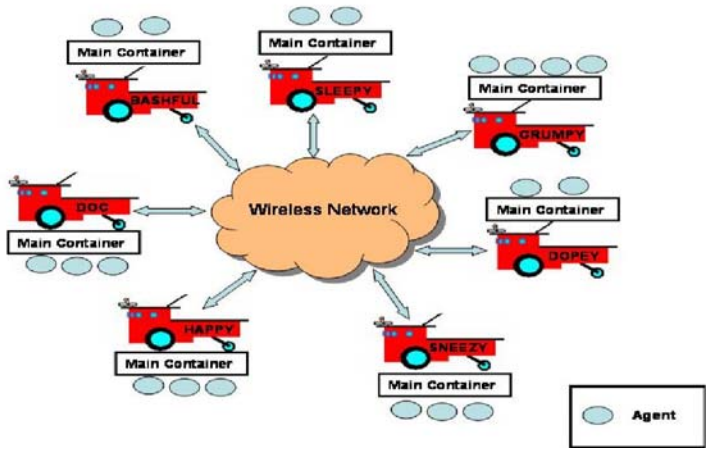


Fig. 5. Structure of the architecture implementation.

Because of the fact that ARIA and Saphira are implemented in C++ and JADE in Java, it has been necessary to use a mechanism to integrate such different programming languages. For this reason, JNI (Java Native Interface) has been used, thus allowing managing C++ code in Java programs.

The robot elements are represented and managed by agents. These agents communicate with the native layer of the global architecture by means of JNI.

5. Examples of applications developed with the Acromovi architecture

In this section, some applications implemented and tested with the Acromovi architecture are explained. These applications demonstrate that the Acromovi architecture is suitable for such tasks for it was developed. Also, they demonstrate the rapid development capabilities and ease-of-use of the agents implemented with this framework. Only the main idea and the operation of each application are shown here. For more details, consult the bibliography annexed.

5.1 Remote vision

This application consists of a graphical user interface (GUI), in Fig. 6, which allows the user to send commands to the vision and camera agents and displays the results. Such commands include the capture and display of the current image grabbed by the robot camera, the pan/tilt/zoom motions of the camera, and the image processing operations provided by the Vislib and Java 2D libraries. Such libraries provide low-level operations like thresholding, blurring, edge detection, as well as high-level ones as colour or motion tracking.

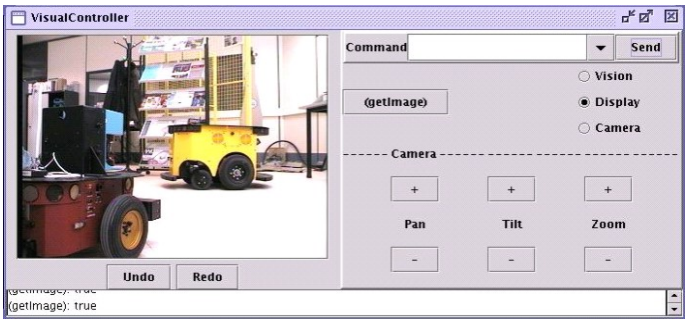


Fig. 6. Graphical user interface of the remote vision application

The graphical interface is built around pure Java Swing components, thus resulting in a cross-platform application, capable of running in any operating system running the Java virtual machine.

The remote vision GUI just sends messages to the agents, which take care of the operations in the own robot. The agents return the appropriate result in another message to the GUI agent, which then displays the image (in grabbing and image processing operations) or just indicates whether the operation has been made correctly or not, in the case, e.g., of camera motions.

The main drawback of this application is responsiveness. Agent communications of images is quite surely not the best efficient way of sending data through a network. However, flexibility and ease of use are its main advantages.

5.2 Robot following

In this application one robot equipped with local vision pursuits another one. The leader robot is assumed to navigate through an indoor environment, controlled either by a user or by its own navigation algorithms. The follower robot uses its PTZ

vision system to determine the position and orientation of the leader robot, by means of a colour pattern target, in Fig. 7, consisting of three rectangles (Renaud et al., 2004).

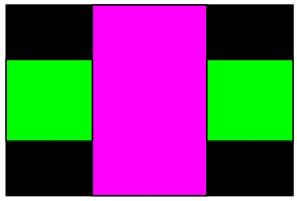


Fig. 7. Target used in the robot following application.

Such target, together with the local visual parameters, allows to easily computing the distance and relative orientation of the leader robot with regard to the follower-observer one.
An example of execution is depicted in Fig. 8. The application may be extended to more robots in a line formation, provided that each robot follows its predecessor, and it is equipped with a vision system.

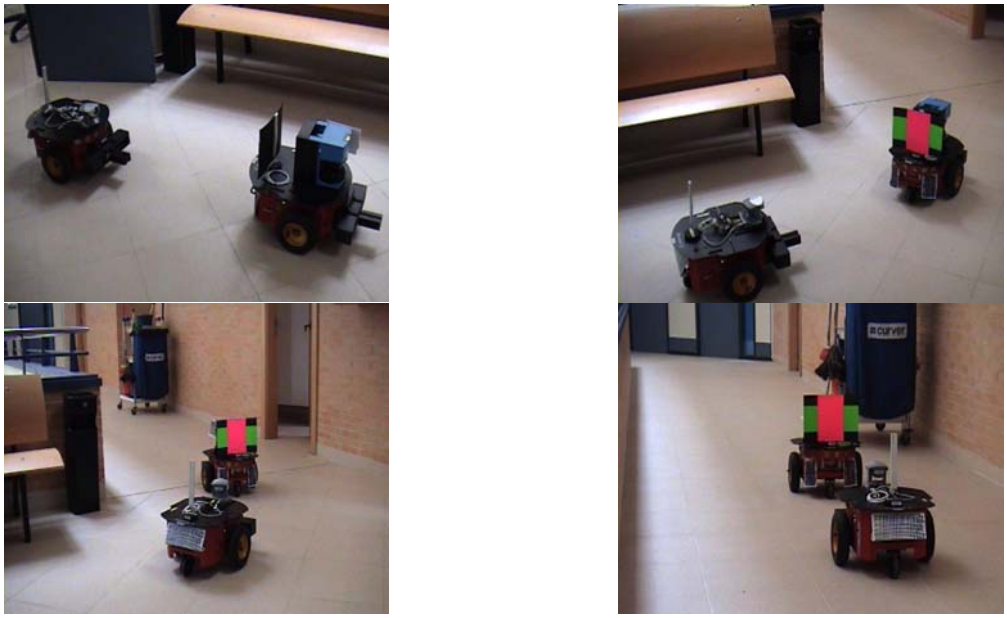


Fig. 8. Some images from a robot following experiment

Colour tracking is provided by the ACTS server, which needs to be trained with the colours of the target. The ACTS Agent works as link between the ACTS server and the Robot Following Application.
The application always tries to have centred the target in the centre of the image (and thus, always visible for the follower robot). To do so, it determines the blob centroid, computes the motion needed by the camera and compensates the camera and robot motions.
When the target is centred, the application calculates the leader robot orientation and position. With these information is easy to calculate the path to the target of the leader robot by means of a Bézier curve, with two intermediate Bézier's curve control points and the initial and final points (the positions of the follower and leader

robots).
When the curve is defined, motion commands are sent to the robot to perform the real motion.
In Fig. 9, the trajectory of two robots in an indoor environment is shown. The follower robot (red trajectory, named Happy) follows a smoother trajectory than that of the leader robot (blue, named Sneezy), as a result of the algorithm based on Bézier curves.

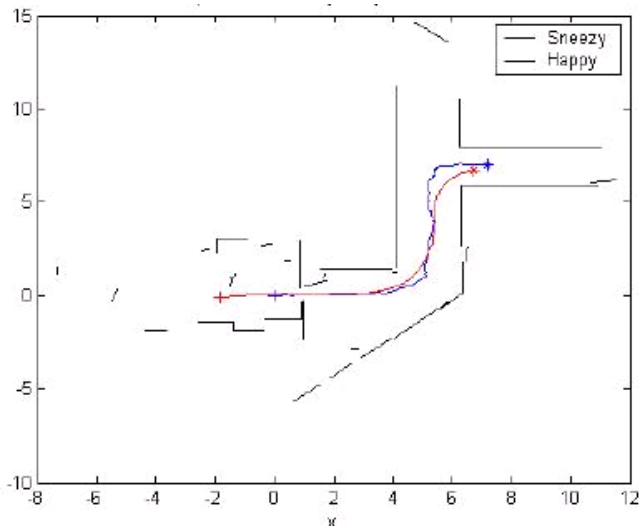


Fig. 9. Trajectory followed by two robots in a real robot following run

It is important to remark that the whole process does adds a very small processing overhead, thus demonstrating the capabilities of Java-based agent architectures for fast real-time vision-based tasks, as long as no image transmission is needed.
Also, it is important to point out that the system is robust enough to respond to sudden changes in the position or orientation of the leader robot, thanks to the real-time tracking system, and the small processing overhead.

5.3 Optical flow navigation
The purpose of this application is to use the optical flow technique to allow a mobile robot equipped with a colour camera navigates in a secure way through an indoor environment without collide with any obstacle. It reacts to the environment by itself. The objective is that the robot be able to avoid obstacles in real-time (Nebot & Cervera, 2005b).
The optical flow is the distribution of the apparent velocities of movement of the brightness pattern in an image, and arises from the relative movement of some objects and a viewer. As the robot is moving around, although the environment is static, there is a relative movement between the objects and the camera onboard the robot.
The visual information coming from the camera of the robot is processed through the optical flow technique, which provides the information the robot needs to safely navigate in an unknown working-environment.
From the optical flow generated in this way the robot can get an inference on how far an object present in a scene is. This information is embedded in the time to crash (or

time to collision) calculated from the optical flow field. Using only optical measurements, and without knowing one's own velocity or distance from a surface, it is possible to determine when contact with a visible surface will be made.
The application, first of all, must to capture the input image frames. The video from the camera on board the robot is captured using the Java Media Framework (JMF). Next, the optical flow derived from the new image is calculated, and from it, the time-to-contact vector is calculated. In Fig. 10, there is an example of the optical-flow field and the time-to-contact graphic.

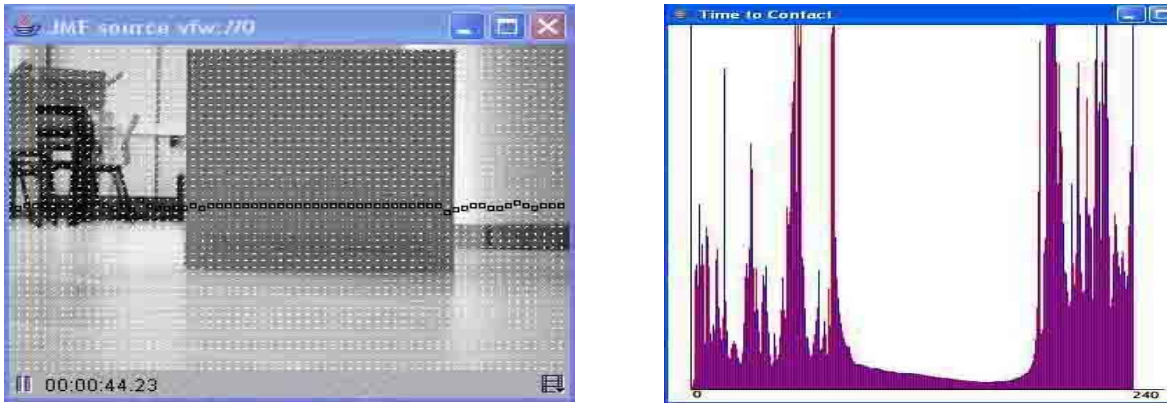


Fig. 10. Frame with the optical flow and the time-to-contact graphic associated

Once the time-to-contact vector is calculated, an agent decides which situation is done in the environment, and depending on the situation decides the movements associated to perform by the robot.
In this case, the experimental results have shown that the robot is effectively able to avoid any obstacle, in real-time, based only on the information from the optical-flow and the time-to-contact agents.

6. Conclusion

In this work, the Acromovi architecture has been explained. This architecture is the support for all the programming and interaction of the agents that manages our team of robots.
The Acromovi architecture is basically an agent-based distributed architecture for the programming and controlling of teams of mobile robots. This architecture allows the use of native components, by means of agent wrappers. The agents add the distributed capabilities, with message interchange for cooperation between robots. This makes possible an important reutilization of code, this being a basic principle in software engineering.
Also, it allows the scalability of the system. That is, if one tested application is of interest for the whole system, it can easily converted in a new agent of the architecture, to serve as basis for new applications more complex.
Moreover, it implements another concept, the resources sharing. This means that all the elements of one of the robots of the team can be easily accessed by all the other robots of the teams by means of the agents that control each of the other robots.
Finally, Acromovi architecture allows the quick development of multirobot applications with the advantages of the multiagent systems. The users need very

little time to develop new applications for the team of robots. In just a few weeks, a student with very low Java skills is able to program agents and develop the necessary testing procedures. In this aspect also is important the first concept of the framework, the reusability of the agents previously implemented.
For these reasons, this architecture is very appropriate for the implementation and execution of tasks that require collaboration or coordination by part of the robots of a team.
As future work, it has been now implemented a new application for the team of robots. This application tries to form multirobot formations. Taking the example of the robot following, robots in line formation, it is possible that the each follower robot could internally add a fixed displacement to its leader position. As can be in Fig. 11, each robot computes a virtual point instead of the leader position. These virtual points can be calculated simply displacing the leader position in the correct form. Now, the followers do not follow the leader, but these virtual points.

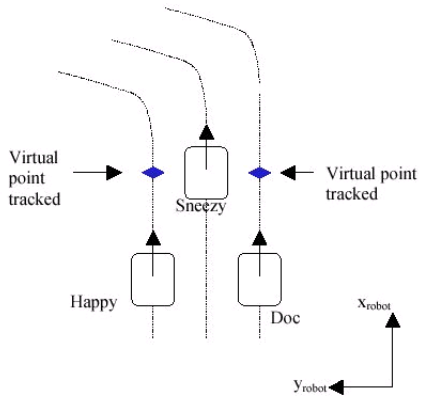


Fig. 11. Triangular multirobot formation using virtual points.

At the end, an interesting application domain would be the use of a team of robots to perform automatic, distributed surveillance tasks in indoor environments, or tasks of vigilance in buildings like factories.

7. Acknowledgements

Support for this research is provided in part by ”Ministerio de Educaci´on y Ciencia”, grant DPI2005-08203-C02-01, and by ”Generalitat Valenciana”, grant GV05/137.

8. References

Arai, T.; Pagello, E. & Parker, L. E. (2002). Guest Editorial: Advances in MultiRobot Systems. *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 5, October 2002, 655-661, 1042-296X.

Cáceres, D.A.; Martínez, H.; Zamora, M.A. & Balibrea, L.M.T. (2003). A Real-Time Framework for Robotics Software, *Proceedings of Int. Conf. on Computer Integrated Manufacturing (CIM-03)*, Wisla (Poland), 2003.

Caloud, P., et al. (1990). Indoor Automation with many Mobile Robots, *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'90)*, pp. 67-72, Ibaraki (Japan), July 1990.

Cao, Y.U.; Fukunaga, A.S. & Kahng, A.B. (1997). Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, Vol. 4, No. 1, March 1997, 7-23, 0929-5593.

Enderle, S.; Utz, H.; Sablatnög, S.; Simon, S.; Kraetzschmar, G.K. & Palm, G. (2002). Miro: Middleware for Mobile Robot Applications. *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 4, August 2002, 493-497, 1042-296X.

Fukuda, T. & Iritani, G. (1995). Construction Mechanism of Group Behavior with Cooperation, *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'95)*, pp. 535-542, 0-8186-7108-4, Pittsburgh (USA), August 1995.

Johnson, J. & Sugisaka, M. (2000). Complexity Science for the Design of Swarm Robot Control Systems, *Proceedings of 26th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pp. 695-700, 0-7803-6456-2, Nagoya (Japan), October 2000.

Jung, D. & Zelinsky, A. (1999). An Architecture for Distributed Cooperative Planning in a Behaviour-based Multirobot System. *Journal of Robotics and Autonomous Systems (RA&S)*, Vol. 26, No. 2/3, February 1999, 149-174, 0921-8890.

Mataric, M. J. (1998). New Directions: Robotics: Coordination and Learning in Multirobot Systems. *IEEE Intelligent Systems*, Vol. 13, No. 2, April 1998, 6-8, 1094-7167.

Nebot, P. & Cervera, E. (2005a). A Framework for the Development of Cooperative Robotic Applications, *Proceedings of 12th International Conference on Advanced Robotics (ICAR'05)*, pp. 901–906, 0-7803-9178-0, Seattle (USA), July 2005.

Nebot, P. & Cervera, E. (2005b). Optical Flow Navigation over Acromovi Architecture, *Proceedings of 15th International Symposium on Measurement and Control in Robotics (ISMCR 2005)*, Brussels (Belgium), November 2005.

Oller, A.; del Acebo, E. & de la Rosa, J.Ll. (1999). Architecture for Co-operative Dynamical Physical Systems, *Proceedings of 9th European Workshop on Multi-Agent Systems (MAAMAW'99)*, Valencia (Spain), July 1999.

Parker ,L. E. (1998). ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 2, April 1998, 220-240, 0882-4967.

Parker, L.E. (2000). Current State of the Art in Distributed Autonomous Mobile Robotics, *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS2000)*, pp. 3-14, 4-431-70295-4, Knoxville (USA), October 2000.

Picault, S. & Drogoul, A. (2000). The Microbes Project, an Experimental Approach towards Open Collective Robotics, *Proceedings of 5th International Symposium on Distributed Autonomous Robotic Systems (DARS'2000)*, pp. 481-482, 4-431-70295-4, Knoxville (USA), October 2000.

Renaud, P.; Cervera, E. & Martinet, P. (2004). Towards a Reliable Vision-based Mobile Robot Formation Control. *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS045)*, pp. 3176-3181, 0-7803-8464-4, Sendai (Japan), September 2004.