Proceedings of the 2005 IEEE
International Conference on Robotics and Automation
Barcelona, Spain, April 2005

Agent-based Application Framework for Multiple Mobile Robots Cooperation*

Patricio Nebot and Enric Cervera
Robotics Intelligence Lab
Jaume-I University
Castelló, Spain
{*pnebot, ecervera*}*@uji.es*

***Abstract*— We present an agent-based framework for the development of distributed applications for a team of heterogenous mobile robots. The main goal is to ease the development of cooperative tasks among teams of robots. The proposed architecture must have not only the capacity to allow the team of robots to accomplish such tasks, but also it must have tools and features that allow programmers to develop complex applications in a reasonable time. Agent-based development platforms are used for this purpose, integrating the necessary capabilities for developing distributed applications, and managing the communications among all the components.**

To demonstrate the advantages of the proposed framework, two real applications, remote vision and robot following, are presented. Both of them consist of a set of agents, and run actually in real robot systems, demonstrating that distributed systems do not add significant overhead to real-time tasks.

***Index Terms*— Mobile robots software reuse rapid-application development cooperation**

I. Introduction

The presented agent-based framework (*Acromovi* - acronym in Spanish for Cooperative Architecture for Intelligent Mobile Robots) was born from the idea that teamwork is an essential capability for a group of multiple mobile robots [1], [2]. It is known 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.

Many multirobot architectures have been proposed in the recent years. *ALLIANCE* is an architecture oriented to the cooperation of a small-medium team of heterogeneous robots, with little communication among them [3]. 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 behaviors or sets of behaviors to make their tasks.

DPA is another architecture for dynamic physical agents, well suited for robotics [4]. 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.

ABBA (Architecture for Behaviour Based Agents) [1] is another architecture for mobile robots, whose purpose is to design an architecture to model the robot's behavior so that it can select reactive behaviors of low level for his survival, to plan high level objectives oriented to sequences of behaviors, to carry out spatial and topological navigation, and to plan cooperative behaviors with other agents.

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, adding the versatility and power of the multiagent systems for the resolution of cooperative tasks for a group of heterogeneous robots.

In addition, we emphasize the reusability of software, allowing the programmer to seamlessly integrate native software components (vision libraries, navigation and localization modules) in the Java-programmed agent-based framework. In this way, powerful distributed applications can be rapidly developed.

Acromovi architecture is a framework for application development, which 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.

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 [5]. Among Java strengths, those particularly pursued were the high-level communication capabilities, and the native interface to existing C/C++ code.

This article describes the *Acromovi* architecture and its constitutive agents, as well as two applications, Remote Vision and Robot Following. These applications demonstrate the rapid development capabilities and ease-of-use of agents implemented with this framework.

Acromovi architecture is a distributed architecture that works as a middleware of another global architecture for programming robots. It has been implemented by means of the MadKit (Multi-Agent Development Kit) multiagent

*This work is partially supported by NSF Grant #2003168 to H. Simpson and CNSF Grant #9972988 to M. King

systems framework, a Java-based library of generic agents [6].

The 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 first distributed application, Remote Vision, allows to manipulate the camera, to capture and to process the images from another machine and even from another robot. This application allows to calibrate and to move the camera, being able to obtain new images on the user demand.

Using *Acromovi* architecture we were able to develop the second application, Robot Following, in a very short time, thanks to its flexibility. By using a small set of agents, with specified roles, and communicating with other general agents previously defined in our system, the development of such application, and other ones with a medium-high level of complexity, is very fast.

In Section II an overview of the *Acromovi* architecture is given. The different agents that constitute the *Acromovi* architecture and their functions are then described in Section III. The Visual Controller and Robot Following applications are finally described in Sections IV and V, before concluding and presenting final remarks.

II. Acromovi Architecture Overview

Acromovi architecture is a distributed architecture for programming and control a team of multiple heterogeneous mobile robots that are capable to cooperate between them and with people to achieve tasks of service in daily environments [7].

The robots that constitute the team have already a programming architecture with native (C/C++) libraries for all their accessories. This architecture is constituted by 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 [8].

Also, the architecture is able to subsume any other extra software, like the ACTS (a color 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 applications track up to colored objects which color the users have been selected previously. ACTS is an external program to the architecture that works as a server, and the applications work as clients that communicate with ACTS server by means of an ARIA library. In the other hand, 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

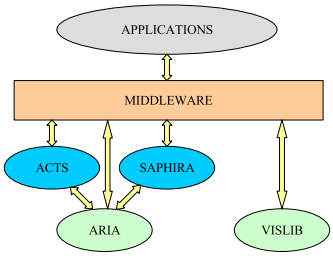


Fig. 1. General architecture diagram

with other distributed agents, either running on remote applications, or even as applets in web pages.

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.

As can be seen in Fig. 1, *Acromovi* architecture is a middleware between this robot architecture and the applications, that allows the collaboration and cooperation of the robots in the team. This middleware is based on an agent oriented focus.

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 kind 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 is still in development. This layer 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,

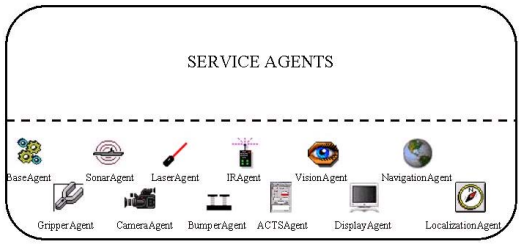
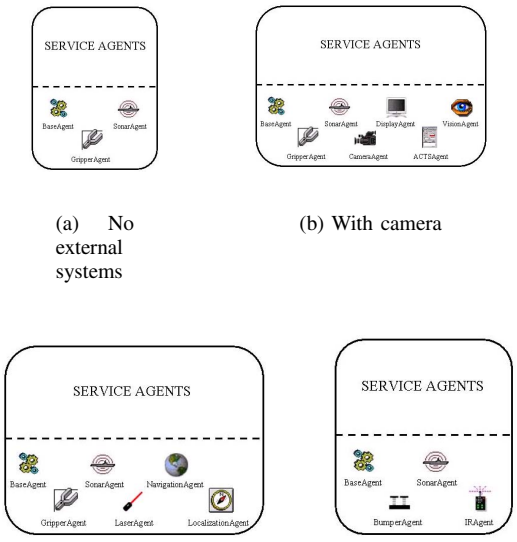


Fig. 2. The middleware layer



(a) No external systems

(b) With camera

(c) With laser

(d) Powerbot

Fig. 3. Different middleware layers.

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. 2.

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. These different configurations can be seen in Fig. 3.

Due to the fact that the middleware is just a set of embedded agents, the MadKit(Multi-Agent Development Kit) multiagent systems programming tool has been selected [6] for its implementation.

MadKit is based on a organizational model called *Aalaadin*, based on three concepts: agent, group and role. MadKit also implements the community concept.

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

Since our facilities include seven mobile robots, the middleware layer is formed by a community of agents called "Seven Dwarfs", that can be seen in Fig. 4. In this community, one active robot defines a group of its elements, which are represented and managed by agents. These agents communicate with the native layer of the global architecture by means of JNI. A brief explanation of the agents is given in the next section.

Over the middleware layer is the applications layer, that can be implemented as agents too. These applications, 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.

III. Acromovi agents

The agents that constitute the *Acromovi* architecture are described in the following paragraphs. These agents are divided in three groups depending on the part of the robot in which the agent carries out its task: body agents, laser agents and vision agents.

A. Body agents

These agents are the basic agents for the operation of the robot and they correspond to the main elements of the robots. There are five body agents: the Base Agent, the Gripper Agent, the Sonar Agent, the Bumper Agent and the IR Agent.

Base Agent: agent responsible of the physical operation of the robot, all about the motion of the motors and internal state of the robot. This agent is the most important because it is the link between all the rest of the agents and the physical robot. Thus, it must be launched in advance so that all the rest of agents can work.

Gripper Agent: agent that makes that the gripper works in a proper way and permits to other agents the manipulation of the gripper. The gripper is a two degree of freedom manipulator attached to the front of the robot and contains in its paddles a grip-sensor and infrared breakbeam switches in front and rear to sense objects and their positions.

Sonar Agent: it makes the sonar work correctly and returns the appropriate sonar values when these are required by other agents or applications. The sonars consist of transducers that provide range information for collision avoidance, localization, and navigation. Measurements range from ten centimeters to over four meters.

Bumper Agent: agent that manages the bumpers of a robot. It permits know if a robot disposes of bumpers, how many of them, and in which position. Also, it allows to recognize which bumper has been triggered. By the

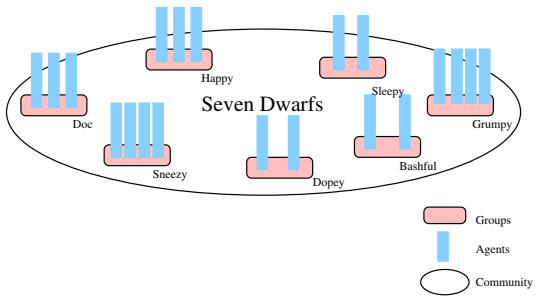


Fig. 4. Structure of the "Seven Dwarfs" community

moment, the only robot of the team that uses this agent is the PowerBot.

IR Agent: agent that is in charge of the management of the 4 IR sensors that are disposed in the front of the Powerbot. These sensors has the function to prevent the robot of ledges such as stairways or holes. The agent indicates the state of each sensor, indicating whether it is triggered or not.

It is important to emphasize that bumper and IR sensors are sensors for the safe navigation of the robot, and they respond at two levels. First, the activation of one of this sensors causes the robot's controller to take a certain action, by the moment it stops the motor and sets the brakes, but this behavior can be changed. At the next level, the controller informs about the condition that has produced the previous action.

B. Laser agents

These agents comunicate with the laser rangefinder, using the ARIA interface, and permit to use two modules of Saphira: Localization and Navigation. These two modules are powerful native libraries integrated into Saphira, using a Montecarlo-Markov method for robot's localization and a Gradient system for the navigation. All their functionality is readily subsumed by the embedded agent.

Laser Agent: agent that allows multiple operations with the laser such logging its lectures, determining the distance to an object with better precision than the sonar, etc. With the tools and primitives provided by the Laser Agent, we are able to do tasks like mapping, environment scout and so on.

Localization Agent: agent that provides the robot a very good localization in a known environment, i.e. a map loaded previously in the robot. This agent also can be used in combination with the sonar sensors. This agent provides the functions needed to load a map in the robot.

Navigation Agent: with the previous two agents -Laser and Localization- the Navigation Agent calculates the path between two points. It also includes a collision-avoidance system and real-time path calculation, that allows the robot to select the best path, avoiding dynamic obstacles in the environment. This agent uses the map loaded in the Localization Agent.

C. Vision agents

Acromovi architecture is increased with the vision agents, who capture images for the robot camera. The vision agents divide themselves in four parts: the Camera Agent, the Acts Agent, the Vision Agent and the Display Agent.

Camera Agent: agent which moves physically the camera (pan, tilt and zoom) and informs about its state. Thus, although the camera is an external device, it can be integrated how any other element (sonar, laser navigation, gripper, etc). The camera is a Pan-Tilt-Zoom camera Canon VC-C4.

ACTS Agent: agent used for color tracking of objects by means of the ACTS system. It provides information about these objects as can be area, bounding box or centroid.

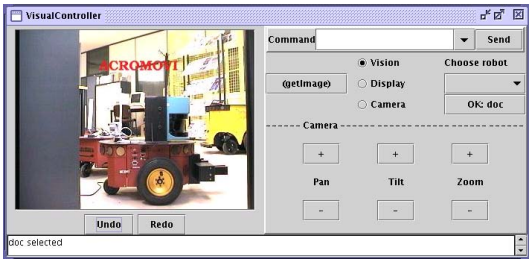


Fig. 5. Visual application interface

This agent makes use of the ARIA class that permits the communication with the ACTS server.

Vision Agent: agent that acquires images by the camera, to visualize and modify them. It makes use of the Vislib. Since Vislib is written in the C language, Vision Agent follows the same mechanism explained for the case of the access to the ARIA library, the use of JNI.

Display Agent: agent that modifies and to process operations on the captured image by the Vision Agent, with the Vislib library. These operations are implemented with Java AWT (Abstract Window Toolkit) y Java 2D [9].

IV. Application: Remote Vision

This application consists of a graphical user interface (GUI), that can be seen in Fig. 5, which allows the user to send commands to the vision and camera agents and display 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.

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. Such agent interaction is intuitively depicted in Fig. 6.

When the Camera Agent receives the message of the Remote Vision Agent, it calls to the corresponding function of the native ARIA library to move the camera. After doing

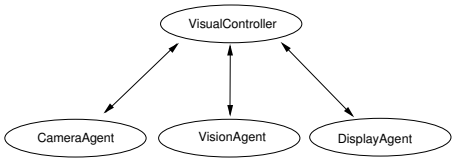


Fig. 6. Agents communication in the Remote Vision application

so, the result of this function is returned back to the calling agent.

The Vision Agent works similarly, by executing the operation described in the message, which in this case is translated to its equivalent one in the native Vislib library. Again, the result of the native function is returned to the calling agent.

Finally, the Display Agent executes operations related to Java 2D libraries. Such operations involve image grabbing and processing, and the result is a new image which is returned back to the Remote Vision Agent for Display.

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.

As interesting future extensions, we would like to explore the capabilities of Java for image compression and continuous video transmission, to enable the Remote Vision agent to grab continuous video in real time.

V. Application: 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 color pattern. Such pattern, together with the local visual parameters, allows to easily compute the distance and relative orientation of the leader robot with regard to the follower-observer one.

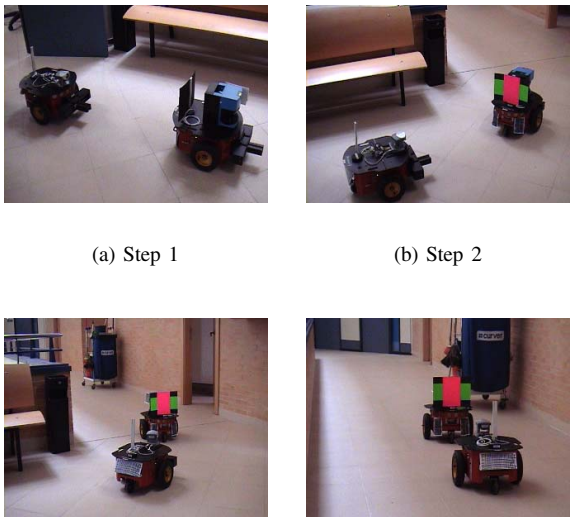
Information exchange results as a consequence of the motion of the robots. Both robots may agree initially in a common value of their velocities. Alternatively, the follower robot may send stop/restart orders to the leader robot, in case that the distance between them grows too much. Such strategies are easily implemented as message protocols between the controller agents of both robots.

An example of execution is depicted in Fig. 7. The method 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.

Also, it is possible that the robot could internally add a fixed displacement to its leader position, thus allowing robot formations other than lines to be easily deployed. As depicted in Fig. 8, 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.

Color tracking is provided by the ACTS server, which needs to be trained with the colors of the target. The Acts Agent works as link between the ACTS server and the Robot Following Application.

A number of agents are directly (in one or another way) involved in this application. A diagram of their interactions is depicted in Fig. 9.



(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

Fig. 7. Robot Following experiment.

The first agent implied in the process is the BlobCenter, whose role is to coordinate the ACTS, Base, and Camera Agents. Its goal is to keep the target visible. To do so, it determines the blob centroid, by means of a message received from ACTS Agent, computes the motion needed by the camera and compensates the camera and robot motions, sending messages to Base and Camera Agents.

The Position Agent calculates the robot orientation and position. When the target is centered on the image (it sends a message to the BlobCenter to know it), it sends messages to the Camera and ACTS Agents to determine the camera position (pan and tilt) and the different blob areas. With this information, it calculates the distance to the target and its orientation, as well as the camera orientation. Those data is forwarded to the Bézier Agent.

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

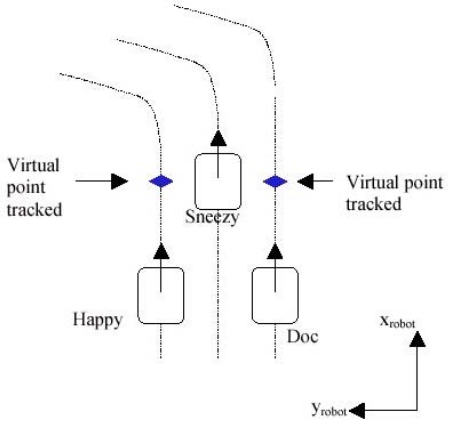


Fig. 8. Triangular formation using virtual points

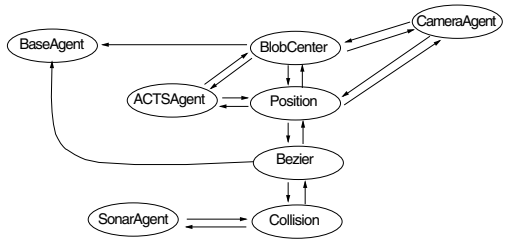


Fig. 9. Agents communication in the Robot Following Application

long as no image transmission is needed.

The Bézier agent determines a path to the target, defined by a Bézier curve. It determines two intermediate Bézier's curve control points and the initial and final points (the positions of the follower and leader robots) [10]. Motion commands are then sent to the Base Agent to perform the real motion of the robot. 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.

Last but not least, in the background, the Collision Agent supervises the robot's path, communicating with the Bézier agent and Sonar Agent. By communicating with the Sonar Agent, it is able to detect the presence of obstacles in the robot's path. When an obstacle is getting closer the robot, the agent sends a message to the Bézier agent, which moderates the robot's speed, stopping the robot if necessary, sending the corresponding message to the Base Agent.

As a future extension, another agent could calculate alternative paths to avoid obstacles maintaining the previous curve marked by the Bézier agent, by e.g. adding new intermediate control points.

In Fig. 10, 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.

VI. Conclusion

Reutilization is a basic principle in software engineering. Our framework enables the use of native components, by means of agent wrappers, providing the programmer with a huge set of preprogrammed, tested and efficient tools for mobile robots. The agents add the distributed capabilities, with message interchange for cooperation between robots.

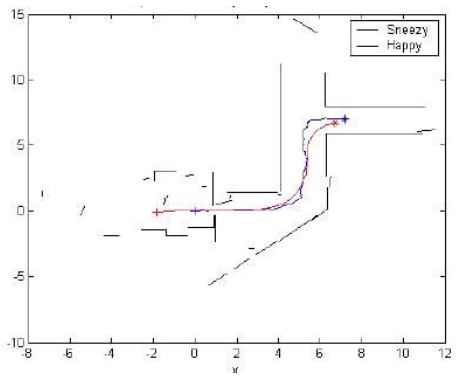


Fig. 10. Trajectory in a real robot following run.

Acromovi architecture, the presented framework, allows the quick development of multirobot applications, by joining the advantages of the distributed computation and multiagent systems. This architecture is thus very appropriate for the implementation and execution of tasks that require collaboration or coordination by part of the robots of the team.

Such as architecture enables us also to share characteristics of a robot among all the team, with the correspondent increase of the robot performance. Moreover, the *Acromovi* agents permit an easy access to the elements of the robot by the applications.

One of the most interesting point is the time needed by users to develop an application of this kind. In just a few weeks, a student with very low Java skills, is able to program agents and develop the necessary testing procedures.

Future extensions of this work include the study of efficiency in image applications and communications.

Acknowledgments

This paper describes research carried out at the Robotic Intelligence Laboratory of Universitat Jaume-I. Support for this research is provided by the Ministerio de Ciencia y Tecnologia under projects DPI2001-3801, HF2001-0112, and FIT-020100-2003-592 (PROFIT), by the Generalitat Valenciana under projects inf01-27, GV01-244, CTIDIA/2002/195, and by the Fundacio Caixa-Castello under project P1-1B2001-28.

References

[1] D. Jung and A. Zelinsky, "An architecture for distributed cooperative planning in a behaviour-based multi-robot system," *Journal of Robots and Autonomous Systems*, vol. 26, no. 2–3, pp. 149–174, 1999.

[2] M. J. Mataric, "New directions: Robotics: Coordination and learning in multirobot systems," *IEEE Intelligent Systems*, vol. 13, no. 2, pp. 6–8, 1998.

[3] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multi-robot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[4] A. Oller, E. del Acebo, and J. de la Rosa, "Architecture for cooperative dynamical physical systems," in *9th European Workshop on Multi-Agent Systems*, 1999.

[5] P. Nebot, D. Gomez, and E. Cervera, "Arquitectura distribuida para un equipo de robots móviles heterogéneos," in *Proc. CAEPIA - TTIA 2003*, 2003.

[6] J. Ferber and O. Gutknecht, "A meta-model for the analisys and design of organizations in multi-agent systems," in *Third International Conference on Multi-agent Systems(ICMAS'98) Proceedings*, 1998.

[7] P. Nebot, D. Gomez, and E. Cervera, "Agents for cooperative heterogeneous mobile robotics: a case study," in *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, 2003.

[8] K. Konolige and K. Myers, *The Saphira Architecture for Autonomous Mobile Robots*. MIT Press, 1998, ch. 9, pp. 211–242.

[9] L. H. Rodrigues, *Building Imaging Applications with Java Technology using AWT Imaging, Java 2D, and Java Advanced Imaging(JAI)*. Addison-Wesley, 2001.

[10] S. Y. Chiem. and E. Cervera, "Vision-based robot formations with bézier trajectories," in *Eighth Conference on Intelligent Autonomous System*, 2004.