Integrating individual and social intelligence into module-based agents without central coordinator

Bianca INNOCENTI [a],Beatriz LÓPEZ [a] and Joaquim SALVI [a]
[a] *Institute of Informatics and Applications, University of Girona*

Abstract. Robots are complex entities that can be modeled as multi-agent systems. The multi-agent paradigm provides an integrated intelligence framework such as a path planning agent that uses search techniques interacts with a fuzzy-based agent that moves the robot to a given location. Agent coordination is required to achieve the appropriate global behavior. When there is no central agent that coordinates the overall architecture, the intelligence required for social interaction should therefore be deployed at the agent level. In such a situation, individual intelligence (how to reach a goal) and social intelligence (how to collaborate or compete for resource with other agents) should be integrated at the agent level. In this paper we propose the use of module-based agents to achieve this integration. The whole multi-agent robot architecture, ARMADiCo, has been implemented with several module-based agents and tested on a Pioneer 2DX of ActivMedia. Some preliminary results are shown and discussed.

Keywords. Robotics, Agents

Introduction

Artificial Intelligence Robotics concentrates on how a mobile robot should handle unpredictable events in an unstructured world (conversely to Industrial Robotics that is concerned on dynamics and kinematics of a robot [10]). For this purpose, some researchers are involved in a long term effort to integrate perception, navigation, planning and uncertainty management methods in a single robot architecture. Traditionally, most researchers focus on a module based approach, in which each robot component is implemented in a module [2,1,15].

More recently, several researchers are concerning about the significant impact of agent technology on the world of robotics [8,7]. Most of them focus on the one-to-one mapping from robots to agents, while keeping the module approach implementation inside a robot. Trying to capture all robot capabilities in a single agent seems unfeasible, thus some of the recent architectures consider multi-agent approaches inside a single robot [12,3,13]. The modeling of the whole robot architecture in a higher abstract level, as a collection of agents (multi-agent system), facilitates the design [11,16].

One of the advantages of using agents is that they have their own autonomy to deploy their tasks. So, if an agent receives a request, it could do it or not depending on its current state and information (priorities, danger, etc.). One of the disadvantages, however, is the need of a coordination mechanism that takes into account the physical actuation of the robot in the environment, so that the final decisions that emerge from the agents interactions do not result in a folly robot behavior. Coordination can be achieved by a central arbiter, as in [3]. However, decentralized coordination can also be considered so that it avoids having a bottleneck agent that governs the overall architecture. This is an important issue when intelligent robots should have a considerable number of agents to cover all the aspects of intelligence.

Moreover, since robots concern physically grounded resources, this coordination should take into account possible disruptions in the robot behavior. Thus, the coordination mechanism should handle all the complexities involved in the resource exchange from the control of one agent to the other one.

Consistently, in a decentralized scenario, each agent has to deal locally with, at least, two kinds of intelligences: individual and social. On one hand, individual intelligence enables the agent to achieve its assigned goals (as for example, planning a trajectory for achieving a target point). On the other hand, social intelligence enables the agent to understand and manage with other agents.

In this paper we present a way of integrating both kinds of intelligence in a module-based agent belonging to a multi-agent robot architecture with no central coordinator. So, a module architecture is kept inside an agent, while the multi-agent approach is followed for the global architecture design. As a consequence, intelligence integration is achieved in two levels: at the agent and at the multi-agent levels. At the agent level, individual and social intelligence is integrated. Each agent follows the most adequate artificial intelligence approach in order to achieve its individual goal. There are some agents that use search methods to achieve their goals, while others employ probabilistic reasoning, fuzzy techniques and others. Each agent uses a utility based and fuzzy based reasoning approach to deal with social interactions. At the multi-agent level, all these artificial intelligence techniques are integrated and thus, as the final interaction of all the agents, the robot has an emergent behavior that achieves the missions proposed by humans.

From our understanding, there is no previous work in which a decentralized coordination is presented for controlling a single robot, as well as the inclusion of a coordination mechanism that takes into account the physically grounded characteristic of the robot resources. Thus, dealing with social coordination at the agent level is not a trivial task, and so, considering artificial intelligence techniques both, at individual and social levels, is necessary.

This paper is organized as follows. First, in section 1 the multi-agent architecture is outlined. Then, in section 2 the module-based architecture is described. Then we continue by giving some experimental results in section 3, some related work in section 4 and we end with the conclusions.

1. The multi-agent robot architecture with distributed coordination

ARMADiCo (Autonomous Robot Multi-agent Architecture with Distributed Coordination) can be described according to the main components required in classical Hybrid Deliberative/Reactive Architectures [10]. That is, the main components that any autonomous robot should include are the following: a deliberative component to reason about how to achieve high level goals; a reactive component to deal with the environment; perception and actuators to deal with the physical world (see Figure 1 top).

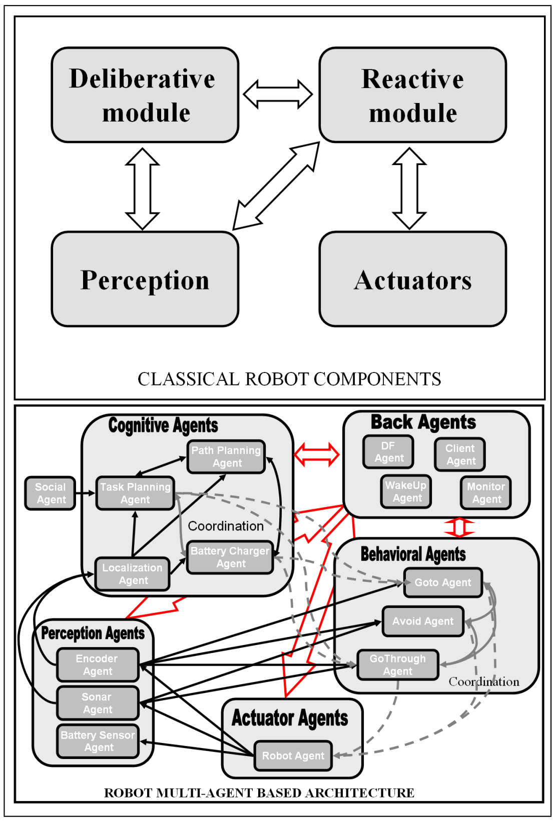

Figure 1. *Top:* Classical robot components. *Bottom:* A robot architecture based on a multi-agent system.

Moving downwards the next abstraction level, each component can be modeled by a set of agents. In Figure 1 (bottom) the previous component model is detailed in several agents that built up the different robot desired capabilities. First, perception is set up by a set of agents. An agent is designed for each sensor (encoder, sonar, battery sensor). Second, actuators are itemized in a set of physically grounded agents. Although a single robot actuator agent is shown in this figure, that deals with robot motors and sensor readings, more agents can be incorporated. For example, an agent that is in charge of a robot arm could be included. Third, the reactive capabilities are detailed in a set of behavioral agents. There is a behavioral agent for each basic behavior as for example go to a point, avoid obstacles, go through a narrow space, etc. Fourth, deliberative capacities are set up by cognitive agents as the task planning, the path planning, the battery charger and the localization agents. Much more agents could be added in order to provide to the robot with higher cognitive capabilities as learning, decision making, etc. Finally, an interface agent is defined to facilitate the robot interaction with either humans or other robots, and a set of back agents to deal with other functionalities required to give support to the overall multi-agent system.

ARMADiCo follows a distributed coordination approach to share the system resources. Being a distributed coordination mechanism, the global system behavior emerges from individual agents (micro level behaviors). For such kind of systems, design is still an open issue [4]. One of the current proposals consists in the use of agent's design patterns [17], and we have followed such approach in the design of ARMADiCo. The agent's design pattern captures common features of the agents and facilitates the incorporation of new agents in the architecture. Our current pattern follows the schema shown in Table 1.

Table 1. Agent's design pattern

- **Internal State:** Mechanism used by the agent in order to know about the progress of its goals, and to update the information of the environment.
- **Goal:**
  - **Goal configuration:** Agent's goals.
  - **Goal methods:** Methods that implements the agent's goals.
- **Competition:** List of possible conflicting agents due to resource sharing, and list of shared resources.
- **Collaboration:** List of agents from/to exchange messages (request, information).
- **Coordination**
  - **Utility computation:** Method (with the required parameters) used to compute the utility value for achieving a coordination agreement.
  - **Resource exchange:** Method used to exchange resources from one agent to another.
- **Helper methods** : All supporting methods that help the agent in registering in the system, communicating, starting up, etc. They are the same for all the agents.

In the agent's design pattern we can perceive information regarding resources (competition and collaboration slots). An agent in ARMADiCo could use a resource and could be a resource, depending on the situation. For example, when the goto agent sends a request to the robot agent to move the robot to a given linear and angular velocities, the robot agent is actuating as a resource. This resource is shared with the avoid agent which can, at the same time, send another linear and angular velocities to the robot agent in order to avoid some obstacle. In this case, we say that the robot agent is a physically grounded resource because its actuation modifies the environment. Conversely, when the task planning agent sends a trajectory to the goto agent, it is the goto agent which is acting as a resource. The battery charger agent could also send another trajectory to the goto agent at the same time. So, the goto agent is a shared resource of the task planning and the battery charger agents.

Figure 2 shows two agent's design pattern instances that illustrates these situations. The competitor agent for the task planning agent is the battery charger agent, and between brackets, the resources in conflict are shown. While the goto agent has as competitors the Avoid and GoThrough agents, regarding the robot agent as a resource. In this figure, it can also be seen that the interface, the localization and the battery charger agents are used as a resource by the task planning agent, but without conflict. For this reason, this kind of resource usage is pointed out in the collaboration slot.

Agent Pattern Design		Task Planning Agent	Goto Agent
Internal State		Maintain mission progress information	Maintain motion progress information
Goal:	Configuration	Achieve mission	Drive the robot to goal position
	Methods	Decomposition into tasks (procedural reasoning)	Fuzzy collaborative control system
Competition		Battery charger agent (path planning, goto)	Avoid, GoThrough agents (robot)
Collaboration		Interface, localization battery charger agents	Encoder, task planning battery charger agents
Coordi nation:	Utility Computation	Based on mission priority	Based on distance to goal position
	Resource Exchange	Trajectory merging	Fuzzy-based smoothing method
Helper Methods			

Figure 2. Pattern Design of two agents

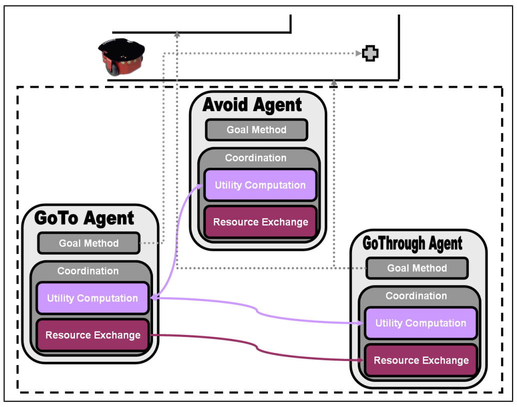

Figure 3. Example of module-based agent distributed coordination.

2. The module-based agent approach for integrated intelligence

In Figure 2 it is possible to distinguish the methods used by the agents both to achieve their goals and to coordinate when there are competitive resources. Agents in ARMADiCo can maintain different lines of reasoning at the same time according to their current states. For example, the goto agent, can continuously run its goal method in order to position the robot to a given target point, while coordinating with the avoid agent the control over the robot agent (shared resource), and executing a resource exchange with the goThrough agent (see Figure 3). For this purpose, a module-base approach has been followed as the basis of the architecture inside each agent.

The main modules in the architecture are the goal module (related to individual intelligence) and the coordination modules (related to social intelligence). An agent can have more than one goal module whenever it has more than one goal. Similarly, it can have more than one coordination module if it competes for resources with more than one agent.

2.1. Individual intelligence modules

The goal method of each agent specifies the kind of intelligence reasoning technique employed by the agent in order to fulfill its goal. Several techniques have been used, according to the different levels of reasoning required: cognitive, perceptual, and behavioral (see Figure 1).

2.1.1. Cognitive agents: Procedural reasoning, search and statistics methods

There are three agents with cognitive capabilities, that we want to highlight regarding the intelligent methods they use: the task planning agent, the path planning agent and the localization agent.

The task planning agent's goal is to plan the sequence of tasks to reach the robot's mission based on the information provided by the interface and the localization agents, and to assure that the mission is achieved. The method employed to decompose a mission into tasks is based on a procedural approach similar to PRS [5]. The task planning agent needs two robot resources: the path planning agent and the goto agent. On one hand, when the task planning agent has to deal with a positioning task, it requests to the path planning agent about a plan for moving from the current position to the destination one. On the other hand, as stated above, the task planning agent could request to the goto agent to follow a trajectory (see Figure 2).

The path planning agent has two main goals: the calculation of a free of non-moving obstacles trajectory to the goal, and the estimation of the energy consumption of the planned trajectory. The optimal trajectory calculation is obtained in two steps: first with a graph method to obtain a general sequence of destinations (considering only rooms and hallways), and second, with a grid method to find the path between two consecutive destinations (considering all the non-moving obstacles). In both methods a search algorithm is used. For the first one, we use the Dijkstra's algorithm and for the second one, the A* search algorithm. Once the trajectory is determined, the estimation of the energy consumption is made based on the cruising speed.

The localization agent's goal is to locate the robot in the global map. It receives information from the sonar agent and the encoder agent. In order to accomplish the agent's goal, a MonteCarlo technique is used to determine the position and the orientation of the robot. With this agent, the encoder agent can correct accumulative errors produced by the encoder's readings.

2.1.2. Behavioral agents: Fuzzy reasoning for the goto agent

The goto agent has the goal of driving the robot to a destination point, according to the trajectories requested by either the task planning or the battery charger agents. Given a desired position (x, y) and an orientation θ, and according to the actual position and heading (obtained thanks to the collaboration with the encoder agent), the goto agent calculates the linear and angular speeds to drive the robot to the destination position. Speeds are calculated using PID controllers. Actually, there are two different PID, one tuned to be very fast but unprecise and the other tuned to be very precise but slow. Both PID's are combined using a Sugeno fuzzy approach that takes into account the distance to the destination point. According to this distance the system determines if the robot is "far" or "close" to the destination; then, the Sugeno system outputs the desired speed as

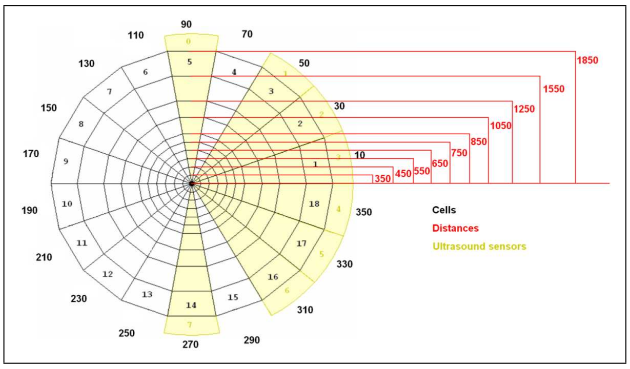

Figure 4. Local map of sonar agent.

a linear combination of both PID's speeds, without having to design a complex model based control system.

2.1.3. Perception agents: Probabilistic reasoning for the sonar agent

The sonar agent's goal is to create a local map to locate obstacles based on the readings of the 8 ultrasonic sensors following a probabilistic approach. It gets the different measurements from the robot agent and treats them in order to find the obstacles in the path of the robot. It also has to update a map as the robot moves on. To create the map, a zone around the robot is split into cells as shown in Figure 4. Cells are obtained dividing the circle around the robot in 18 circular sectors that represent the ultrasonic sensor visibility zones, and each circular sector in 10 parts representing different distances from the robot. This organization in cells is useful for dealing with noise and fictitious obstacles detected by ultrasonic sensors. If an object is detected in a cell several times, then the probability associated to the cell increments, indicating the presence of the object. So, each cell is labeled by a probability regarding the fact that an obstacle has been detected inside. Probabilities are incremented when an obstacle is sensed in the cell and decremented otherwise. In this way, we introduce some "memory" to sensors. At each sample time, this agent first applies the movement to the map (to move objects according to the robot motion), then updates the sonar information and sets it to the map. After that, and using probabilities, this agent finds the closest point (center of the cell where an object has been detected) to the robot and the closest point in front of the robot (they can be the same or not). Depending on the situation one or both points must be eluded, so their coordinates are sent to the avoid agent in order to be used for speed calculation.

2.2. Social intelligence modules

When a resource is shared by more than one agent, a conflict can arise. In order to coordinate shared resource usage, ARMADiCo uses a peer-to-peer coordination mechanism, that is, between the agent that is currently controlling the resource and the agent that wins the resource. No central arbiter decides upon the resource usage. Thus, agents needs to reason about coordination issues. Moreover, since robots concerns physically grounded resources, this coordination should take into account possible disruptions in the robot behavior.

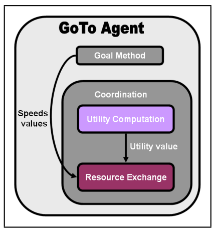

Figure 5. Goto Module interaction.

We propose, then, to split the coordination process in two different parts: winner determination, and resource exchange. In the former part, the agents that wish to use the resource determine, without any arbiter, who will use it. In the second part, the agent who wins the resource changes the current state of the resource to the desired one by avoiding undesired global behaviors of the robot.

2.2.1. Winner determination method

In case of conflict on a resource usage, each agent involved in the conflict computes an utility value of its action. The agent with the highest utility wins the resource. Each agent has its own utility method, but all the utility values are in the [0,1] interval, so they are comparable. For example, the goto agent calculates its utility u_g as a function of the distance that remains to reach the destination location. Similarly, the avoid agent calculates its utility u_a as a function of the distance to the closest obstacle. Both utilities can be compared, and the agent who has the highest one takes the control of the resource. The agent that is controlling the resource sends periodically its utility value to the other agents with whom share the resource. When one agent has a higher utility value, it informs the rest with this value and takes the control over the resource. Thus, our coordination mechanism is decentralized. For coordination issues, only the agents that share the resource communicate among themselves. It is important to note that only one agent at the time (the one that has the resource) sends messages to the other agents involved in the conflict; there is no broadcast to all the agents in the architecture.

2.2.2. Fuzzy-based method for resource exchange

As desired actions requested by the agent that looses the resource control and the agent that wins it can be very different, the winner agent needs to reason about how to perform the change from the current state of the resource to the desired one, avoiding undesired global behaviors. The change from one state to the other is performed based on a window time frame that depends on whether this change is critical or non-critical. For critical changes the window time frame depends on a fixed value (for example, the time to a collision) while for non-critical changes it is calculated using a Sugeno fuzzy system. The input variables of the system are the action differences (for example, the linear velocities differences); the output variable is the number of cycles needed to change from the current resource state to the desired one. The information related to velocities is provided by the goal module (see Figure 5).

Once the window time frame is calculated, a weighted mean is applied to compute the parameters that modify the state of the resource for each robot cycle, from 1 to the end time determined by the computed window frame. Weights vary along time depending on the utility values of the agents in conflict and the current fraction of time. The utility value of the winner agent (the agent in charge of performing the resource exchange) is obtained through the winner determination module while the other utility is modified according to a decreasing function (see more details in [6]).

3. System Demonstration

In order to test the synergies of the overall set of methods implemented both, at the agent and multi-agent levels, we have developed a prototype of ARMADiCo. For doing so, we have implemented an ad hoc multi-agent platform, programmed in C++ on Linux because the majority of the commercial platforms have an agent that centralizes the functioning of entire platform and because they are not capable of dealing with systems that need to respond in real time. The robot used for experimentation is a Pioneer 2DX of ActivMedia Robotics.

3.1. Experimental Setup

In order to demonstrate intelligence integration at the agent level and the different agents interactions, we propose the following experimental scenario: the robot is on room A and its goal is to move to room E (see Figure 6). The active agents in the platform are the interface agent, the path planning agent (search methods), the task planning agent (procedural reasoning), the goto agent (fuzzy method), the avoid agent, the sonar agent (probabilistic method), the encoder agent and the robot agent. The goto agent and the avoid agent are sharing the robot agent and they are using the fuzzy-based coordination to control the resource.

3.2. Results

In this scenario, the following agent's interactions happen. After receiving from a human operator the mission, the interface agent requests the desired destination (coordinates x_d,y_d and θ_d of Room E) to the task planning agent, and informs the initial position (coordinates x_i,y_i,θ_i of Room A) in the global map to the encoder agent. The task planning agent, recognizes the current robot mission as a simple positioning task, and it requests the desired position to the path planning agent. This latter agent, computes the trajectory needed to achieve the destination and informs back about it to the task planning agent. Then, the task planning agent sends this trajectory to the goto agent, which at the same time receives information about the current position from the encoder agent. Concurrently, the avoid agent receives information from the sonar agent, and coordinates with the goto agent the speeds commands to be sent to the robot agent. Finally, it is the robot agent the one that connects with the real robot to obtain the sensor readings and to execute the speeds commands. Figure 6 shows how this mission is achieved by the robot that follows a smooth trajectory, dodging the obstacles, crossing doors, and moving through a corridor.

Figure 7 shows the interaction of the different intelligences integrated in the module-based architecture of the goto agent. At the beginning, the goto agent is running its individual fuzzy reasoning to achieve the different target points of the trajectory provided by the path planning. Concurrently, the coordination method with the avoid agent is also

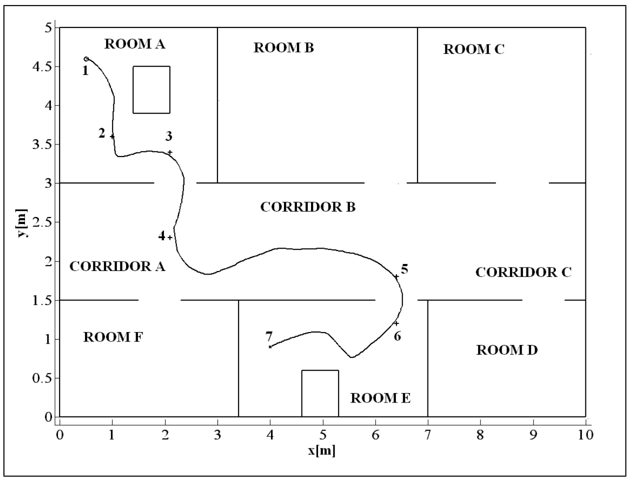

Figure 6. Robot task execution with ARMADiCo.

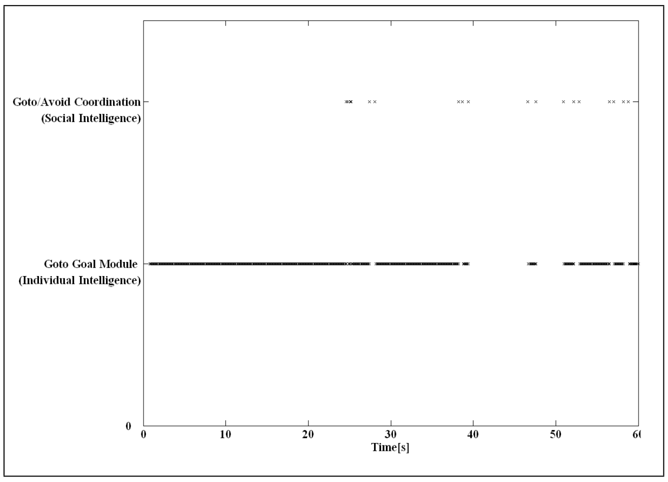

Figure 7. Module activation in the Goto module-based agent.

running: the goto agent is always the winner until that time. In the Goto/Avoid Coordination line, it can be seen the time dedicated to coordination. Then, at the 40 second, the avoid agent gets the resource control; thus, the goto agent dedicates most of its effort for getting back the control again. When it gets it, the goto agent is using its individual reasoning for achieving further points in the trajectory. And so on, until the time when the robot has fulfilled its mission.

4. Related work

The application of multi-agent system to robotics has been mainly concerned to multiple robot system. For example, in [14] several soccer robots coordinate their activities based on case-based retrieval.

Regarding the development of multi-agent architectures for a single robot, there are fewer works. For example, the multi-agent architecture proposed for the reactive level by [12] has two types of agents: elemental, with basic skills, and high-level, responsible for integrating and coordinating various elemental agents. In [3], a multi-agent system is proposed for the navigation system, in which five agents (map manager, target tracker, risk manager, rescuer, and communicator) are coordinated by means of a bidding mechanism to determine the action to be carried out. In [13], a multi-agent architecture is also proposed to deploy an intelligent wheelchair. The agents considered in this architecture are the sensor handler, the collision detector, the corridor recognizer and the drive controller. The behaviors implemented in the system are obstacle avoidance, door passage and wall following. Specifically, the collision detector, responsible for the safety of the robot, is fuzzy-based. The input of the agent is the linear distance, and the velocity and turn-angle are the output.

Regarding coordination, other approaches, as the organization and strategic alliances architectures proposed by [9], follow a central arbiter; while our approach follows a distributed mechanism. This has triggered our research on the integration of different kinds of intelligence at the agent level, since most of the previous approaches, that rely on central arbiters, leave few room to local reasoning methods for coordination.

5. Conclusions

The complexity involved in the design of an autonomous robot with integrated intelligence implies the use of a flexible model that allows the integration of basic control functions, as avoiding obstacles, with higher cognitive capabilities, as performing a task. When tackling such a work, multi-agent systems offer the appropriate abstraction level.

As in any multi-agent approach, conflicts due to shared resource usage can be solved by means of either a central or a decentralized way. By following a decentralized way, social intelligence issues should be considered at the agent level (local) in addition to the individual intelligence ones (that allow agents achieve their goals). In this paper, we have proposed a module-based agent as a way of integrating different artificial intelligence methods to achieve both kinds of intelligences locally, at the agent level.

Regarding individual intelligence, some of the agents follow a search-based method, others a fuzzy logic reasoning, while others a probabilistic approach. Regarding social intelligence, a two steps method has been proposed. In the first step, each agent uses its own private utility method to determine who is the winner of the conflicting resource. In the second step, a Sugeno fuzzy system is used for resource exchange that takes into account the physically grounded features of some of the resources.

All the agents in the architecture follow the same agent-pattern that determines its modular architecture and that makes explicit all the intelligence methods required for a given agent. All agents, at the same time, are integrated in the ARMADiCo multi-agent architecture with no central arbiter. Thus, when dealing with distributed coordination, a module-based agent approach affords us the integration of individual and social intelligence for achieving a single robot feature (goto a point, plan a path) that can be flexible and appropriately used inside the global robot architecture (thanks to the social intelligence). At the multi-agent level, from the interaction of all the module-based agents, the robot achieves its mission based on the combination of several features.

Our architecture has been implemented in a Pioneer 2DX robot of ActivMedia Robotics, and the experimental results show the viability of our approach. As a future work, we need to extend the experiments to much more scenarios, with more agents in the architecture that use other AI techniques, and to compare our approach to other architectures. Although this latter issue is a difficult challenge, due to the fact that architecture replication depends on too many parameters, we should, at least, test our architecture in the same scenarios published in the literature, and extract solid conclusions on our work.

Acknowledgements

This work was partially supported by the Spanish MEC Project DPI2006-09370 and by the DURSI Automation Engineering and Distributed Systems Group, 00296.

References

[1] R. C. Arkin, Behavior-Based Robotics, The MIT Press, 1998.

[2] R. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2** (1),14–23,1986.

[3] D. Busquets; C. Sierra; and R. López de Màntaras, A multiagent approach to qualitative landmark-based navigation, *Autonomous Robots* **15**, 129–154, 2003.

[4] T. De Wolf, Panel discussion on engineering self-organising emergence, *http://www.cs.kuleuven.be/ tomdw/presentations/presentationSASOpanel2007.ppt*, SASO 2007 10-07-2007, MIT, Boston/Cambridge, MA, USA, 2007.

[5] M. Georgeff and A. Lansky, Procedural knowledge, *Proceedings of the IEEE* **74**(10), 1383–1398, 1986.

[6] B. Innocenti; B. López and J. Salvi, Resource Coordination Deployment for Physical Agents, *From Agent Theory to Agent Implementation, 6th Int. Workshop, May 13, AAMAS 2008*, 2008.

[7] G. A. Kaminka, Robots are agents, too!, in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, Invited talk, 2007.

[8] G. A. Kaminka, Robots are agents, too!, *AgentLink News*, 16–17, December 2004.

[9] M. Kolp; P. Giorgini and J. Mylopoulos, Multi-agent architectures as organizational structures, *Autonomous Agents and Multi-Agent Systems* **13**, 1–2, 2006.

[10] R. Murphy, *Introduction to AI Robotics*,The MIT Press, 2000.

[11] R. Murray; K. Åström; S. Boyd; R. Brockett and G. Stein, Future directions in control in an information-rich world, *IEEE Control Systems Magazine* **23**(2), 20–33,2003.

[12] M. C. Neves and E. Oliveira, A multi-agent approach for a mobile robot control system, *Proceedings of Workshop on "Multi-Agent Systems: Theory and Applications" (MASTA'97 - EPPIA'97) - Coimbra -Portugal*, 1–14, 1997.

[13] Y. Ono; H. Uchiyama and W. Potter, A mobile robot for corridor navigation: A multi-agent approach, *ACMSE'04: ACM Southeast Regional Conference. ACM Press*, 379–384, 2004.

[14] R. Ros Espinosa and M. Veloso, Executing multi-robot cases through a single coordinator, *Proceedings of AAMAS'07, the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems, Honolulu, Hawaii, May 2007*, 2007.

[15] J. K. Rosenblatt, *DAMN: A Distributed Architecture for Mobile Navigation*, Ph.D. Dissertation, Robotics Institute at Carnegie Mellon University, 1997.

[16] O. Sauer and G. Sutschet, Agent-based control, *IET Computing & Control Engineering*, 32–37, 2006.

[17] Y. Tahara; A. Ohsuga and S. Honiden, Agent system development method based on agent patterns, *ICSE '99: Proceedings of the 21st international conference on Software engineering*, 356–367, 1999.