Resource Coordination Deployment for Physical Agents

Bianca Innocenti*
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
bianca.innocenti@udg.edu

Beatriz López
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
beatriz.lopez@udg.edu

Joaquim Salvi
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
joaquim.salvi@udg.edu

ABSTRACT

When developing a multi-agent architecture for controlling a single robot, as the one described here, agents share resources and a coordination mechanism is required to solve possible resource usage conflicts. Moreover, some the resources involved in a robot act in the physical world, and the agent exchange on the resource usage can cause some disruptions on the physical robot behavior. So, in addition to coordination method, a mechanism to effectively implement the resource exchange from one agent to another one is needed. In this paper we present a methodology to achieve a resource exchange among different agents that mitigates any possible disruption in the physical robot behavior. The methodology comprises a Sugeno fuzzy system to determine the time window interval required to perform an smoothing change. Our methodology has been tested in a Pioneer 2DX of ActivMedia Robotics.

Keywords

Distributed coordination, physical resource exchange, autonomous robot architecture

1. INTRODUCTION

There is an increasing interest on the use of Agent Technology for developing robots as physical agents [14, 15, 19, 23]. In the majority of the approaches a multi-agent system model a collection of robots. There are some examples in building multi-agent systems for controlling a single robot, too. For example, in [20] a multi-agent architecture is proposed to control a single robot in which two types of agents are distinguished: elemental agents, with basic skills, and high-level agents, responsible for integrating and coordinating various elemental agents. All the agents in the Neves approach deal with reactive robot global capacities. In [1], 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.

Multi-agent approaches facilitate the robot development with higher abstract level modelling capabilities than traditional single-agent, module-based approaches [23]. Today's robots still exhibit simple behaviors compared with humans, and their ability to perform high level reasoning and cooperation is limited [19]. AI has matured enough to offering planning techniques, adaptation and learning methods. Modelling all such cognitive capabilities in a single agent-based robot architecture seems an unfeasible task, and the multi-agent approach offers a new way of combining such AI techniques. A multi-agent architecture provides modularity, distributedness, flexibility and robustness: agents can be added, changed or modified without caring about the other agents [16].

As any robot architecture, resource usage is a key point. While most of the traditional single-agent approaches centralize their use, multi-agent technology offers both centralize and decentralize methods for resource sharing. For example, Neves [20] follows a centralized approach while we follow a decentralized one. In any case, a coordination mechanisms is required so that agents use the shared resources without conflict.

The particularity of the robot case, however, remains on the fact that we are dealing with physical resources, and special attention should be paid when the resource changes from one agent to another one. That is, let us suppose that an agent A is using the robot motor resource, maintaining the robot at a fast speed to the North coordinate. Then, a second agent B, after coordination, gets the robot motor resource. Agent's B goal is to move the robot towards the West and at a low speed. Thus, changing the motor resource from A to B could cause a disruption on the robot action. For example in the experiment shown in Figure 1 there are two agents governing the robot's movement, the avoid obstacle agent (avoid) and the go to a point agent (goto). The robot's initial position is $(x_i, y_i) = (0, 0)m$ and the destination point is $(x_g, y_g) = (5, 0)m$. The trajectory described by the robot is shown, as well as the agent that is controlling the robot movement during the trajectory execution. We can see an abrupt robot movement as a consequence of the resource exchange between the two agents of the architecture at $x = 2.4m$ approximately (see circle in the figure). Of course this could be necessary in dangerous situations, but probably this is not a desired behavior in most of the cases. More often, the agent resource exchange should not affect the robot performance.

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

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

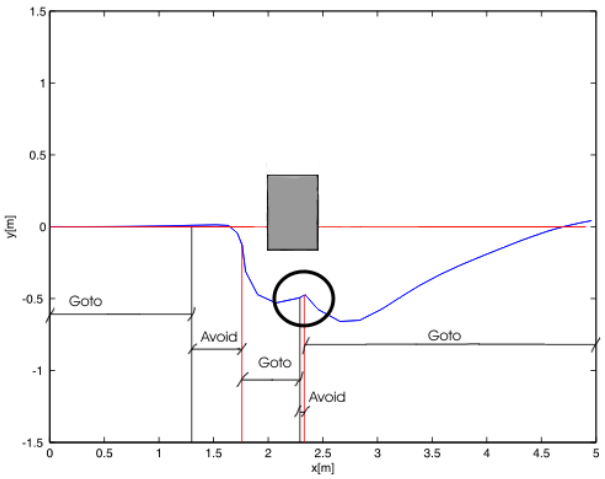

Figure 1: Example of abrupt resource control exchange.

From our understanding, there is a current gap between agent theory and agent implementation regarding this issue of deploying coordination agreements in the physical world. Particularly in [12] we present the multi-agent robot architecture that provides a coordination mechanism to deal with shared resources, but suffers from such a problem. The contribution of the current paper is to introduce a resource exchange methodology in order to cover this gap and that complements and improves our previous work. Regarding theory, we have followed the MaSE methodology to describe our architecture.

This paper is organized as follows. First, we describe in section 2 the multi-agent architecture we have used for physical agents as mobile robots. Then, in section 3 we explain the coordination deploy methodology to avoid abrupt robot behavior changes. In section 4 we show the results we have obtained when implementing the method on a multi-agent architecture for controlling on a Pioneer 2DX of ActivMedia Robotics. Finally, we end with some related work and conclusions.

2. AUTONOMOUS ROBOT MULTI-AGENT ARCHITECTURE WITH DISTRIBUTED COORDINATION

Our architecture, ARMADiCo, (Autonomous Robot Multi-agent Architecture with Distributed Coordination) has been designed to be a general purpose mobile robot architecture such that it can be applied to a wide range of mobile robots. In this paper we illustrate our architecture with the Pioneer 2DX robot but in a near future we will test it with other robots that are in our laboratory. The multi-agent framework allows an easy customization of the architecture to a given robot, by adding the appropriate agent that represents the real robot.

Agent coordination in the architecture is distributed, thus from the micro level (individual agents) emerge a macro level behavior (global system behavior) [3]. Engineering of emergence systems is still an open issue. Recent studies argue in favor of the use of a scientific methodology for their design [6]. This methodology distinguish two phases: theory and practice. Theory follows a top-down design: goals to roles and then to local behaviors. In this phase, current agent methodologies can help [3]. On the other hand, practice is a top-down process: from microscopic behavior, to phenomena and then to macroscopic behavior. Even though experimentation is the only known tool for this phase, several alternatives as information flows, design patterns and others have been proposed to understand the global system behavior [4].

From our experience, we have used the MaSE [5, 25] methodology in the first phase. MaSE is a UML based methodology that has some coincidences with the information flows approaches [4] in order to help in the second experimentation phase. MaSE proposes several steps in the definition of a multi-agent system. In Figure 2 the resulting agent class diagram is shown, as well as the messages sent from and received by each agent. This diagram does not include, for the sake of the figure complexity, the basic FIPA agents, as the Directory Facilitator (DF) agent, but they are considered in the architecture. Finally, when running our agents, we can check how, from the individual agent implementation arises the adequate robot behavior.

For the sake of length we focus on the remaining of this section on the description of two possible agent interactions with conflicting resources, and then on the explanation of the distributed coordination mechanism defined in the architecture.

2.1 Use cases

In order to illustrate the interactions of the agents in order to achieve a global robot behavior, we present two possible use case of the whole architecture, identifying some possible conflicting situations related to the utilization of shared resources.

2.1.1 Case 1. Planning a robot movement.

The task planning agent receives from the interface agent missions to achieve. In order to conduct them, it has a set of procedures similar to PRS [7], in which missions are decomposed into a set of tasks. Eventually, one of this tasks could involve the re-positioning of the robot. So, the task planning agent request to the path planning agent the optimal (in some aspect) trajectory from the current location to the desired one (see *RequestPath* arrow at the top of Figure 2). The path planning agent responds with the trajectory and the energy needed to reach the destination point (see *InformPathEnergy* arrow at the top of Figure 2).

Moreover, the task planning agent sends this information to the battery charger agent (see *RequestViability* arrow). In turn, this latter agent asks the path planning agent for a trajectory from the desired location to closest battery charger point (*RequestPath* arrow from battery charger agent to path planning agent). When the battery charger agent receives the answer from the path planning agent (*InformPathEnergy* arrow), it knows if there is enough energy for achieving the target position and returning back to the charger point (see *InformViability* arrow). In case there is not enough energy, the battery charger agent informs about the corresponding risk to the task planning agent. Thus, the task planning agent could change the task or not; however, in the latter case, it is assuming the possibility that the battery charger interrupts the task execution, as shown in the next scenario. Other alternatives, as merging trajectories

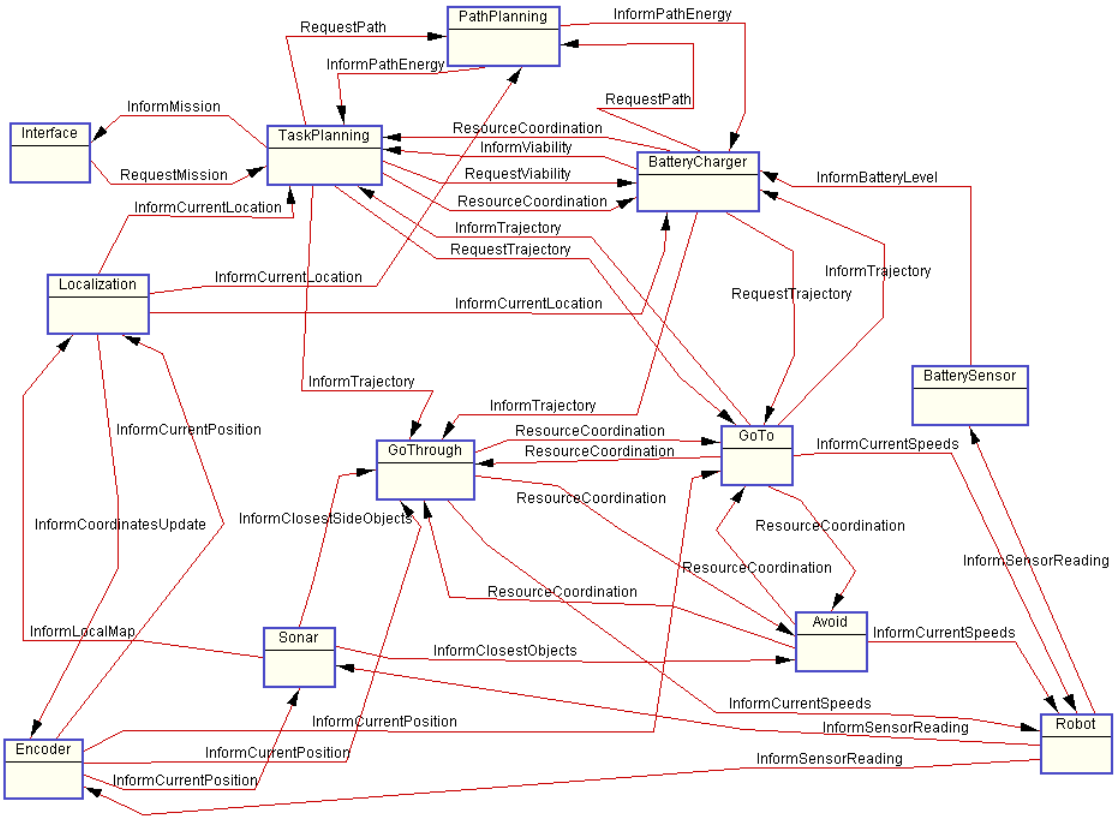

Figure 2: Agent Class Diagram. Boxes are agents and arrows indicates message flow (from an agent and to an agent).

that search a charger point while going to the destination point, are also considered.

As a summary, the task planning, battery charger and path planning are the agents involved in this scenario. Particularly, the path planning agent can be considered as a shared resource between the task and battery charge agents, and it provides the trajectory and energy required to move the robot to a destination point.

2.1.2 *Case 2. Moving the robot.*

Once the task planning agent knows the required trajectory T, to reach a destination position, it sends this trajectory to the goto and gothrough agents (see *InformTrajectory* and *RequestTrajectory* outgoing arrows of the task planning agent in Figure 2). This trajectory T is a sequence of points $p_1, \ldots, p_n$ to reach.

The goto agent, who knows the current position and heading (*InformCurrentPosition* arrow outgoing from the encoder agent), calculates the linear and angular velocities necessary to reach the first point of the trajectory, p_1. The gothrough agent does the same calculations but considering obstacles at the side of the robot (in the case that the robot is in a hallway or narrow place). On the other hand, the avoid agent is constantly looking for obstacles in front of the robot in order to dodge them. Then, after coordination (see *ResourceCoordination* arrow among these three agents), one of them, the winner, sends the computed commands to the robot agent (*InformCurrentSpeeds* arrow).

Once the robot has reached the first position of the trajectory p_1, the goto and the gothrough agents continue with the next one, p_2, and so on until the last point of the trajectory p_n is reached.

It could be the case, however, that when the goto agent is sending a request to the robot agent to move towards the p_i point, the battery charge agent realizes about the battery level is under the minimum security level required. This could happen, for example, when the robot has performed a larger trajectory than the planned one due to the presence of obstacles. Then, the battery charger requests a new trajectory T' to the goto agent in order to reach a charger position as soon as possible.

In these scenario, then, three agents (goto, avoid, gothrough) are sharing the same resource: the robot agent to which they send conflicting commands. In turn, the goto agent is a resource of the task planning and battery charger agents, since it can receive conflicting trajectory requests from them.

2.2 Distributed coordination

Coordination among agents is necessary when there are several agents trying to use the same robot resource at a given time. For example, a conflict can arise between the task planning and charge battery agents when trying to request different destination positions to the path planning agent (case 1) or when requesting different positions to the goto agent (case 2). Also, among the avoid, the goto and the gothrough agents when trying to request conflicting actions to the robot agent (case 2). Note then, that the path planning agent, the goto agent and the robot agent can be

considered shared resources, depending on the situation.

One solution to the problem is that the conflicting resources, being agents, solve by themselves the conflicting situation in a centralized way (with an auction, for example). However, we have adopted a decentralized approach in which the agents involved in the conflict decide which of them take the resource control. This decision avoids having a single agent arbitrating the overall architecture that could become a bottleneck when the number of agents is high.

So, in ARMADiCo, when an agent enters in the system, it provides to the DF agent information about the resources it uses. Then, the DF sends back to the agent the identification of the other agents using the same resources. Thus, any new agent in the system knows which are the set of agents to which it needs to coordinate in case of conflict.

To decide what agent wins the shared resource in case of conflict, each agent computes a normalized utility value (between [0,1]) regarding its actuation. The procedure to compute the utility is only known by the agent itself. The outcome of the computation, that is, the utility value, is the one used for coordination. The utility computation has been described in [12], and other approaches as for example [18] can also be followed.

In a given moment, there is only one agent that uses the resource in conflict. This agent sends its utility to the other agents who share the resource. When another agent in the architecture has a higher utility, it takes the control of the resource, and starts sending to the remaining agents the value of its utility. Thus, the agent who has a higher value of utility wins the resource.

For example, suppose the use case 2, in which the robot agent is the shared resource, that is currently controlled by the goto agent. This control agent is informing to the gothrough and the avoid agents about its utility. In a given moment, the goto agent has a utility value of 0.5, and the avoid agent of 0.7; being 0.7 the higher value. Then the avoid agent takes the control of the situation, and it is the only one that request some actuation to the robot agent.

After a coordination agreement is achieved (a new winner is decided), additional computation is also required to adequate the robot physical actuation that encompass the resource exchange between the two agents.

3. COORDINATION DEPLOYMENT METHODOLOGY FOR PHYSICAL AGENTS

When an agent, after coordination, obtains the opportunity to use a robot shared resource, the agent should proceed on the resource usage taking into account its impact on the physical world in a similar manner than control fusion [9, 12, 22]. For doing so, we propose a method based on the information used in the coordination process.

Let a_w be the agent that wins the resource, and let a_l be the agent that loses the resource (it has been using the resource until now). For example, if the shared resource is the robot agent, a_w could be the goto agent, and a_l could be the avoid agent. Let u_w and u_l be the utilities of a_w and a_l correspondingly (so $u_w > u_l$). In [12] details of the utility computation are given. Also in this previous work arises the problem of smoothing the resource exchange, which we are tackling in this paper.

Let a_r be the shared resource (agent that owns a shared robot resource). The resource is characterized by n parameters that configure the possible actions requested to the resource (for example, robot commands). Then let $a_{w_1}, \ldots, a_{w_n}$ be the actions requested by the winner agent and $a_{l_1}, \ldots, a_{l_n}$ the ones of the loser agent.

First of all, we need to determine the time window frame t_f available in order to perform the resource action exchange, that is, how much time we have to change from the current action of the loser agent to the required action of the winner agent. This time depends on the criticality of the robot state:

- The robot is changing from a critical to a non critical situation
- The robot is changing from a non critical situation to a critical one

How t_f is computed in both cases is shown below.

Then, a progressively change from each loser action a_{l_i} to each winner action a_{w_i} is performed according to the following expression (that extends the work presented by [9]):

$$\frac{\delta_w(t_i, u_w) * a_{w_i} + \delta_l(t_i) * a_{l_i}}{\delta_w(t_i, u_w) + \delta_l(t_i)} \quad (1)$$

where $\delta_w(t_i, u_w)$ is the contribution of a_{w_i} in time t_i, and $\delta_l(t_i)$ is the contribution of a_{l_i}. Time t_i is measured in robot cycles. So after the first cycle since the winner agent obtains the resource, t_1, the contribution of δ_l should be higher than the one of the δ_w. As cycles go on, the value of δ_w wins importance; in a given moment, t_f, the action value should be the one of the winner agent, so δ_w should have the highest value (1) and δ_l the lowest one, (0).

According to this desired behavior, δ_w and δ_l have been defined in [0, 1] as follows:

$$\delta_w(t_i, u_w) = \frac{u_w}{t_f} * t_i \quad (2)$$

$$\delta_l(t_i) = \frac{u_l}{t_f} * (t_f - t_i) \quad (3)$$

Note that the resource exchange is performed by the winner agent that knows all the information regarding the utilities of the loser agent, as well as the actions. However, the information about the loser agent is a "snapshot" of the loser agent values in the moment that the resource exchange is performed, while the values of the winner agent can be updated in each robot cycle. This is the reason that $\delta_w(t_i, u_w)$ changes also according to u_w, while in Equation 3, u_l is static.

The summary of the coordination deployment algorithm is shown in Figure 3. The 2.2 step is briefly described, but involves the agent utility computation, coordination information exchange and others.

In the remaining of this section, we give details of how t_f is computed, and we illustrate the overall methodology with an example.

3.1 Time window frame from critical to non-critical situations

The coordination exchange from critical to non-critical situations can happens, for example, when the loser agent is the avoid agent, the winner the goto agent, and the robot agent is the resource agent. In this situation, the avoid agent

1. Find the time window frame (t_f)
2. While u_w is the resource winner and $t_i < t_f$ do
   {
2.1 Compute current action parameters $a_c = \frac{\delta_w(t_i,u_w)*a_{w_i}+\delta_l(t_i)*a_{l_i}}{\delta_w(t_i,u_w)+\delta_l(t_i)}$
2.2 Execute a_c
2.3 Next i
   }

Figure 3: Coordination deployment algorithm.

has dodged an obstacle, and now, the goto agent wins the resource in order to continue to the next robot goal position.

Then, the time window frame t_f depends on how much different the actions are between the loser and the winner ones. For measuring this difference, we use the Tchebychev distance [24] applied to the most critical action parameters, as follows:

$$max_{criticalParameters(1,\ldots,n)}(|a_{w_i} - a_{l_i}|) \quad (4)$$

The $criticalParameters(1,\ldots,n)$ function is defined for each shared resource, and returns the set of critical parameters of the resource. For example, in the case of the robot agent, the critical parameter is the linear velocity.

So, if this distance is large, the exchange period should be long, while if the distance is short, the period of change should be close to 0. The concepts of large and short can be easily modelled following fuzzy variables, and then computing the time window frame t_f according to a fuzzy system.

Particularly, we have followed a Sugeno fuzzy system. In such a system, we distinguish the input variables, the output variables and the rules. There is a single input variable "diff" that represents the action parameters difference according to the distance function defined in Equation 4.

The $diff$ variable is defined in [0,100], since the maximum velocity of the robot is 100cm/s. Four different fuzzy values are defined for the fuzzy variable: *equal*, *small*, *medium*, and *big*. A gaussian membership function is associated to each value. Figure 4 shows the functions used for each fuzzy value.

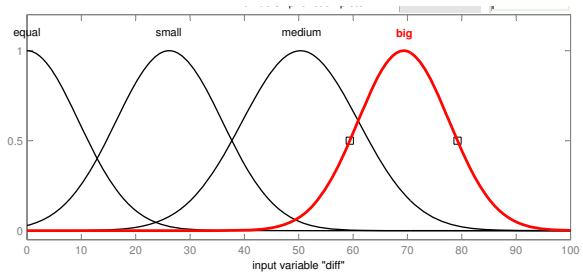

Figure 4: Input fuzzy values of the diff variable.

There is a single output variable too, that represents the length of the time window frame, t_f. Each value of t_f in [0,10] represents the number of cycles required to change from the current action parameters to the new ones. Four different values have been defined for t_f: *null*, *short*, *medium*, *long*. In a Sugeno system, each value of a output variable is a linear combination of the input variables. That is:

$$t_{fx} = c_1 * diff + c_2 \quad (5)$$

In our case, we have taken $c_2 = 0$ (otherwise there will be always a delay in sharing the resource even though when the required actions are the same), and the following c_1 coefficients for each t_f values have been defined:

$$t_{fnul} = 0 * diff$$
$$t_{fshort} = 3 * diff$$
$$t_{fmedium} = 5 * diff$$
$$t_{flong} = 10 * diff$$

Regarding the rules, four possible rules have been considered, according to the input and output possible values. These are the following ones:

R1. If $diff$ is equal then t_f is null
R2. If $diff$ is small then t_f is short
R3. If $diff$ is medium then t_f is medium
R4. If $diff$ is big then t_f is long

Finally, the output value of t_f is defuzzified by using the weighted mean. As the cycle time of the robot is $100ms$, it is possible to neglect the required time to calculate t_f, since all the calculations can be done before a cycle elapses.

3.2 Time window frame from non-critical to critical situations

The coordination exchange from a non critical situation to a critical one happens, for example, when the loser agent is the goto agent, the winner the avoid agent, and the robot agent is the shared resource. In this case, the avoid agent has detected a dangerous situation and the time required to react to it, t_c, is critical. Then, $t_f = t_c$. Note that t_c is known by the winner agent who applies the coordination deployment method. In the next section we show how to calculate t_c for a particular situation.

3.3 Example

The robot agent is shared by the avoid and the goto agent. The conflicting actions requested to the robot agent are the velocities (linear and angular) to which the robot should move. In order to illustrate the coordination deployment mechanism, let us suppose two different scenarios: when the avoid agent obtains the resource and when the goto agent obtains the resource.

In the first scenario, we have the following configuration:

- a_r is the robot agent, with 2 parameters: the linear and the angular velocities.
- a_w is the avoid agent, with $a_{w_1} = 20cm/s$ (linear velocity), and $a_{w_2} = -11degrees/s$ (angular velocity). The avoid agent has a utility value of $u_w = 0.7$.
- a_l is the goto agent, with $a_{l_1} = 76cm/s$, $a_{l_2} = -14degrees/s$, and $u_l = 0.6$.

Since the situation is characterized as a critical one, t_f is set to the critical time computed by the avoid agent. The critical time is computed as the time to collide with the obstacle, that is, the distance to the object divided by the current linear speed. Finally, the second step of the algorithm

of Figure 3 is applied, obtained an smoothing behavior in the global robot control.

In the second scenario, we have the following configuration:

- a_r is the robot agent, as in the previous scenario.
- a_w is the goto agent, with $a_{w_1} = 46cm/s$, $a_{w_2} = 19.4degrees/s$, and $u_w = 0.77$.
- a_l is the avoid agent, with $a_{l_1} = 20cm/s$, $a_{l_2} = 0degrees/s$ and $u_l = 0.28$.

This scenario corresponds to a transition from a critical to non critical situation. The $criticalParameters(1, 2)$ of the robot agent is the first one, that is, the linear velocity. Then, the time window frame t_f is set according to the Sugeno fuzzy system. The difference between the two linear velocities is the following:

$$diff = |a_{w_1} - a_{l_1}| = |46cm/s - 20cm/s| = 26cm/s$$

Such value partially fulfills the small and equal fuzzy values of the $diff$ input variable of our Sugeno fuzzy system. As a consequence, the resulting output variable t_f is set to 3 cycles.

4. EXPERIMENTAL RESULTS

In order to test ARMADiCo we have implemented an ad hoc multiagent platform, programmed in C++ on Linux. This implementation decision is based on the fact that the majority of the commercial platforms have an agent that centralizes the functioning of the entire platform and 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. This robot has a minimal sensor configuration: two encoders and eight ultrasound sensors. A complete description of the robot, as well as its dynamics can be found in [13].

In order to test the coordination deployment methodology proposed in this paper, we have implemented the following agents: interface agent, (very simple) task planning, path planning, goto, avoid, sonar, encoder, and robot agents. In this configuration, the robot agent is the shared resource between the goto and avoid agents.

4.1 Experimental setup

Two different ways of taking the resource control have been tested. The first one occurs when the agents change abruptly the control over the shared resource and the second one, when the proposed method is used. Thus, we have two main configurations:

- Abrupt: changing the control abruptly
- Smoothing: using our proposed method. That is, using Equation 1 to have a smoothing behavior when exchanging the resources after coordination.

In order to test the whole behavior of the architecture, we propose the scenario shown in Figure 5, in which the robot must go from room A to room B. For doing so, the *task planning* agent provides to the *goto* agent the corresponding trajectory necessary to pass through the doors.

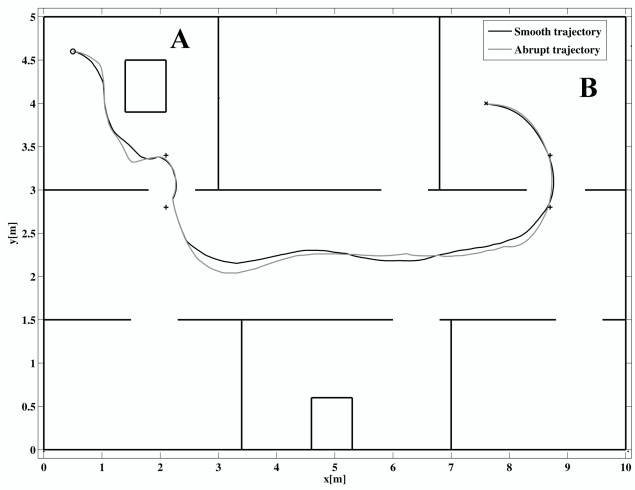

Figure 5: Example of a trajectory with our methodology.

4.2 Results

Figure 5 shows an example of the trajectory described by the robot, when using the abrupt exchange of resources (grey line) and using the proposed methodology (black line). As can be seen in this figure, at the points $(x, y) = (1.5, 3.3)m$ and $(x, y) = (6.3, 2.4)m$ approximately, there are abrupt changes in the trajectory described by the robot using the abrupt resource exchange methodology (grey line), but they are avoid using the smoothing algorithm (black line).

In order to compare the results, the following measures have been considered:

- **Travelled Distance (TD):** the distance travelled by the robot to reach the goal.
- **Final Orientation (FO):** the heading of the robot at the goal position.
- **Total Time (TT):** the total amount of time the robot needs to achieve the goal.
- **Precision (P):** how closed to the goal position is the center of mass of the robot.
- **Time Goto (TG):** the total time the *goto* agent has the robot control.

The experiments consisted of five executions in the scenario for each configuration. Table 1 shows the average and standard deviation of each evaluation measure. Even though mean values are not so different among the two situations, deviations are enhanced.

Table 1: Comparison of the results of the three tested situations.

Parameters	Abrupt	Smoothing
TD	$11.96 \pm 0.103m$	$11.99 \pm 0.17m$
FO	$2.14 \pm 0.47^{\circ}$	$2.16 \pm 0.85^{\circ}$
TT	$69.37 \pm 2.43s$	$64.39 \pm 1.24s$
P	$99.648 \pm 0.08\%$	$99.82 \pm 0.05\%$
TG	$73.86 \pm 3.23\%$	$51.25 \pm 5.41\%$

Comparing the abrupt change of control with the smoothing one, the most significant improvement is the total time (TT) needed to reach the goal. In the *smoothing* configuration, TT has decreased meaning that the cruising speed can be maintained for more time and changes between agents with a short duration are almost eliminated, achieving softer trajectories. At the same time, the time the goto agent has the control (TG) has been decreased, while the avoid agent has the control in more consecutive cycles. So, when an agent takes the resource control, it does it for a larger time using our methodology than without using it, thus, the agent can effectively follow their own goals for a larger period of time; and consistently the robot global behavior is more coherent.

5. RELATED WORK

As stated in the introduction, there are some works related to the use of multi-agent system for a single robot architecture [20, 1, 21]. In [16], several agents are organized in a spreading activation network, so that each one is defined with a set of pre- and post-conditions. When an agent accumulates enough activation (i.e., the pre-conditions are satisfied over a given threshold), it becomes active. Conflicting interactions among agents (for example, when parallel actuation requires a three hands robot) are solved according to different selection parameters. These parameters establish the urgency to fulfil the different goals. This approach proposed by Maes's work is more related to coordination (even though at a lower level than our approach and other recent ones [1], [20]) than to the physical resource exchange issue we are dealing here. From our understanding, none of the previous authors address the problem of filling the gap between coordination agreement and coordination execution in multi-agent systems when dealing with physically-grounded shared resources.

In the literature of control, however, there are several proposals to perform this resource exchange. For example, in [10], several controllers are used to control each wheel of the robot. The controllers are modelled as finite automata whose inputs are the location of the robot and the output, the values -1,1 and 0. After each controller produces its output, a central module performs the average of the total vote of the outputs, determining the desired increase in the movement.

Gerkey in [9] presents a similar idea but instead of having an explicit module that fusions command controls, all the controllers actuate directly on the motors, being these elements the ones that overlap the control signals and achieve the resulting movement. He proved that the presence of "malicious" controllers degrade the overall performance or produce a catastrophic failure when the proportion of them is higher enough.

Another example is presented in [22]. In particular, the authors use fuzzy logic to model the control actions coming from the heterogeneous controllers and to decide, in accordance with the dynamic model of the robot, what combination of control actions should be carried out at any given moment. We are also using a fuzzy system but with a different purpose. Instead of using it to compute the final command value, we use a fuzzy system to compute the time interval required to perform the resource exchange.

The change of commands in a robot has also been studied at the task level in behavioral-based architectures. For example, in [18], an adaptation method to deal with conflicting behavior interactions is proposed. After several trials, the robot learns the appropriate sequence of behaviors. Even though Mataric's proposal is closer to coordination mechanism than our resource exchange problem, some similarities can be found with our approach. That is, we are also proposing a sequence of action combination (transition phase) that facilitates the resource exchange between two agents. Note, however, that in [18] and also in [17], only one of the behaviors is active at a time.

From the agent community, physical resources has been studied by the holonic manufacturing community [8]. However, most of these approaches (see for example [11]) are mapping a robot to an agent (or a machine to an agent), conversely to our approach, in which we are building a multi-agent system for a single robot.

6. CONCLUSIONS AND FUTURE WORK

There is an increasing interest in the use of agent technology to develop robots as physical agents, and there starts to be some examples of the use of multi-agent systems for building a single robot architecture. In this paper we describes a multi-agent architecture with this purpose, ARMADiCo and we focus on the importance of the deployment of the coordination outcomes since they affect the robot global behavior. That is, when defining such architecture, several robot resources are shared by different agents. So, a coordination mechanism is required in order to avoid conflicting resource uses. This is something that most of the theoretical studies considers. However, when trying to deploy coordination mechanism to the physical world, as in robots, there is gap related to how the resource exchange is performed; and that matters. In this paper we describe a coordination deployment methodology to cover such a gap. The methodology comprises a Sugeno fuzzy system to determine the time window interval required to perform an smoothing resource exchange.

The methodology presented has been tested using a Pioneer 2DX of ActivMedia Robotics. The results show that the robot maintains a cruising speed more constant when using our coordination method than without it, as a consequence of reducing the number of short resource exchanges between agents, and achieves in the end a smoothing global behavior when moving through a space with obstacles.

As a future work, we are considering new experimental configurations with the incorporation of additional agents and resources. Consequently we are analyzing other possible resource allocation methods of the multi-agent literature (as for example [2]). We are also studying the definition of more complex scenarios that involve higher cognitive capabilities. Finally, we are also analyzing the use of new and still open paradigms as information flows [4] that helps in the understanding of the global emergence multi-agent system behavior.

7. REFERENCES

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

[2] Y. Chevaleyre, P. Dunne, U. Endris, J. Lang, M. Lemaître, N. Maudet, J. PAdget, S. Phelps, J. Rodríguez-Aguilar, and P. Sousa. Issues in

multiagent resource allocation. *Informatica*, 30:3 – 31, 2006.

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

[4] T. De Wolf and T. Holvoet. Using UML 2 activity diagrams to design information flows and feedback-loops in self-organising emergent systems. *Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems (EEDAS 2007)*, pages 52 – 61, 2007.

[5] S. A. DeLoach. Analysis and design using MaSE and agentTool. *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, 2001.

[6] J. Fromm. On engineering and emergence. *SAKS/06, Workshop on Adaptation and Self-Organizing Systems (nlin.AO)*, arXiv:nlin/0601002v1 [nlin.AO], 2006.

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

[8] C. Gerber, J. Siekmann, and G. Vierke. Holonic multiagent systems. Research RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1999. www: http://www.dfki.de.

[9] B. Gerkey, M. Mataric, and G. Sukhatme. Exploiting physical dynamics for concurrent control of a mobile robot. *Proceedings ICRA '02. IEEE International Conference on Robotics and Automation*, 4:3467 – 3472, 2002.

[10] K. Goldberg and B. Chen. Collaborative control of robot motion: robustness to error. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 655–660, 2001.

[11] H. Hsu and A. Liu. A flexible architecture for navigation control of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Part A: Systems and Humans*, 37(3):310–318, 2007.

[12] B. Innocenti, B. López, and J. Salvi. A multi-agent architecture with cooperative fuzzy control for a mobile robot. *Robotics and Autonomous Systems*, (55):881 – 891, 2007.

[13] B. Innocenti, P. Ridao, N. Gascons, A. El-Fakdi, B. López, and J. Salvi. Dynamical model parameters identification of a wheleed mobile robot. *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles (preprints)*, 2004.

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

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

[16] P. Maes. The dynamics of action selection. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 991 – 997, 1989.

[17] M. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents*, 9 (2-3):323 – 336, 1997.

[18] F. Michaud and M. Matarić. A history-based approach for adaptive robot behavior in dynamic environments. *Proceedings of the second international conference on Autonomous Agents.*, pages 422 – 429, 1998. ISBN:0-89791-983-1.

[19] 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, issue 2:20 – 33, 2003.

[20] 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*, pages 1 – 14, 1997.

[21] 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.*, pages 379 – 384, 2004.

[22] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180 – 197, 1997.

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

[24] R. Wilson and T. R. Martínez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1 – 34, 1997.

[25] M. F. Wood and S. A. DeLoach. An overview of the multiagent systems engineering methodology. *Lecture Notes in Computer Science. Vol. 1957/2001, Springer Verlag.*, pages 207 – 221, 2000.