Distributed multi-agent architecture for humanoid robots

Philippe Lucidarme

University of Angers,LISA
62avenue notre Dame du Lac
49000 Angers

Abstract

In this paper, a distributed architecture for redundant robot is presented. In the proposed approach, each actuator is respectively associated with an agent and each agent acts independently from the other to reach a common goal. The architecture has been simulated on the model of the humanoid robot HRP2. The architecture has to deal with a multi-objective task: reaching a target with the hand without falling (keeping the center of mass in the footprint). Simulation results show the efficiency of the architecture. The architecture is failure-tolerant; the lost of one or several agents may be supported by the system. Computation time is discussed. Extensions are presented in order to control the speed and the trajectory of the end-effector.

Keywords

Multi-agent system, distributed architecture, humanoid, stability.

1 INTRODUCTION

Recently, many studies have focused on the development of humanoid biped robot platforms. To control such robots, the designers have to face several problems. First of all, unlike industrial robots, these robots are not attached to the ground and the stability has to be taken into consideration. Secondly, such machines are generally composed of a large number of actuated joints and belong to the highly redundant robots category. For a given position of the end effector, there is an infinite number of robot's postures. Finally, due to their human-inspired shapes, it appears necessary for the robot to behave like human does, of course this subjective criteria is quite hard to transform into equations.

For these reasons, the classical controllers (originally design for industrial robots) are generally not well suited for humanoid robots. This paper introduces an original distributed architecture inspired from multi-agents systems. In the proposed architecture the controller may be physically distributed into the robot. Each actuator is controlled by an agent and the communications between the agents are local, i.e. each agent communicates only with its direct neighbors.

This article is organized as follows. Section 2 gives an overview of related works. Section 3 presents the proposed architecture. Next section is dedicated to the implementation on a humanoid robot and to the control of the stability. Section 5 discusses about simulation results. A general conclusion ends the paper and presents some perspectives.

2 RELATED WORKS

The first researches on multi-agent system began about 20 years ago in the field of distributed artificial intelligence. The first application to robotics appears in the 90's with the works of R. Brooks inspired from ant's colonies [1] and [2]. He proposed several architectures based on reactive behaviors and machine learning.

More recently, the cooperative control of multiple robots has been studied. R. Arkin worked on the behavior based control of cooperative groups of robots [3] and [4], and more recently on the formation control of multiple mobile robots [4] and [5]. Various architectures inspired from these works have been proposed applied to spacecraft [7], to satellites [8] or to nonholonomic robots [9] …

In the last ten years, self-reconfigurable robotics appeared as a new research topic of prime interest [10]. Such machines are composed of modules that can be reconfigured according to the task. In the existing platforms, each module is fully autonomous; it embeds batteries, computational power … Several distributed architecture has been proposed based on multi-agent approach. A recent work focused on the stability of self-reconfigurable robots. A particularly interesting survey on the distributed control of the centre of mass (CoG) was performed by M. Moll and al. in [11]. In this approach, the authors proposed a distributed architecture allowing the calculation and the control of the CoG based on local communication. However, the task is dedicated to the stability and this work is not focussed to multi-objective. The problem of stability while performing a task (object tracking for example) is not considered. The distributed architecture proposed in the following can deals with several objectives, even conflicting.

3 DESCRIPTION OF THE ARCHITECTURE

In the proposed distributed approach each actuator q_i is respectively associated with the agent A_i. Each joint or agent acts independently from the other in order to minimize the Euclidian distance ε between the end-effector and the target. Each agent is described by the following 3 items : input, output and behavior .

3.1 Input

The input is an information or a data, to which the agent can access. The agent A_i knows the following variables:

- ${}^{0}T_{i-1}$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_{i-1}. This information is communicated by the agent A_{i-1} except for the first agent.
- ${}^{i}T_{n}$: transformation matrix linking the joint q_{i+1} to the end effector. This matrix is communicated by the agent A_{i+1} except for the last agent.
- q_i : current measured position of the joint controlled by the agent A_i. This position if provided by a sensor on the joint .
- P_{Target} : coordinate of the target in the frame $F_{0.}$ We assume that this information is known by each agent. On a real physically distributed system, this information may be computed by a dedicated controller and transmitted to the closest agent. This agent transmits this information to the neighbors and the coordinates are propagated into the rest of the system.

3.2 Output

Each agent A_i must provide two kinds of outputs: command on the actuator and information to the neighbors:

- Δq_i : command applied on the joint (angular speed).
- ${}^{0}T_{i}$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_i. This information is communicated to the agent A_{i+1}.

- ${}^{i-1}T_n$: transformation matrix linking the joint q_i to the end effector. This matrix is communicated to the agent A_{i-1}.
- P_{Target} : coordinate of the target transmitted to the agents A_{i-1} , A_{i+1} or both.

A general view of the architecture, showing the information exchanged between the agents is shown on Figure 1.

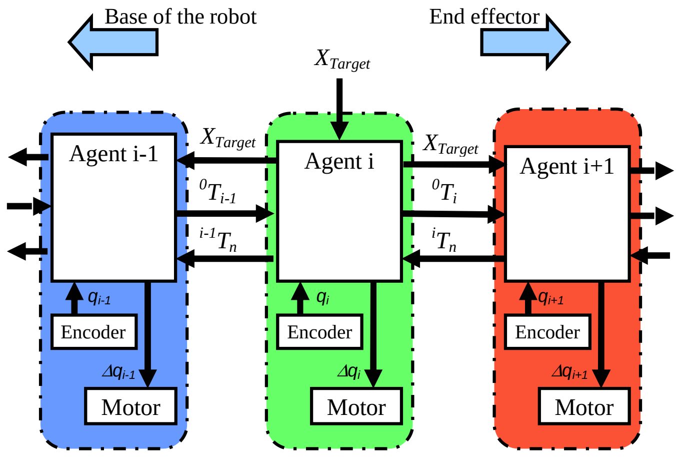

Figure 1 : Overview of the multi-agent architecture

3.3 Behavior

This is the way the agents link the input to the outputs. Note that, as the system is distributed, the behaviors are local, i.e. the agent *Ai* does not know the way the other agents will act. The global goal is to reach the target with the end-effector of the robot. Internal collisions are not considered here. The behavior of the agent *Ai* consists of computing at each time step the value Δq_i that minimizes the Euclidian distance between the position of the end effector *Pn* and the position of the target P_{target} in regard to the information known by the agent. Due to its efficiency and its simplicity, the gradient descent technique has been chosen. This choice was motivated by the following reasons:

- The mathematical relationship between the actuator's position and the distance to minimize is a known function. The derivative of this function may be easily computed.
- There are few parameters to set, just one for each agent and evolutionary computation is well suited for tuning these parameters.
- The main advantage is probably that the algorithm slides into the minima. It does not provide just the final solution, but provides a trajectory that slide from initial to final

Each agent updates its output according to equations 1

$$\Delta q_i(t) = -\alpha_i . \frac{d\varepsilon}{dq_i} \qquad (1)$$

ε is the Euclidian distance between the hand of the robot and the target

q_i is the current position of the joint *i*

α_i is a parameter that determine the behavior of the agent A_i . The influence of this parameter is described in the next section.

The derivative of the Euclidian distance is computed using the equation 2.

$$\frac{d\varepsilon}{dq_i}=\frac{(x_{T\arg et}-x_n).\frac{dx_n}{dq_i}+(y_{T\arg et}-y_n).\frac{dy_n}{dq_i}+(z_{T\arg et}-z_n).\frac{dz_n}{dq_i}}{\sqrt{(x_n-x_{T\arg et})^2+(y_n-y_{T\arg et})^2+(z_n-z_{T\arg et})^2}} \qquad (2)$$

Note that, as the system is fully distributed, the agent A_i cannot compute $\frac{dx_n}{dq_i}$, $\frac{dy_n}{dq_i}$ and $\frac{dz_n}{dq_i}$ by using the jacobian because according to our hypothesis, this agent A_i cannot access to the other agent's *dq*. In spite of this, the matrix product described on equation 3 can compute the derivative of the end-effector position according to the joint *i*.

$$\frac{dP_n}{dq_i}={}^0T_{i-1}.\frac{d\,{}^{i-1}T_i}{dq_i}.{}^iT_n.X_0 \qquad (3)$$

The relationship described on equation 4 is only true in the case of serial robots. Only one transformation matrix (${}^{i-1}T_i$) is expressed in term of q_i, t. The following terms can be considered as scalar and don't need to be derivate: $dX_n\ dq_i$, ${}^0T_{i-1}$ and ${}^iT_n$. The behavior of the agent A_i is described in the algorithm presented on the Figure 2.

```
Algorithm AgentBehavior

input : $^OT_{i-1}$, $^iT_n$, $P_{T\arg et}$, $q_i$
output : $\Delta q_i$, $^{i-1}T_n$, $^OT_i$

At each time step :
    Compute : $^{i-1}T_i$ and $\frac{d\,^{i-1}T_i}{dq_i}$
    Compute : $\frac{dP_n}{dq_i}$ and $\frac{d\varepsilon}{dq_i}$

    Update : $^OT_i={}^0T_{i-1}.{}^{i-1}T_i$
    Update : $^{i-1}T_n={}^{i-1}T_i.{}^iT_n$
    Update : $\Delta qi=-\alpha_i.\frac{d\alpha\varepsilon}{dq_i}$
```

Figure 2 : Algorithm of the agent A_i

Note that for each agent a parameter α_i have to be tune. Experiments have shown that these parameters influence the behavior of the robot. For example, the agents with the highest coefficients will be more frequently stimulated.

4 APPLICATION TO THE HUMANOID ROBOT HRP2

4.1 Hypothesis

The proposed architecture has been simulated with the model of the robot HRP2 (shown on Figure 3).

Figure 3 : The humanoid robot HRP2

This humanoid robot has 30 degrees of freedom (6 by leg, 6 by arm, 1 by hand, 2 for the hip and 2 for the neck). This robot has been chosen for its high redundancy, even if many other robots may have been used. In the present application, the robot has to reach a target with its right hand without falling or being unbalanced. We assume that the target is reachable without walking, i.e. feet are considered as linked on the floor. The robot may have to bend down according to the position of the target. The position of the target is assumed to be known by the robot; it may be done by using the vision system of the robot for example. The body of the robot is decomposed in agents each controlling a joint. The system is fully distributed, i.e. information about the agents is not centralized. Of course existing robots use a central processor and have not been designed to support such architecture. Currently, the distribution is simulated by sharing the central processor, but it may be possible to embed a controller in each joint to perform the agent behavior. A homogeneous multi-agent system is considered; however, as each agent controls a joint of the humanoid robot, the computed model and parameters vary.

4.2 Stability

In order to apply the algorithm to a humanoid robot, the control of the static stability has been added. This is based on the same principle that for target tracking; rather than minimizing the distance between the hand and the target, the goal is to minimize the distance between the projection on the floor of the center of mass (COM) and the center of the footprint. To deal with the two objectives at the same time, the formula presented on the equation 4 is used. This equation warrant that when the robot is perfectly stable ($\gamma=1$) the algorithm will control only target tracking. On the other hand, when the projection of the center of mass is on the edge of the footprint (ie. the robot is at the limit of stability) hundred percent of the command will be dedicated to keeping the stability ($\gamma=0$). Between these two extremes, the ratio is linear. The equation been implemented on the model of HRP2. The first simulations have immediately shown the efficiency of the method and the emergence of behavior. For example, the robot dedicates its right arm only for stability because the right arm is not useful for target tracking. One actual drawback of this technique is that the designer has twice as many parameters to set. One set of parameters for target tracking, and one set of parameters for stability. Experiments have shown that the parameters can be set arbitrary, likely due to redundancy. But finding a set of parameters that provide an optimal and/or human-like motion is not trivial. This is the reason why evolutionary computation has been used to evolve such parameters. For more information about the evolutionary computation of the parameters, the reader can consult [12].

$$\Delta q_i = -\gamma . \alpha_i^{Target} . \frac{d\varepsilon_{Target}}{dq_i} - (1-\gamma) . \alpha_i^{COM} . \frac{d\varepsilon_{COM}}{dq_i} \tag{4}$$

Where :

$$\gamma = \frac{D_2}{D_1 + D_2}$$

ε_{COM} is the distance between the projection of the center of mass and the center of the footprint

D_2 is the distance between the projection of the center of mass and the center of the footprint

D_1 is the distance between the projection of the center of mass and the closest edge of the footprint

5 RESULTS

Experiments have been conducted by simulation using the geometric model of the robot HRP2, including the center of mass of each body allowing the computation of the global center of gravity of the whole robot. Each joint is bounded in the same interval as on the real robot. The simulator is of course based on a centralized processor that simulates at each time step the behavior of each agent. This allows the simulation of the distributed system. Three fitness functions has been experimented with the simulator, minimizing the time of the motion, minimizing the traveled distance of the hand, and minimizing the energy. Even if the two first functions provide interesting results, the simulations clearly show that minimizing the energy provide nice and smooth motion. The robot is able to reach the target whatever its position in the workspace of the robot.

Figure 4 shows the motion of the robot. The target is located on the floor and the robot has to bend down to reach this point. Note that when the robot leans forward it puts back its right arm to compensate the move of the center of gravity.

5.1 Failures

So as to test the robustness of the system failures have been simulated. The first test was a failure on an actuator. As the agents are independent, the system still works; the other agents naturally compensate the inactivity on a joint. The second test simulated failures on several joints. When too many actuators are inactive, the system is not always able to reach the target, sometimes even if it's always theoretically possible. These experiments have shown that the presented architecture is particularly well suited for redundant robots. An observed conclusion is that the more the system is redundant, the more it can extract from local minima, mainly due to the fact that minima are not concerning the whole population. When an agent is attracted into a minima, the others, when moving, drags him from it, in the same way that a failure on an agent is compensated by the other agents.

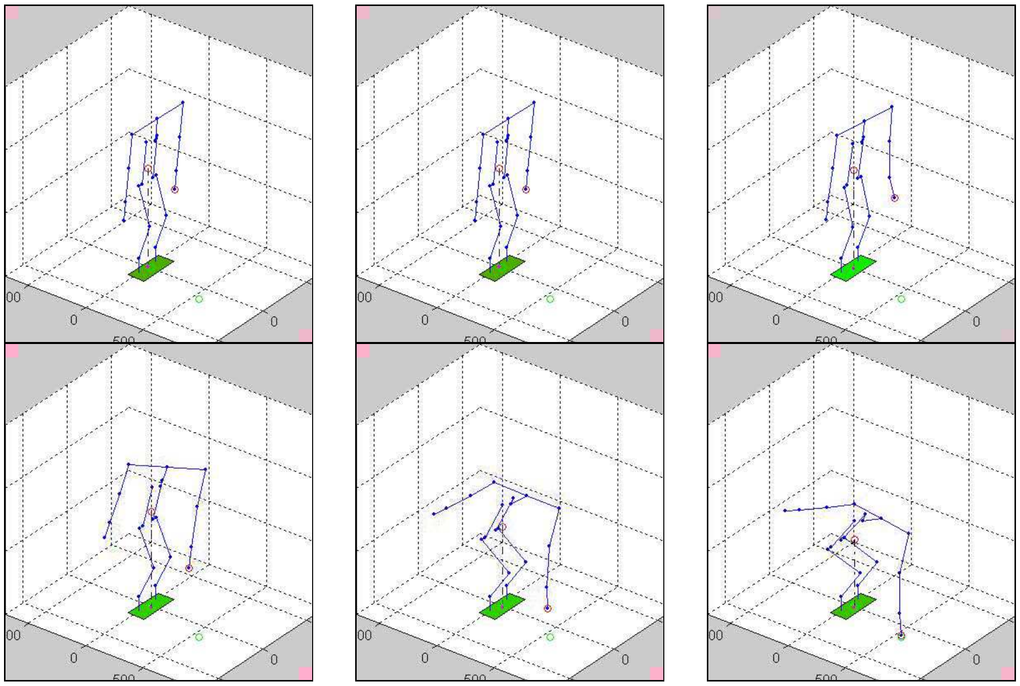
Figure 4 : Example of motion of the robot

5.2 Trajectory

Improvements have been performed on the agent's behavior in order to perform trajectory following. The final goal of these improvements is to perform obstacle avoidance with external obstacles. On the presented results (Figure 5) the desired trajectory has been sampled. During the first part of the motion, the hand of the robot is attracted by the first point, i.e. the first point and the target are merged. Once the hand is close enough to this point the algorithm switches to the second one and so on until reaching the last point.

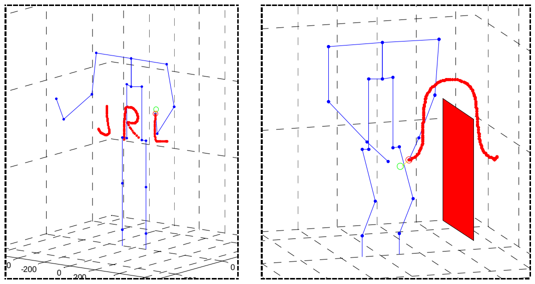

Figure 5 : Example of trajectory followed by the hand of the robot (left), and simulation of obstacle avoidance (right).

5.3 Speed control

Controlling the speed of the end effector is a problem that needs to be addressed. This aspect may be important to control the motion for example to approach the target slowly or to implement minimum jerk model [13]. Controlling the speed of the end effector is equivalent to control the speed of the joints. By speeding up each joint, this will of course modify the speed of the end effector. The speed of each joint is proportional to its coefficient α_i. By multiplying α_i by a coefficient K_s, it becomes possible to speed up (K_s>1) or to slow down (K_s<1) the speed of the joints. The relationship between the joint's speed and the effector speed is given by the kinematics model and is not linear. It means that the function given by equation 5 is not linear, but this function is monotonic.

$$V_{Effector} = f(K_s) \tag{5}$$

Where V_{Target} is the linear speed of the end effector.

It means that the speed of the end effector may be controlled using an iterative process that will optimize K_S in order to converge to the desired speed. At each time step, K_S is simply recomputed according to the equation (6).

$$K_s = \frac{V_{Effector}^{Desired}}{V_{Effector}^{Mesured}} \tag{6}$$

Where

- $V_{Effector}^{Desired}$ is the desired linear speed of the end effector.
- $V_{Effector}^{Mesured}$ is the mesured linear speed of the end effector.

If the desired speed is inferior to the desired one, K_S will be superior to 1, and the robot will increase its joint's speed and vice versa. As K_S is a common coefficient to the whole system the speed controller is currently centralized and the advantages of the distributed system are uunfortunately lost.

5.4 Computation time

Measurement of the computation time is now addressed. Note that the measurement has been made under Matalb, so we may expect better results with a compilated software. The computation for the whole robot requires 0.68ms per cycle. Half of this time is needed to compute the kinematics model, and about 44% for the computation of the derivative, 3% for the computation of the distances and 3% is needed to apply the command on the actuators. On a real distributed architecture, these delays may be divided by the number of agents.

Note that the results have been partially confirmed by the implementation of the algorithm on the OpenHRP platform where the dynamic model of the robot is fully implemented. Unfortunately, as the internal collision detection is not yet operational, an implementation on the real robot is currently too risky.

6 CONCLUSION

This paper presented a distributed architecture for redundant robots. This architecture is inspired from multi-agent systems and deals simultaneously with two objectives: target tracking and stability. Simulation results have shown that the architecture is well suited for redundant robots and is fault tolerant. Extensions have been proposed to control the trajectory and the speed of the end-effector. Unfortunately, the properties of the distributed architecture are lost while controlling the speed. Future works will focus on the study of a distributed speed controller. Currently, as the architecture doesn't take into consideration collisions, an implementation on a real robot is too risky. Future works will be oriented on the design of a collision detection module and on the implementation on a real robot.

References

[1] R. A. Brooks, "A robust layered control system for a mobile robot", IEEE Journal of Robotics and Automation, p. 14-23, 1986.

[2] R. A. Brooks, "New Approaches to Robotics", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

[3] R.C. Arkin, "Cooperation without Communication : Multiagent Schema-Based Robot Navigation", Journal of Robotic Systems, Vol. 9 (3), p.351-364, avril 1992.

[4] T. Balch and R.C. Arkin, "Communication in Reactive Multiagent Robotic Systems", Autonomous Robots, 1, p 27-52, 1994.

[5] R. C. Arkin, "Cooperation without communication: multiagent schemabased robot navigation," J. of Robotic Systems, vol. 9, pp. 351–364, 1992.

[6] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," IEEE Trans. on Robotics and Automation, vol. 14, no. 6, pp. 926–939, 1998.

[7] R. W. Beard, J. Lawton, and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control," IEEE Trans. on Control Systems Technology, vol. 9, no. 6, pp. 777–790, 2001.

[8] W. Kang and H.-H. Yeh, "Coordinated attitude control of multi-satellite systems," Int. J. of Robust and Nonlinear Control, vol. 12, pp. 185–205, 2002.

[9] W. Dong and J. A. Farrell, "Decentralized cooperative control of multiple nonholonomic dynamic systems with uncertainty," Automatica, submitted, 2007.

[10] Yim, M.; Wei-Min Shen; Salemi, B.; Rus, D.; Moll, M.; Lipson, H.; Klavins, E.; Chirikjian, G.S. "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]" IEEE Trans. on robotics, Vol. 14 No. 1 pp. 43-52, 2007.

[11] M. Moll, P. Will, M. Krivokon and W-M. Shen, "Distributed control of the center of mass of a modular robot",IEEE proc. On Intelligent Robots and Systems (IROS'06), 2006, pp.4710-4715.

[12] P. Lucidarme "Evolutionary computation of multi-robot/agent systems", ARS Robotic Books, ISBN 978-3-902613-19-6, 2008.

[13] T. Flash and N. Hogan, « The coordination of Arm movements : an experimentaly confirmed mathematical model », J. of neuroscience, Vol. 5, No. 7, pp.1688-1703, 1985.