Proceedings of the 2nd Biennial IEEE/RAS-EMBS International
Conference on Biomedical Robotics and Biomechatronics
Scottsdale, AZ, USA, October 19-22, 2008

Multi-Agent Hierarchical Architecture Modeling Kinematic Chains employing Continuous RL Learning with Fuzzified State Space

John N. Karigiannis and Costas S. Tzafestas

Abstract— **In the context of multi-agent systems, we are proposing a hierarchical robot control architecture that comprises artificial intelligence (AI) techniques and traditional control methodologies, based on the realization of a learning team of agents in a continuous problem setting. In a multi-agent system, action selection is important for cooperation and coordination among the agents. By employing reinforcement learning (RL) methods in a fuzzified state-space, we accomplish to design a control architecture and a corresponding methodology, engaged in a continuous space, which enables the agents to learn, over a period of time, to perform sequences of continuous actions in a cooperative manner, in order to reach their goal without any prior generated task model. By organizing the agents in a nested architecture, as proposed in this work, a type of problem-specific recursive knowledge acquisition is attempted. Furthermore, the agents try to exploit the knowledge gathered in order to be in position to execute tasks that indicate certain degree of similarity. The agents correspond in fact to independent degrees of freedom of the system, and achieve to gain experience over the task that they collaboratively perform, by exploring and exploiting their state-to-action mapping space. A numerical experiment is presented in this paper, performed on a simulated planar 4 degrees of freedom (DOF) manipulator, in order to evaluate both the proposed hierarchical multi-agent architecture as well as the proposed methodological framework. It is anticipated that such an approach can be highly scalable for the control of robotic systems that are kinematically more complex, comprising multiple DOFs and potentially redundancies in open or closed kinematic chains, particularly dexterous manipulators.**

I. INTRODUCTION

Reinforcement Learning (RL) [1,2,3] is an active area of machine learning research that is also receiving attention from the fields of decision theory and control engineering. Various RL methods [5,6,7] have been employed on multi-agent architectures that target the control of mobile robots operating within a fully or partially observable environment. Moreover, in [8,9,10,11], we have seen cases where single agent architectures employ RL methods in a continuous three-dimensional space, implemented by neural networks. In this paper, we propose a methodology that introduces a hierarchical multi-agent architecture of nested agents that learn to explore their space and reach to their common goal by going through a set of collaborative action-selection steps. Thus, an attempt is made to incorporate specific RL methods in a nested multi-agent architecture and evaluate the overall behavior of the multi-agent system in the domain of dexterous manipulation control. The joints (or links) of the manipulator are considered as distinct agents that utilize RL methods in order to establish certain skills. Within the multi-agent environment that is formulated in the proposed system, every agent in the group is selecting an action independently of the rest by observing and performing an estimate of what the rest will do. The resulting cumulative action of the system is a joint effort (joint actions) of all the nested entities comprising the system. Although autonomous, the agents are closely coupled with each other due the physical connectivity, making the precise cooperation and coordination among them extremely important in order to achieve stability of such a system. The next section describes the proposed framework from the perspective of the algorithm proposed. Section 3, introduces the agents architecture. The following section focuses on the approach that we are proposing on the basis of RL method, both from the side of state space continuity, as well as from the perspective of action selection. Section 5 presents the experimental setup, the results obtained for a simulated 4 dof planar manipulator, as well as a general discussion of results. Section 6, concludes with the current limitations and certain directions for future work.

John N. Karigiannis is Ph.D Candidate at the Faculty of Electrical and Computer Engineering, Division of Signals, Control and Robotics, National Technical University of Athens (NTUA), Zografou, Athens 15773, Greece john@fhw.gr

Costas S. Tzafestas is with the Faculty of Electrical and Computer Engineering, Division of Signals, Control and Robotics, National Technical University of Athens (NTUA), Zografou, Athens 15773, Greece ktzaf@softlab.ntua.gr

II. PROPOSED FRAMEWORK

The proposed framework is an attempt to bridge the gap between high-level and low-level control through a hybrid architecture that integrates both AI learning techniques and traditional control methods. In the hierarchical architecture presented in this paper, the higher layer consists of nested agents formulating a system that incorporates RL component aiming to enable the agents to establish certain policies over time, while the lower end comprises of a simple classic local feedback controller, responsible to drive the corresponding actuators. The basic features and requirements of the proposed control framework are the following:

(a) Each joint is to be assigned an agent having as a function to govern local control at that joint level. The challenge here is to build global dexterity through progressive acquisition of local skills at each local agent level. Now, we should note here that although every joint is to be assigned an agent, the reverse definition is different. Every agent could represent more than a single degree of freedom. So we could have an agent that actually is comprised of two or more degrees of freedom.

(b) Each agent functions locally by observing and performing an estimate of the actions that the rest of the

978-1-4244-2883-0/08/$25.00 ©2008 IEEE

agents could potentially perform; however, a measure of global task performance is supposed to exist, provided by the higher-level agent and distributed to lower agents in the nested architecture, guiding in that manner the reinforcement learning process through the computation of reward function. This reward function must be computed on a continuous scale (instead of waiting for a discrete event of type success or failure to occur), leading to a continuous adaptive dynamic behavior for the system.

(c) The learning process must also function in a continuous state-space, for the system to establish manipulation skills. For the methodology proposed in this paper, a fuzzification step is applied to the readings forming the system state. Learning is then accomplished in a discrete state-action mapping sense; a defuzzification step can then be employed to perform action selection in a continuous domain.

Let us consider a system that is comprised of n dofs (agents $i = 1, \ldots, n$), nested in the following manner presented in Fig. 1. We define as state of agent a_i, $S_i =< q_i, \theta_i, d_i, \vec{g_i} >$, where: q_i is the current joint position of the

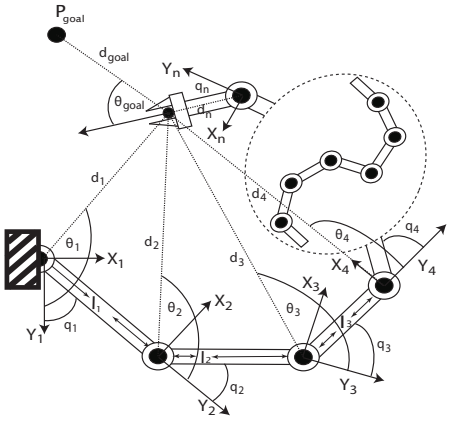

Fig. 1. $n - dof$ Dexterous Manipulator

i^{th} joint, θ_i of the current position of the i^{th} agent, d_i the current Euclidian distance of the i^{th} agent from the end-effector, $\vec{g_i}$ the current vector that describes the goal at the task space (i.e goal with respect to the end effector). The flow of operations depicted in Fig. 2 demonstrates, at an agent-level, the basic structure of the algorithm proposed in this paper, as described below. To achieve continuity over the state space and the action space we have to fuzzify both according to the process described in subsequent section. Every $agent_i$ identifies certain parameters of its state and forwards that information in a nested manner to the agent(s) of the next layer, in order to facilitate them in their own process of identifying their parameters. This process is a top-down process, it starts from the root agent of the hierarchy and travels to the lower ones. During this initial phase the agent identifies its joint parameters as well as its physical characteristics (i.e $agent_i < q_i, l_i >$). Where q_i is the agents joint angle, l_i is its length. It can be seen that q_i defines partially the state of $agent_i$. In order to fully define its state, $agent_i$ requires additional computation of

```
Fuzzification of the State Space
Fuzzification of the Action Space

State Evaluation
      Agent = Agent(i), Tries to fully evaluate its state.
      If success
            Traverse back the Hierarchy of agents and provide them
            with all the available information in order for them to
            successfully define their state

            Go to Action Selection
      Else
            Pass Information gathered to Agent(i+1) and Agent = Agent(i+1)

            Go to State Evaluation

      Loop until all agents have evaluated their states.

Action Selection
      Agent = Agent(i),
      If NO experience exists in Agent
            Stochastically select Action = a(i)
            Stochastically estimate all other Agents actions
      Else if experience exist
            If Agent wants to explore
                  Select Action = not necessary the best action
                  Stochastically estimate all other Agents actions
            If Agent do not want to explore
                  Select Action = action that will generate the greater reward
                  Stochastically estimate all other Agents actions
            End
      End
      Agent = Agent(i+1)
      Loop until all agents have selected Action

Joint Action Execution
Reward Assigned to all Agents
Go Back to State Evaluation
```

Fig. 2. Proposed Algorithm

the following parameters $< \theta_i, d_i, \vec{q_i} >$. Those parameters for $agent_i$ cannot be computed at this phase since their computation requires information from the other agents that comprise the agent community, and that information is not available at the moment. So, since $agent_i$, cannot fully solve the problem of defining its state at the moment, it forwards the partially computed solution computed to the $agent_{i+1}$ residing at the next layer. Similarly, $agent_{i+1}$ calculates those parameters that can be computed and forwards the partial solution to the agents below. The process iterates until we reach an $agent_n$ that succeeds in calculating all those variables that uniquely allow him to compute and define its state. Next, the recursive process continuous by traversing back from $agent_{i+1} \rightarrow agent_i$ providing the agents at the higher layer with the information that they were missing. So when this phase concludes, every agent in the system fully solves its state definition problem, resulting in that way to a multi-agent system with a fully defined state. This iterative/recursive process is repeated in order to define the fuzzified state of every agent in the multi-agent environment. Every agent in the system acts initially without having any prior knowledge, thus acting in a sense stochastically. The $agent_i$ decides to perform a random action a_i and at the same time performs an estimate of what the other agents in the system might choose as their potential actions. That is, each agent, independently of the rest, selects an action, while at the same time performing an estimate of how the other agents are likely to act; each agent thus learns joint actions. This process is again recursive downwards from the top agent $agent_i \rightarrow agent_{i+1}$ where $agent_{i+1}$ is the agent(s) at the lower level. So, $agent_i$ selects action a_i and estimates that the rest of the agents will select $a'_{i+1}, a'_{i+2}, \ldots a'_n$, respectively. Following, in the same manner $agent_{i+1}$ selects an action a_{i+1} while performing an estimate of what the other agents might select (i.e. $a''_{i+2} \ldots a''_n$). After the completion of this recursive process, a specific joint action is

then formulated which the multi-agent system executes. The system provides reward or punishment to the agents for the joint effort that they have demonstrated.

III. AGENT ARCHITECTURE

Fig. 3 demonstrates in a block diagram the proposed hierarchical multi-agent architecture while the subsequent figure, Fig. 4, shows the proposed architecture for an n dof open kinematic chain system. In Fig. 5 we have the interconnections and communication signals exchanged between the different building blocks comprising an agent. So, what we are proposing is a nested multi-agent architecture where uniformity, regarding the structure and the representation of all agents, is a key design principle creating those conditions necessary to expand and further scale up a system potentially utilizing the proposed architecture. More specifically, every

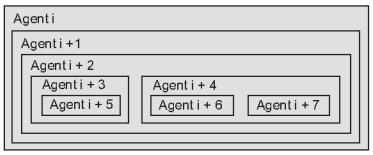

Fig. 3. Multi-Agent Hierarchical Framework for Dexterous Manipulation

agent sees only those agents that are one level below. This means that the agent i passes/inherits the knowledge he obtains only to/from the agent(s) that are visible one level below (in the case of Fig. 3, this means $agent_{i+1}$ only). In the case of $agent_{i+2}$, the observed agents are only $agent_{i+3}$ and $agent_{i+4}$ only. This approach provides to our multi-agent system the ability of a recursive solution searching approach downwards from $agent_i \rightarrow agent_{i+1}$. If an agent cannot solve the problem (or cannot contribute to its solution), then pass the knowledge gathered to the agent(s) below. When an agent is reached where a possible contribution to the problem is feasible, then the agent acts (contributes) and traverses backwards the chain of agents, distributing (propagating) back the knowledge obtained. The basic building blocks in this architecture, with their input/output and requirement characteristics, are described hereafter.

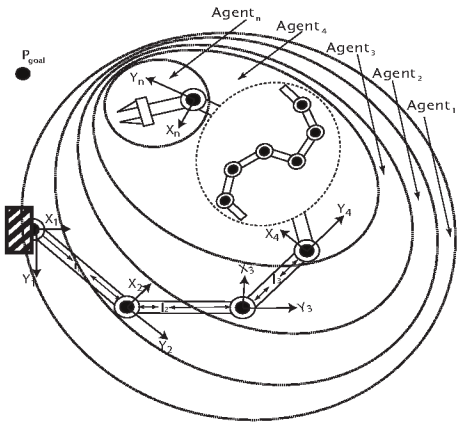

Fig. 4. Multi-Agent architecture on n-dof kinematic chain

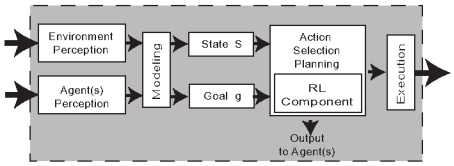

Fig. 5. Basic Building Blocks and Interconnections of an Agent

A. Environment Perception Block

Input: Agents Sensors Input, *Output*: The Environment State interpreted from the specific sensors of the agent. *Requirement(s)*: The signals coming from the specific sensors are mapped/translated to a certain state. That state gives specific information regarding the status of the agent, based on the input that is coming not only from the sensors of the agents (environment perception block) but also potentially from the agents perception block which is gathering information from the agent(s) in higher location(s) in the nested architecture.

B. Agent(s) Perception Block

Input: The state information of the parent agent(s) that the specific agent has. *Output*: The new global state that is formulated. *Requirement(s)*: None

C. Task Modeling Block

Input: Environment & Parent Agent(s) Perception States *Output*: Current State and Goal, *Requirement(s)*: The architecture that is adopted for the control of this multi-agent system is behavior-based. There is no central task model within the agent, something that would result in a situation where moving from one task to another would be very difficult. There exists a local task model for every agent that is created incrementally based on the feedback received after every iteration. Our architecture is utilizing reinforcement learning in order to optimize this dynamically created local task model.

D. Action Selection / Planning Block

Input: Current State and Goal, *Output*: Action to be Executed/ Triplet with the state, the action selected, and the goal to be achieved, *Requirement(s)*: The architecture adopted for the control of this multi-agent system is behavior-based, which means that there is no global action-planning module. What exists is a local action plan encoded in the state-action mapping space. This local action plan of every agent is optimized after each iteration, by utilizing reinforcement learning after each action is executed.

E. Execution Block

Input: Action Selected, *Output*: Control-Level Output
Employing this multi-agent architecture in the case of a 4-dof kinematic chain, we obtain a hierarchical configuration with nested generic agents as shown in Fig. 4. Every joint-agent has certain built-in behaviours (state-action mappings) that get optimised by utilizing reinforcement learning algorithms, while the coordinator-agent is responsible to observe the overall performance and provide feedback to the joint-agents.

IV. CONTINUOUS REINFORCEMENT LEARNING

A. Learning Method

In this section we describe the learning process that has been adopted in our system. Reinforcement learning (RL) methods have been applied in significant number of cases [8,12-17], mostly on mobile robots. In our case, RL method is employed in a quite different domain, namely skill learning and behavior-based multi-agent control of robotic (dexterous) manipulation. Back in 1992 [18], a multi-agent architecture for controlling a multi-fingered hand was presented, but without incorporating any RL methods building skill learning on an agent base. In [19,20], some cases are presented where RL methods are employed in dexterous manipulations, but on a single-agent system architecture. What we are proposing is an attempt to develop a nested multi-agent architecture that aims to enable the control system, through RL methods, to acquire by itself skills and knowledge on how to perform agile manipulation. More formally, we assume a collection of n (homogeneous) agents, each agent $i \in n$ having a finite set of individual actions A_i available to it. Agents repeatedly operate within the framework of the environment posed, in which they each independently select an individual action to perform. In [4], RL is defined as the problem faced by an agent that must learn a behavior through trial-and error interactions with a dynamic environment. In terms of mathematical description, RL has been formalized as a Markov Decision Process (MDP). An MDP has four components: states, actions, transitions and reward distributions. More precisely, an MDP is a 4-tuple, (S, A, T, r) where S denotes a finite set of states, A denotes the action space, T is a probabilistic transition function $T : S \times A \times S \rightarrow [0, 1]$, that denotes the probability for transition from a state s to a new state s' when a certain action a is applied, and $r : S \times A \rightarrow \Re$ is a reward function that denotes the reward for applying a certain action a to a certain state s. Subsequently, we need a formal description for the state of our system. Due to the agent architecture that we have formulated, the state of every individual agent, as well as the state of the entire multi-agent system, are both expressed as $S_t =< q_i, \theta_i, d_i, \vec{g_i} >$. For a 4 dof manipulator, the corresponding state definition of each individual agent and subsequently the state definition of the entire system are $S_t = \{< q_1, \theta_1, d_1, \vec{g_1} >, < q_2, \theta_2, d_2, \vec{g_2} > , < q_3, \theta_3, d_3, \vec{g_3} >, < q_4, \theta_4, d_4, \vec{g_4} >\}$. All agents wish to select actions that maximize the (expected) reward. Each agent contributes its own action component to the joint action that is eventually applied to the environment and determines the transition. The goal is to find a policy that maximizes the sum of discounted reward [2]. Before proceeding we adopt some standard game theory terminology in order to facilitate the discussion below [21]. A randomized policy for an agent i is a distribution $\pi \in \Delta(A_i)$ (where $\Delta(A_i)$ is set of distributions over the agents action set A_i). Intuitively, $\pi(a^i)$ denotes the probability of agent i selecting the individual action a^i. A policy π is deterministic if $\pi(a^i) = 1$ for some $a^i \in A_i$. A collection of policies for each agent i is called policy profile, $\Pi = \{\pi_i : i \in n\}$. The expected value of acting according to a fixed profile can easily be determined. If each $\pi \in \Pi$ is deterministic, we can think of Π as a joint action. A reduced profile for agent i, is a policy profile for all agents but i (denoted Π_{-i}). Given a profile Π_{-i} , a policy π_i is a best response for agent i if the expected value of the policy profile $\Pi_{-i} \cup \{\pi_i\}$ is maximal for agent i ; that is, agent i could not do better using any other policy π'_i. In the following section we refer to the requirement of continuous state-space, introducing an issue that has to be resolved, which is the infinite number of states [1], which is discussed hereafter.

B. State-Space Fuzzification for Continuous Problem Sets

In a continuous state-space the number of parameters to be learned by the agent grows exponentially as the number of states increases. In order to achieve the desired continuity in the state-space without building huge lookup tables storing all the parameters of the agent, each of the parameters defining the state of each agent and the state of the system (joint angles, angular displacements, Euclidean distance and all other signals required) are fuzzified using membership functions of the type shown in Fig. 6. Here, each joint (agent) continuous angular position, ranging from 0 to 2π, is divided into 8 discrete states (but with assigned weight, for each state, from 0 to 1). For this reason, standard triangular (equidistant) fuzzy membership functions are used. The action-selection

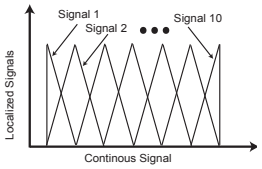

Fig. 6. The fuzzified state space for the agent

space is also fuzzified in the same manner. In the sequel, the action selection and reward computation functions are described.

C. Action Selection and Reward Function

Action selection is significantly difficult if there are multiple optimal joint actions. If the joint actions are chosen randomly, or in some way reflecting personal biases, then there is a risk to select a suboptimal or uncoordinated joint action. So, we have the general problem of equilibrium selection (or joint action selection) which can be addressed by several ways. One way is the communication among the agents [22]; another is to introduce conventions or rules that restrict behaviors and so to ensure coordination. What we are proposing results in a coordination among the agents action through a repeated performance of the specific task by the same agents. In our action selection mode, each agent i keeps a count of the number of times a specific action has been performed in the past by the same agent (as well as by its collaborative agents). That concept although simple, is some times quite effective, and is known as fictitious play [23][24]. More precisely, each agent i keeps a count $C^j_{a^j}$, for each

$j \in n$, and $a^j \in A_j$, of the number of times agent j has used action a^j in the past. When a task is assigned to our multi-agent system, agent i treats the relative frequencies of each of agents $j's$ moves as indicative of $j's$ current policy. That is, for each agent j, i assumes agent j performs action $a^j \in A_j$ with probability $Pr^i_{a^j} = C^j_{a^j} / \sum_{b^i \in A_j} C^j_{b^j}$. We note that most models (in game theory) assume that each agent can observe the actions executed by its counterparts with certainty. What we actually employed is something more general that allows each agent to obtain an observation that is related stochastically to the actual joint action selected. Since action selection is more difficult when agents are not aware of the rewards associated with various joint actions, and since the problem treated in this paper belongs in this category, the utilization of RL by the agents, in order to estimate based on previous experience the expected reward associated with individual and joint action, appears to be an acceptable approach. Q-learning algorithm developed by Watkins [25] is the most frequently used RL algorithm. An agent estimates the utility for doing each of its actions, chooses an action based on a selection function of the expected values, observes the reward, and then updates the Q-value or the estimate of the utility of that action. In the case of a stateless setting we have an agent updating its estimate $Q(a)$ as follows: $Q(a) \leftarrow Q(a) + \lambda(r - Q(a))$ where action a was performed resulting in reward r. Here λ is the learning rate ($0 \leq \lambda \leq 1$) governing to what extent the new sample replaces the current estimate. The next issue concerns the action selection function, which is particularly important, since effective learning requires sufficient exploration. An agent can try its actions at any time; there is no requirement to perform actions that are currently estimated to be best. Of course, if we want to enhance overall performance during learning, it makes sense to bias selection towards better actions. In order to estimate the probability of choosing an action, we employed Boltzmann distribution:

$$\pi(a^i) = \frac{e^{\frac{EV(a^i)}{T}}}{\sum_{a^{i'} \in A_i} e^{\frac{EV(a^{i'})}{T}}}$$

where EV is the expected value of an action and T is the temperature parameter that is controlled to diminish over time so that the exploitation probability is increased. Temperature determines the likelihood for an agent to explore other actions: when T is high even when the EV of an action is high, an agent may still choose an action that appears to be less desirable. This exploration strategy is especially important in stochastic environments like the one we are examining, where payoffs received for the same action combination may vary. For effective exploration, high temperature is used in the early stage of the task. The temperature is then decreased over time to favour exploitation, as the agent is more likely to have discovered the true values of different actions. The temperature as a function of iterations is given by: $T(x) = 1 + T_{max} * e^{-sx}$ where x is the iteration number, s is the rate of decay and T_{max} is the starting temperature. Now let us elaborate on the definition of the expected value EV. The presence of multiple agents, each one learning simultaneously with others, is a potential impediment to the successful employment of Q-learning (and RL in general) in multi-agent settings like the one considered in this paper. When agent i is learning the value of its actions in the presence of other agents, it is learning in a non-stationary environment. Thus, convergence of the Q-values is not guaranteed. What we need is each agents policy to settle. This is a key issue and is discussed below. In general there are two distinct ways in which Q-learning could be applied in a multi-agent system; the Independent and the Joint action learner algorithm [26] [27] [28]. In an Independent Learner algorithm each agent learns its Q-values regardless of what the other agents are performing. This method is appropriate to be employed when and agent has no reason to believe that other agents are acting strategically. Joint action learner algorithm is the one where the agents do not learn Q-values of their individual actions but the Q-values of their joint actions. This implies that each agent can observe the actions of other agents. Each agent in such a system maintains beliefs about the policies of other agents. So, an agent i assesses the expected value EV of its individual action a^i to be

$$EV(a^i) = \sum_{a^{-i} \in A_{-i}} Q(a^{-i} \cup \{a^i\}) \prod_{j \neq i} \{Pr^i{}_{a^{-i}[j]}\}$$

The reward that the agent receives at time instance t, after selecting certain action A_t and moving to a new state, is defined by the reward function R_t which is the formulated as follows:

$$R_t = \begin{cases} e^{-c * Dist_{Goal}(x)} \\ if (Dist_{Goal}(x) \leq Dist_{min}) \wedge (\Delta Dist_{Goal}) \leq 0) \\ -2 \\ if (Dist_{Goal}(x) > Dist_{min}) \\ -1 \\ if (Dist_{Goal}(x) < Dist_{min}) \wedge (\Delta Dist_{Goal}) > 0) \end{cases}$$

where $Dist_{Goal(x)}$ is the distance from the goal at the iteration time x . $Dist_{min}$ is a threshold distance after which the agents starts receiving reward. $\Delta Dist_{Goal}$ is the rate of change of distance from the goal.

D. Proposed RL Based Robot Control Architecture

The robot control architecture employed has three layers. The first layer is the error observation layer, the second layer is the action selection, and the last one is the robotic mechanism. The first layer receives as input the desired goal for our multi-agent system along with the feedback from the last layer (as shown in Fig. 7). The error observation layer provides input to the next layer, which is the action selection layer. We can see that the action selection layer is coupled with the RL module. The RL module, augmenting the action selection mechanism, receives the error observation and without any initial previous knowledge regarding commanded joint-level motion, provides appropriate input to the action selection mechanism, in order for certain action(s) to be generated. Subsequently the action selection mechanism generates the joint-level motion for the robot(s),

TABLE I

EXPERIMENTAL PARAMETERS

	Link 1	Link 2	Link 3	Link 4
Length	3 cm	2 cm	4 cm	3 cm
Initial Joint Angle	0°	20°	30°	40°
Step: Joint Angle	18°	18°	18°	18°
Step: Goal Angle	40°	40°	40°	40°
Overlapping of function f_1	15%	15%	15%	15%
Overlapping of function f_2	25%	25%	25%	25%

based on stochastic models, while providing information to the RL module regarding the probabilistic distribution that was assigned to the different action(s) by the action selection process. The commanded joint-level motion propagates to the third layer which corresponds to the robotic mechanism. This last layer generates the robot motion for the agent(s) involved in the setup.

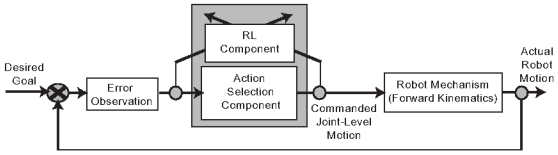

Fig. 7. RL-Based Robot Control Architecture

V. EXPERIMENTAL SETUP - RESULTS AND DISCUSSION

To evaluate the performance of the proposed architecture, we have performed a series of numerical experiments, considering a simulated kinematic chain that consists of four links. The detailed parameters used in the simulation are shown in Table I. In order to achieve continuity over the joint space of the system, the space is fuzzified with several signals. In total, we have for every joint angle 20 signals with overlapping of 15%. Similarly, is the case of task space again we need to have continuity of the state definition so what we have is nine signals each one spanning over a range of 40 degrees (in total nine localized signals as can be seen in Fig. 8). The overlapping of those signals is 25%. The task for this multi-agent environment is to respect the

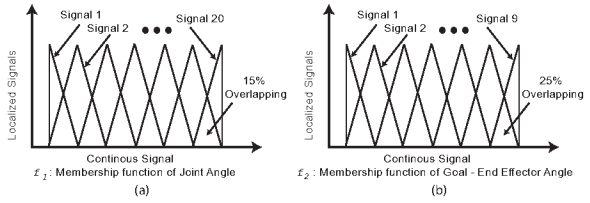

Fig. 8. Fuzzified Joint Angle State Space and Goal - End Effector Angle State Space

limitiations that their physical interconnectivity poses, and without any previous experience or knowledge or task model, to learn how to collaborate and communicate information among them, in order to reach the goal position. The first phase of the process is the training of the multi-agent system. The system is trained for 200 epochs, each epoch lasting 500 sec. This means that we allow the group of four agents to come up with a solution to the problem assigned within the time slot of 500 seconds. In any case (whether a solution has been found or not), the knowledge obtained within the specific time interval is stored and the agents are expected to further improve their behavior during the subsequent epochs. The behavior of the agents at the beginning of their collaborative activity is focusing on exploration; thus, in order to satisfy this requirement, the parameters in the equation $T(x) = 1 + T_{max} * e^{-sx}$, are defined as follows, for $x = [1 \dots 500]$, we have $T_{max} = 100$, the maximum temperature, and the decay factor $s = 0.35$, which indicates that we have high exploration at the first 30 seconds of an epoch (meaning that during this first period the agents might select actions that do not have the highest Q-values) while in the remaining time the agents select those actions with the higher Q-values. Moreover the learning rate is low, which means that we promote a slow learning approach (i.e $\lambda = 0.1$). The agents interacting with the environment they receive rewards. In our simulation $Dist_{min} = 3$ meaning that the agents do not receive any positive reward if the end-effector is not closer that 3 units from the goal position. Fig. 9, summarizes the above training results. The initial set of results presented demonstrates the agents in four distinct epochs. During epoch 1, we see the agents exploring their workspace, a behavior which results in an error on position, with respect to the goal position that they are trying to reach, an error that is not reduced over the entire time duration of this epoch. We allow the process to repeat itself for another ten epochs. In epoch 10, a similar behavior can be observed with a slight improvement. In epoch 100 and 200, however, the evolution of the agents collaborative behavior to reduce the error becomes apparent, since the actions with highest Q-values are now dominating the joint actions that the agents are selecting. In Fig. 10, we run again the same test with

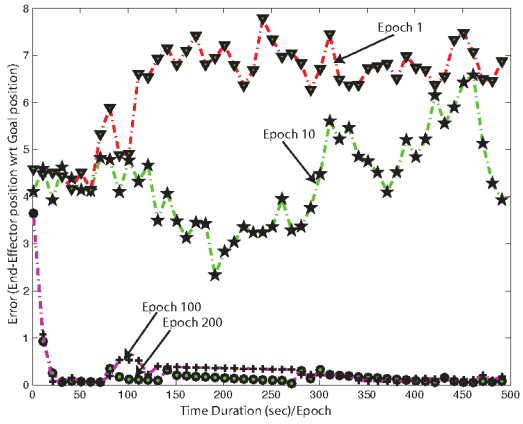

Fig. 9. Error over time over Different Epochs for factor $s = 0.35$

only a minor modification, the decay factor in now increased to $s = 0.75$. The results show that, already by epoch 10,

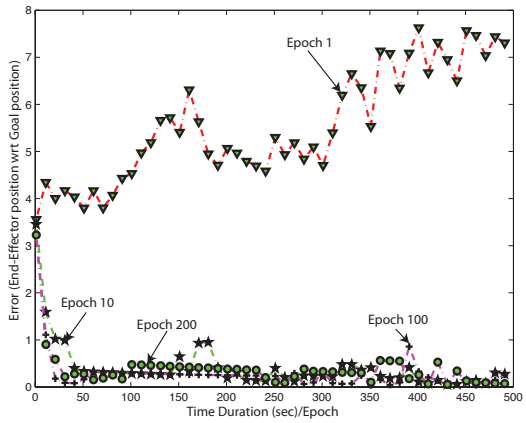

Fig. 10. Error over time over Different Epochs for factor $s = 0.75$

TABLE II
DIFFERENT SOLUTIONS DERIVED BY OUR ARCHITECTURE VERSUS THE PSEUDOINVERSE METHOD $J^{\dagger}$

	$J^{\dagger}$	$s = 0.75$			$s = 0.35$			$s = 0.05$		
	c_1	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3
Q_1	0	-2.0	-5.8	-11.8	-13.9	-39.0	29.4	-46.8	-46.9	-27.6
Q_2	24.1	63.9	41.9	57.8	39.1	137.3	22.6	120.8	119.0	93.2
Q_3	53.7	0.1	40.1	28.1	64.7	-33.3	-2.9	5.3	25.7	6.3
Q_4	59.8	84.1	59.8	55.9	26.1	27.2	120.3	10.4	-25.0	43.9

the positioning error is now considerably reduced. In the subsequent epochs we again see that the agents behavior does settle to a sequence of actions that have the highest Q-values. Having seen the error convergence of the proposed approach, another important issue concerns evaluating the solution(s) provided by this mechanism. To solve the inverse kinematic problem of the 4 dof kinematic chain, an iterative method can be employed, based on the computation of the pseudoinverse of the Jacobian matrix, $J^{\dagger}$. The Jacobian matrix J in our case is a 3×4 matrix consisting of the following column vectors $\vec{J_1}, \vec{J_2}, \vec{J_3}, \vec{J_4}$. Each $\vec{J_i}$ vector $(i = 1 \ldots 4)$ can be computed as the cross product of the vector representing the axis of rotation of the i^{th} link against the vector expressing the distance between the end-effector and the corresponding i^{th} joint. We can then write $\Delta\theta = J^{\dagger} * \Delta p$, where $\Delta\theta$ is the increment at the joint angle that causes the end effector of the chain to move by Δp where the pseudoinverse $J^{\dagger}$ is equal to $J^T * (J * J^T)^{-1}$. Employing this iterative method we obtain an optimum set of angular displacements Q_i for $i = 1 \ldots 4$ as shown in Table II (optimum in the least-square sense, leading to a minimum joint displacement for every link in the kinematic chain). The corresponding solutions obtained with our proposed architecture are also shown in the table, varying according to the decay factor selected. Fig. 11, depicts the results contained in Table II, showing that in almost all different decay factors the multi-agent system proposed in this paper generates quite natural solutions, while in certain cases, close to the optimum one as can be see in the corresponding schematics (stick diagrams). When

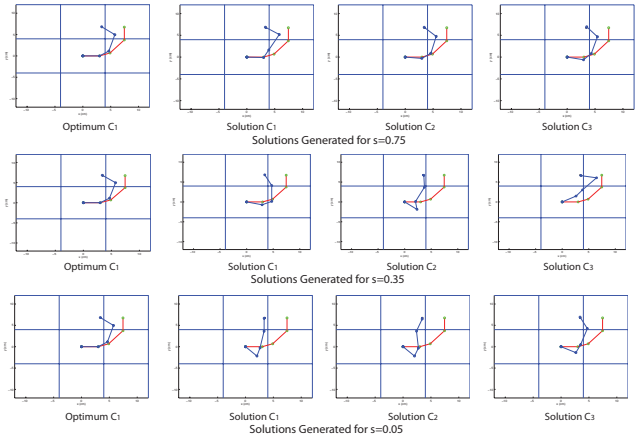

Fig. 11. Generated Solutions for $s = 0.75$, $s = 0.35$ and $s = 0.05$

the training is completed and the agents have learned how to reach their goal, the next step is to see how the agents use the knowledge acquired and how, without any additional training, they can explore and reach targets that are related to the ones trained. The approach we follow to test the behavior of our system is defined on the basis of selecting goals that are located within an area of a given radial distance measured from the initial (trained) goal position. We want to see whether our system is in a position to handle existing knowledge and exploit it further. So, having as a center of our potential exploitation area the initial goal position, we define five areas, each one having a different radius $(r_1 = 0.7, r_2 = 1.4, r_3 = 2.1, r_4 = 2.8, r_5 = 4.2)$. For each of these areas, we assign 5 different new goals to our system; so in total we have twenty-five new goals. We average the errors obtained per area and the results are presented in Fig. 12. We can see that points in areas close to the initial goal position are indeed reached without any further exploration.

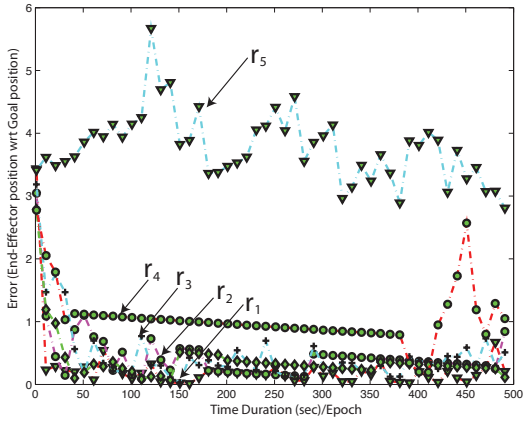

Fig. 12. Error observed while reaching new goals at five different areas

VI. CONCLUSION AND FUTURE WORK

By combining Reinforcement Learning (RL) methods and traditional control approaches over a hierarchical multi-agent architecture, in a fuzzified state-space, we obtain a hybrid robot architecture, with respect to control topology, which is applied in a continuous state-space to perform a robot manipulation task. The nested, self-evolving multi-agent framework proposed in this paper, which constitutes in fact an implementation of a recursive mechanism able to search for solution(s) in a specific problem, appears to be particularly suitable for robot control problems where increased degree of dexterity is required. The basic advantage of such an approach is that no global task model (in the case-study of this paper, no robot inverse kinematics model) is needed. Moreover, the proposed multi-agent system, owing to its homogeneous characteristics (all agents obey the same structural/modular internal architecture), as well as to its hierarchical formation, facilitates scaling of the system to more complex structures. Fig. 13 depicts a potential application of the proposed framework, where a rather more complicated multi-agent environment could be envisaged. By employing the proposed framework in the domain of dexterous manipulation, we believe that challenging problems in this specific area can be tackled in a very elegant, interesting and powerful way (for instance, in the sense of modularity, robustness and extensibility). Similar (in some ways equivalent) problem settings, like grasp planning, locomotion control, or designing optimal climbing (and generally gaiting) patterns, could also be approached within the same framework, lending to the notions of evolving cooperative learning and developmental robotics. Bringing together the areas of multi-agent architectures, machine learning and dexterous robotics, will create new challenges, both theoretic and application-oriented, for all these domains of research.

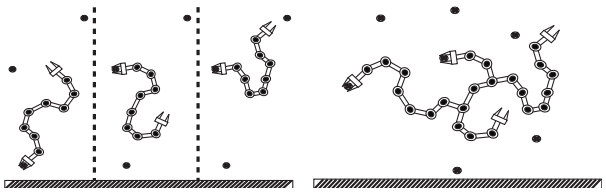

Fig. 13. Climbing Robotic Chain

REFERENCES

[1] R.S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA; 1998.

[2] D. P. Bertsekas and J. N, Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont MA; 1996.

[3] Peter Dayan and L. F. Abbott, *Theoretical Neuroscience, Computational and Mathematical Modeling of Neural Systems*, MIT Press, Cambridge, MA; 2001.

[4] Kaelbling, L.P., Littman, M.L. and Moore, A.W. Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, 4, 237-285, 1996.

[5] Jelle R. Kok, Nikos Vlassis, "Sparse Tabular Multiagent Q-Learning", *Proceedings of Annual Machine Learning Conference of Benelearn* 2004.

[6] Javier Zamora, Jose del R. Millan, Antonio Murciano, "Learning and Stabilization of Altruistic Behaviors in multi-agent systems by reciprocity", *Biological Cybernetics 78*, 197-205, Springer-Verlag 1998.

[7] Javier Zamora, Jose del R. Millan, Antonio Murciano, "Specialization in multi-agent systems through learning", *Biological Cybernetics 76*, 375-382, Springer-Verlag 1997.

[8] Takashi Takahashi, Toshio Tanaka, Kenji Nishida, Takio Kurita, "Self-Organization of Place Cells and Reward-Based Navigation for a mobile Robot", *ICONIP* 2001.

[9] Pawel Wawrzynski and Andrsej Pacut, "A Simple Actor-Critic Algorithm for Continuous Environments", *Proceedings of the 10th MMAR Int. Conf.* , Miedzyzdroje, Poland pp 1143-1149 IEEE 2004.

[10] Getachew Hailu, "Symbolic Structures in Numeric Reinforcement for learning optimum robot trajectory", *Robotics and Autonomous Systems 37*, 53-68, Elsevier 2001.

[11] Kenji Doya "Temporal Difference Learning in Continuous Time and Space", *Advances in Neural Information Processing Systems 8*, MIT Press, 1996.

[12] Toshiyuki Kondo, Koji Ito, "A Reinforcement Learning using Adaptive State Space Construction Strategy for Real Autonomous Mobile Robots", *Robotics and Autonomous Systems*, vol. 46 no.2 pp. 111-124 Elsevier, (2004)

[13] Masaru IIDA, Masanori SUGISAKA, and Katsunari SHIBATA, "Application of Direct-Vision-Based Reinforcement Learning to a Real Mobile Robot", *Artificial Life and Robotics*, Vol. 7, No. 3, pp. 102-106, 2004

[14] Katsunari Shibata, Koji Ito and Yiochi Okabe, "Direct-Vision-Based Reinforcement Learning in Going to Target Task with an Obstacle and with a Variety of Target Sizes", *Proc. of NEURAP(Neural Networks and their Applications)*98, pp. 95-102. 1998

[15] Katsunari Shibata, Masanori Sugisaka and Koji Ito, "Fast and Stable Learning in Direct-Vision-Based Reinforcement Learning", *Proc. of Intl Sympo. On Artificial Life and Robotics (AROB) 6th*, pp. 562-565, 2001.

[16] Katsunari Shibata and Yoichi Okabe, "Smoothing-Evaluation Method in Delayed Reinforcement Learning", 1995

[17] Katsunari Shibata and Yoichi Okabe, "A Robot that Learns an Evaluation Function for Acquiring of Appropriate Motions" *World Congress on Neural Networks-San Diego, 1994 Intl. Neural Network Society Annual Meeting*, Vol.2. Jun. 1994, pp. II. 29-II34.

[18] Toshihiro Matsui, Tooru Omata, and Yasuo Kaniyoshi, "Multi-Agent Architecture for Controlling a Multi-finger Robot", *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh. NC 1992.

[19] Katsunari Shibata and Koji Ito, " Effect of Force Load in Hand Reaching Movement Acquired by Reinforcement Learning", *ICONIP02, Proceedings of the 9th International Conference on Neural Information Processing*, Computational Intelligence for the E-Age.

[20] Katsunari Shibata and Koji Ito, "Hidden Representation after Reinforcement Learning of Hand Reaching Movement with Variable Link Length", *Proc. of IJCNN(Intl Conf. on Neural Networks) 2003*, 1475-674, pp. 2619-2624, 2003.7

[21] R. B. Myerson, *Game Theory: Analysis of conflict*, Harvard University Press, Cambridge 1991

[22] Y. Shoham and Tennenholtz, "On the synthesis of useful social laws for artificial agent societies", *Proc. AAAI-92*, pp. 276-281, San Jose, 1992

[23] D. Fundenberg and D. M. Kreps, "Lectures on Learning and Equilibrium in Strategic Form Games", *CORE Foundation* Louvain-La-Neuve, Belgium, 1992

[24] G. W. Brown, "Iterative solution of games by fictitious play." In T.C. Koopmans editor, *Activity Analysis of Production and Allocation*, Wiley, New York 1951

[25] Watkins,C. "Learning from Delayed Rewards", *PhD Thesis*, University of Cambidge,England, 1989

[26] Martin Lauer, Martin Riedmiller, "Reinforcement Learning for Stochastic Cooperative Multi-Agent Systems," *aamas, pp. 1516-1517, Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3 (AAMAS'04)*, 2004

[27] McGlohon, M. and S. Sen. "Learning to cooperate in multi-agent systems by combining Q- learning and evolutionary strategy." *World Conference on Lateral Computing*, December 2004

[28] Caroline Claus and Craig Boutilier, "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems", *AAAI/IAAI*, pp. 746-752, 1998