Open access • Proceedings Article • DOI:10.1109/ICARCV.2006.345351

Decentralized Reinforcement Learning Control of a Robotic Manipulator

Lucian Busoniu, Bart De Schutter, Robert Babuska

Institutions: Delft University of Technology

Published on: 01 Dec 2006 - International Conference on Control, Automation, Robotics and Vision

Topics: Reinforcement learning and Multi-agent system

Decentralized Reinforcement Learning Control of a Robotic Manipulator

Lucian Buşoniu Bart De Schutter Robert Babuška
Delft Center for Systems and Control
Delft University of Technology
2628 CD Delft, The Netherlands
Email: {i.l.busoniu,b.deschutter,r.babuska}@tudelft.nl

***Abstract*— Multi-agent systems are rapidly finding applications in a variety of domains, including robotics, distributed control, telecommunications, etc. Learning approaches to multi-agent control, many of them based on reinforcement learning (RL), are investigated in complex domains such as teams of mobile robots. However, the application of decentralized RL to low-level control tasks is not as intensively studied. In this paper, we investigate centralized and decentralized RL, emphasizing the challenges and potential advantages of the latter. These are then illustrated on an example: learning to control a two-link rigid manipulator. Some open issues and future research directions in decentralized RL are outlined.**

***Keywords*—multi-agent learning, decentralized control, reinforcement learning**

I. Introduction

A multi-agent system (MAS) is a collection of interacting agents that share a common environment (operate on a common process), which they perceive through sensors, and upon which they act through actuators [1]. In contrast to the classical control paradigm, that uses a single controller acting on the process, in MAS control is distributed among the autonomous agents.

MAS can arise naturally as a viable representation of the considered system. This is the case with e.g., teams of mobile robots, where the agents are the robots and the process is their environment [2], [3]. MAS can also provide alternative solutions for systems that are typically regarded as centralized, e.g., resource management: each resource may be managed by a dedicated agent [4] or several agents may negotiate access to passive resources [5]. Another application field of MAS is decentralized, distributed control, e.g., for traffic or power networks.

Decentralized, multi-agent solutions offer several potential advantages over centralized ones [2]:

- Speed-up, resulting from parallel computation.
- Robustness to single-point failures, if redundancy is built into the system.
- Scalability, resulting from modularity.

MAS also pose certain challenges, many of which do not appear in centralized control. The agents have to *coordinate* their individual behaviors, such that a coherent joint behavior results that is beneficial for the system. Conflicting goals, inter-agent communication, and incomplete agent views over the process, are issues that may also play a role.

The multi-agent control task is often too complex to be solved effectively by agents with pre-programmed behaviors. Agents can do better by *learning* new behaviors, such that their performance gradually improves [6], [7]. Learning can be performed either online, while the agents actually try to solve the task, or offline, typically by using a task model to generate simulated experience.

Reinforcement learning (RL) [8] is a simple and general framework that can be applied to the multi-agent learning problem. In this framework, the performance of each agent is rewarded by a scalar signal, that the agent aims to maximize. A significant body of research on multi-agent RL has evolved over the last decade (see e.g., [7], [9], [10]).

In this paper, we investigate the single-agent, centralized RL task, and its multi-agent, decentralized counterpart. We focus on cooperative low-level control tasks. To our knowledge, decentralized RL control has not been applied to such tasks. We describe the challenge of coordinating multiple RL agents, and briefly mention the approaches proposed in the literature. We present some potential advantages of multi-agent RL. Most of these advantages extend beyond RL to the general multi-agent learning setting.

We illustrate the differences between centralized and multi-agent RL on an example involving learning to control a two-link rigid manipulator. Finally, we present some open research issues and directions for future work.

The rest of the paper is organized as follows. Section II introduces the basic concepts of RL. Cooperative decentralized RL is then discussed in Section III. Section IV introduces the two-link rigid manipulator and presents the results of RL control on this process. Section V concludes the paper.

II. Reinforcement Learning

In this section we introduce the main concepts of centralized and multi-agent RL for deterministic processes. This presentation is based on [8], [11].

A. Centralized RL

The theoretical model of the centralized (single-agent) RL task is the Markov decision process.

Definition 1: A *Markov decision process* is a tuple $\langle X, U, f, \rho \rangle$ where: X is the discrete set of process states, U is the discrete set of agent actions, $f : X \times U \to X$ is the

1–4244–0342–1/06/$20.00

state transition function, and $\rho : X \times U \to \mathbb{R}$ is the reward function.

The process changes state from x_k to x_{k+1} as a result of action u_k, according to the state transition function f. The agent receives (possibly delayed) feedback on its performance via the scalar reward signal $r_k \in \mathbb{R}$, according to the reward function ρ. The agent chooses actions according to its *policy* $h : X \to U$.

The learning goal is the maximization, at each time step k, of the discounted return:

$$R_k = \sum\nolimits_{j=0}^{\infty} \gamma^j r_{k+j+1}, \tag{1}$$

where $\gamma \in (0, 1)$ is the discount factor. The *action-value function* (Q-function), $Q^h : X \times U \to \mathbb{R}$, is the expected return of a state-action pair under a given policy: $Q^h(x, u) = E\{R_k \,|\, x_k = x, u_k = u, h\}$. The agent can maximize its return by first computing the optimal Q-function, defined as $Q^*(x, u) = \max_h Q^h(x, u)$, and then choosing actions by the greedy policy $h^*(x) = \arg\max_u Q^*(x, u)$, which is optimal (ties are broken randomly).

The central result upon which RL algorithms rely is that Q^* satisfies the Bellman optimality recursion:

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u' \in U} Q^*(f(x, u), u') \ \forall x, u. \tag{2}$$

Value iteration is an offline, model-based algorithm that turns this recursion into an update rule:

$$Q_{\ell+1}(x, u) = \rho(x, u) + \gamma \max_{u' \in U} Q_\ell(f(x, u), u') \ \forall x, u. \tag{3}$$

where ℓ is the iteration index. Q_0 can be initialized arbitrarily. The sequence Q_ℓ provably converges to Q^*.

Q-learning is an online algorithm that iteratively estimates Q^* by interaction with the process, using observed rewards r_k and pairs of subsequent states x_k, x_{k+1} [12]:

$$\begin{aligned} Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \\ \alpha\big[r_{k+1} + \gamma \max_{u' \in U} Q(x_{k+1}, u') - Q_k(x_k, u_k)\big], \end{aligned} \tag{4}$$

where $\alpha \in (0, 1]$ is the learning rate. The sequence Q_k provably converges to Q^* under certain conditions, including that the agent keeps trying all actions in all states with nonzero probability [12]. This means that the agent must sometimes *explore*, i.e., perform other actions than those dictated by the current greedy policy.

B. Multi-Agent RL

The generalization of the Markov decision process to the multi-agent case is the Markov game.

Definition 2: A *Markov game* is a tuple $\langle A, X, \{U_i\}_{i \in A}, f, \{\rho_i\}_{i \in A} \rangle$ where: $A = \{1, \ldots, n\}$ is the set of n agents, X is the discrete set of process states, $\{U_i\}_{i \in A}$ are the discrete sets of actions available to the agents, yielding the joint action set $\boldsymbol{U} = \times_{i \in A} U_i$, $f : X \times \boldsymbol{U} \to X$ is the state transition function, and $\rho_i : X \times \boldsymbol{U} \to \mathbb{R}, i \in A$ are the reward functions of the agents.

Note that the state transitions, agent rewards $r_{i,k}$, and thus also the agent returns $R_{i,k}$, depend on the *joint action* $\boldsymbol{u}_k = [u_{1,k}^{\mathrm{T}}, \ldots, u_{n,k}^{\mathrm{T}}]^{\mathrm{T}}, \boldsymbol{U}_k \in \boldsymbol{U}$, $u_{i,k} \in U_i$. The policies $h_i : X \times U_i \to [0, 1]$ form together the joint policy $\boldsymbol{h}$. The Q-function of each agent depends on the joint action and is conditioned on the joint policy, $Q_i^{\boldsymbol{h}} : X \times \boldsymbol{U} \to \mathbb{R}$.

A fully cooperative Markov game is a game where the agents have identical reward functions, $\rho_1 = \ldots = \rho_n$. In this case, the learning goal is the maximization the common discounted return. In the general case, the reward functions of the agents may differ. Even agents which form a team may encounter situations where their immediate interests are in conflict, e.g., when they need to share some resource. As the returns of the agents are correlated, they cannot be maximized independently. Formulating a good learning goal in such a situation is a difficult open problem (see e.g., [13]–[15]).

III. Cooperative Decentralized RL Control

This section briefly reviews approaches to solving the coordination issue in decentralized RL, and then mentions some of the potential advantages of decentralized RL.

A. The Coordination Problem

Coordination requires that all agents coherently choose their part of a desirable joint policy. This is not trivial, even if the task is fully cooperative. To see this, assume all agents learn in parallel the common optimal Q-function with, e.g., Q-learning:

$$\begin{aligned} Q_{k+1}(x_k, \boldsymbol{u}_k) = Q_k(x_k, \boldsymbol{u}_k) + \\ \alpha\big[r_{k+1} + \gamma \max_{\boldsymbol{u}' \in \boldsymbol{U}} Q(x_{k+1}, \boldsymbol{u}') - Q_k(x_k, \boldsymbol{u}_k)\big]. \end{aligned} \tag{5}$$

Then, in principle, they could use the greedy policy to maximize the common return. However, greedy action selection breaks ties randomly, which means that in the absence of additional mechanisms, different agents may break a tie in different ways, and the resulting joint action may be suboptimal.

The multi-agent RL algorithms in the literature solve this problem in various ways.

Coordination-free methods bypass the issue. For instance, in fully cooperative tasks, the Team Q-learning algorithm [16] assumes that the optimal joint actions are unique (which will rarely be the case). Then, (5) can directly be used.

The agents can be *indirectly* steered toward coordination. To this purpose, some algorithms learn empirical models of the other agents and adapt to these models [17]. Others use heuristics to bias the agents toward actions that promise to yield good reward [18]. Yet others directly search through the space of policies using gradient-based methods [11].

The action choices of the agents can also be *explicitly coordinated* or negotiated:

- Social conventions [19] and roles [20] restrict the action choices of the agents.
- Coordination graphs explicitly represent where coordination between agents is required, thus preventing the agents from engaging in unnecessary coordination activities [21].

– Communication is used to negotiate action choices, either alone or in combination with the above techniques.

B. Potential Advantages of Decentralized RL

If the coordination problem is efficiently solved, learning speed might be higher for decentralized learners. This is because each agent i searches an action space U_i. A centralized learner solving the same problem searches the joint action space $\boldsymbol{U} = U_1 \times \cdots \times U_n$, which is exponentially larger.

This difference will be even more significant in tasks where not all the state information is relevant to all the learning agents. For instance, in a team of mobile robots, at a given time, the position and velocity of robots that are far away from the considered robot might not be interesting for it. In such tasks, the learning agents can consider only the relevant state components and thus further decrease the size of the problem they need to solve [22].

Memory and processing time requirements will also be smaller for smaller problem sizes.

If several learners solve similar tasks, then they could gain further benefit from sharing their experience or knowledge.

IV. Example: Two-Link Rigid Manipulator

A. Manipulator Model

The two-link manipulator, depicted in Fig. 1, is described by the nonlinear fourth-order model:

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + G(\theta) = \tau \tag{6}$$

where $\theta = [\theta_1, \theta_2]^{\mathrm{T}}$, $\tau = [\tau_1, \tau_2]^{\mathrm{T}}$. The system has two control inputs, the torques in the two joints, τ_1 and τ_2, and four measured outputs – the link angles, θ_1, θ_2, and their angular speeds $\dot{\theta}_1, \dot{\theta}_2$.

The mass matrix $M(\theta)$, Coriolis and centrifugal forces matrix $C(\theta,\dot{\theta})$, and gravity vector $G(\theta)$, are:

$$M(\theta) = \begin{bmatrix} P_1 + P_2 + 2P_3\cos\theta_2 & P_2 + P_3\cos\theta_2 \\ P_2 + P_3\cos\theta_2 & P_2 \end{bmatrix} \tag{7}$$

$$C(\theta,\dot{\theta}) = \begin{bmatrix} b_1 - P_3\dot{\theta}_2\sin\theta_2 & -P_3(\dot{\theta}_1+\dot{\theta}_2)\sin\theta_2 \\ P_3\dot{\theta}_2\sin\theta_2 & b_2 \end{bmatrix} \tag{8}$$

$$G(\theta) = \begin{bmatrix} -g_1\sin\theta_1 - g_2\sin(\theta_1+\theta_2) \\ -g_2\sin(\theta_1+\theta_2) \end{bmatrix} \tag{9}$$

The meaning and values of the physical parameters of the system are given in Table I.

Using these, the rest of the parameters in (6) can be computed by:

$$\begin{aligned} P_1 &= m_1c_1^2 + m_2l_1^2 + I_1 \qquad & P_2 &= m_2c_2^2 + I_2 \\ P_3 &= m_2l_1c_2 & & \\ g_1 &= (m_1c_1 + m_2l_1)g \qquad & g_2 &= m_2c_2g \end{aligned} \tag{10}$$

In the sequel, it is assumed that the manipulator operates in a horizontal plane, leading to $G(\theta) = 0$. Furthermore, the following simplifications are adopted in (6):

1) Coriolis and centrifugal forces are neglected, leading to $C(\theta,\dot{\theta}) = \mathrm{diag}[b_1, b_2]$;
2) $\ddot{\theta}_1$ is neglected in the equation for $\ddot{\theta}_2$;
3) the friction in the second joint is neglected in the equation for $\ddot{\theta}_1$.

After these simplifications, the dynamics of the manipulator can be approximated by:

$$\begin{aligned} \ddot{\theta}_1 &= \frac{1}{P_2(P_1 + P_2 + 2P_3\cos\theta_2)} \cdot \\ &\qquad \left[P_2(\tau_1 - b_1\dot{\theta}_1) - (P_2 + P_3\cos\theta_2)\tau_2\right] \\ \ddot{\theta}_2 &= \frac{\tau_2}{P_2} - b_2\dot{\theta}_2 \end{aligned} \tag{11}$$

The complete process state is given by $x = [\theta^{\mathrm{T}}, \dot{\theta}^{\mathrm{T}}]^{\mathrm{T}}$. If centralized control is used, the command is $u = \tau$; for decentralized control with one agent controlling each joint motor, the agent commands are $u_1 = \tau_1$, $u_2 = \tau_2$.

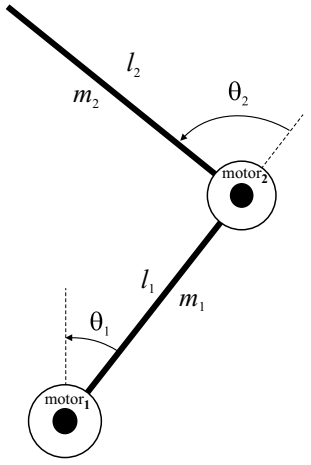


Fig. 1. Schematic drawing of the two-link rigid manipulator.

TABLE I
Physical Parameters of the Manipulator

Symbol	Parameter	Value
g	gravitational acceleration	9.81 m/s^2
l_1	length of first link	0.1 m
l_2	length of second link	0.1 m
m_1	mass of first link	1.25 kg
m_2	mass of second link	1 kg
I_1	inertia of first link	0.004 kgm^2
I_2	inertia of second link	0.003 kgm^2
c_1	center of mass of first link	0.05 m
c_2	center of mass of second link	0.05 m
b_1	damping in first joint	0.1 kgs^{-1}
b_2	damping in second joint	0.02 kgs^{-1}
$\tau_{1,\max}$	maximum torque of first joint motor	0.2 Nm
$\tau_{2,\max}$	maximum torque of first joint motor	0.1 Nm
$\dot{\theta}_{1,\max}$	maximum angular speed of first link	2π rad/sec
$\dot{\theta}_{2,\max}$	maximum angular speed of second link	2π rad/sec

B. RL Control

The control goal is the stabilization of the system around $\theta = \dot{\theta} = 0$ in minimum time, with a tolerance of $\pm 5 \cdot \pi/180$ rad for the angles, and ± 0.1 rad/sec for the angular speeds.

To apply RL in the form presented in Section II, the time axis, as well as the continuous state and action components of the manipulator, must first be discretized. Time is discretized with a sampling time of $T_s = 0.05$ sec; this gives the discrete system dynamics f. Each state component is quantized in fuzzy bins, and three torque values are considered for each joint: $-\tau_{i,\max}$ (maximal torque clockwise), 0, and $\tau_{i,\max}$ (maximal torque counter-clockwise).

One Q-value is stored for each combination of bin centers and torque values. The Q-values of continuous states are then interpolated between these center Q-values, using the degrees of membership to each fuzzy bin as interpolation weights. If e.g., the Q-function has the form $Q(\theta_2, \dot{\theta}_2, \tau_2)$, the Q-values of a continuous state $[\theta_{2,k}, \dot{\theta}_{2,k}]^{\mathrm{T}}$ are computed by:

$$\tilde{Q}(\theta_{2,k}, \dot{\theta}_{2,k}, \tau_2) = \sum_{\substack{m=1,\ldots,N_{\theta_2} \\ n=1,\ldots,N_{\dot{\theta}_2}}} \mu_{\theta_2,m}(\theta_{2,k}) \mu_{\dot{\theta}_2,n}(\dot{\theta}_{2,k}) \cdot Q(m, n, \tau_2), \ \forall \tau_2 \quad (12)$$

where e.g., $\mu_{\dot{\theta}_2,n}(\dot{\theta}_{2,k})$ is the membership degree of $\dot{\theta}_{2,k}$ in the n^{th} bin. For triangular membership functions, this can be computed as:

$$\mu_{\dot{\theta}_2,n}(\dot{\theta}_{2,k}) = \begin{cases} \max(0, \frac{c_{n+1} - \dot{\theta}_{2,k}}{c_{n+1} - c_n}), & \text{if } n = 1 \\ \max\left[0, \min(\frac{\dot{\theta}_{2,k} - c_{n-1}}{c_n - c_{n-1}}, \frac{c_{n+1} - \dot{\theta}_{2,k}}{c_{n+1} - c_n})\right], & \\ \qquad \text{if } 1 < n < N_{\dot{\theta}_2} & \\ \max(0, \frac{\dot{\theta}_{2,k} - c_{n-1}}{c_n - c_{n-1}}), & \text{if } n = N_{\dot{\theta}_2} \end{cases} \quad (13)$$

where c_n is the center of the n^{th} bin – see Fig. 2 for an example.

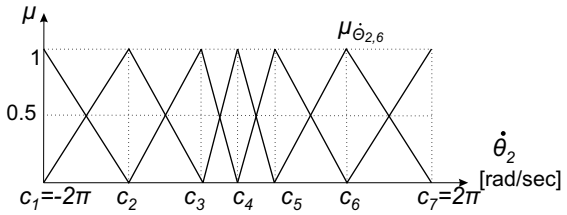


Fig. 2. Example of quantization in fuzzy bins with triangular membership functions for $\dot{\theta}_2$.

Such a set of bins is completely determined by a vector of bin center coordinates. For $\dot{\theta}_1$ and $\dot{\theta}_2$, 7 bins are used, with their centers at $[-360, -180, -30, 0, 30, 180, 360] \cdot \pi/180$ rad/sec. For θ_1 and θ_2, 12 bins are used, with their centers at $[-180, -130, -80, -30, -15, -5, 0, 5, 15, 30, 80, 130] \cdot \pi/180$ rad; there is no 'last' or 'first' bin, because the angles evolve on a circle manifold $[-\pi, \pi)$. The π point is identical to $-\pi$, so the 'last' bin is a neighbor of the 'first'.

Algorithm 1 Fuzzy value iteration for a SISO RL controller

$Q_0(m, u_j) = 0$, for $m = 1, \ldots, N_X$, $j = 1, \ldots, N_U$
$\ell = 0$
repeat
  for $m = 1, \ldots, N_X$, $j = 1, \ldots, N_U$ **do**
    $Q_{\ell+1}(m, u_j) = \rho(c_m, u_j) + \gamma \sum_{\tilde{m}=1}^{N_X} \mu_{x,\tilde{m}}(f(c_m, u_j)) \max_{\tilde{u_j}} Q_\ell(\tilde{m}, \tilde{u_j})$
  end for
  $\ell = \ell + 1$
until $\|Q_\ell - Q_{\ell-1}\| \leq \delta$

The optimal Q-functions for both the centralized and decentralized case are computed with a version of value iteration (3) which is altered to accommodate the fuzzy representation of the state. The complete algorithm is given in Alg. 1. For easier readability, the RL controller is assumed single-input single-output, but the extension to multiple states and / or outputs is straightforward. The discount factor is set to $\gamma = 0.98$, and the threshold value to $\delta = 0.01$.

The control action in state x_k is computed as follows (assuming as above a SISO controller):

$$u_k = h(x_k) = \sum_{m=1}^{N_X} \mu_{x,m}(x_k) \arg\max_{\tilde{u_j}} Q(\tilde{m}, \tilde{u_j}) \quad (14)$$

Centralized RL. The reward function ρ for the centralized learner computes rewards by:

$$r_k = \begin{cases} 0 & \text{if } |\theta_{i,k}| \leq 5 \cdot \pi/180 \text{ rad} \\ & \quad \text{and } \left|\dot{\theta}_{i,k}\right| \leq 0.1 \text{ rad/sec}, i \in \{1, 2\} \\ -0.5 & \text{otherwise} \end{cases} \quad (15)$$

The centralized policy for solving the two-link manipulator task must be of the form:

$$[\tau_1, \tau_2]^{\mathrm{T}} = h(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) \quad (16)$$

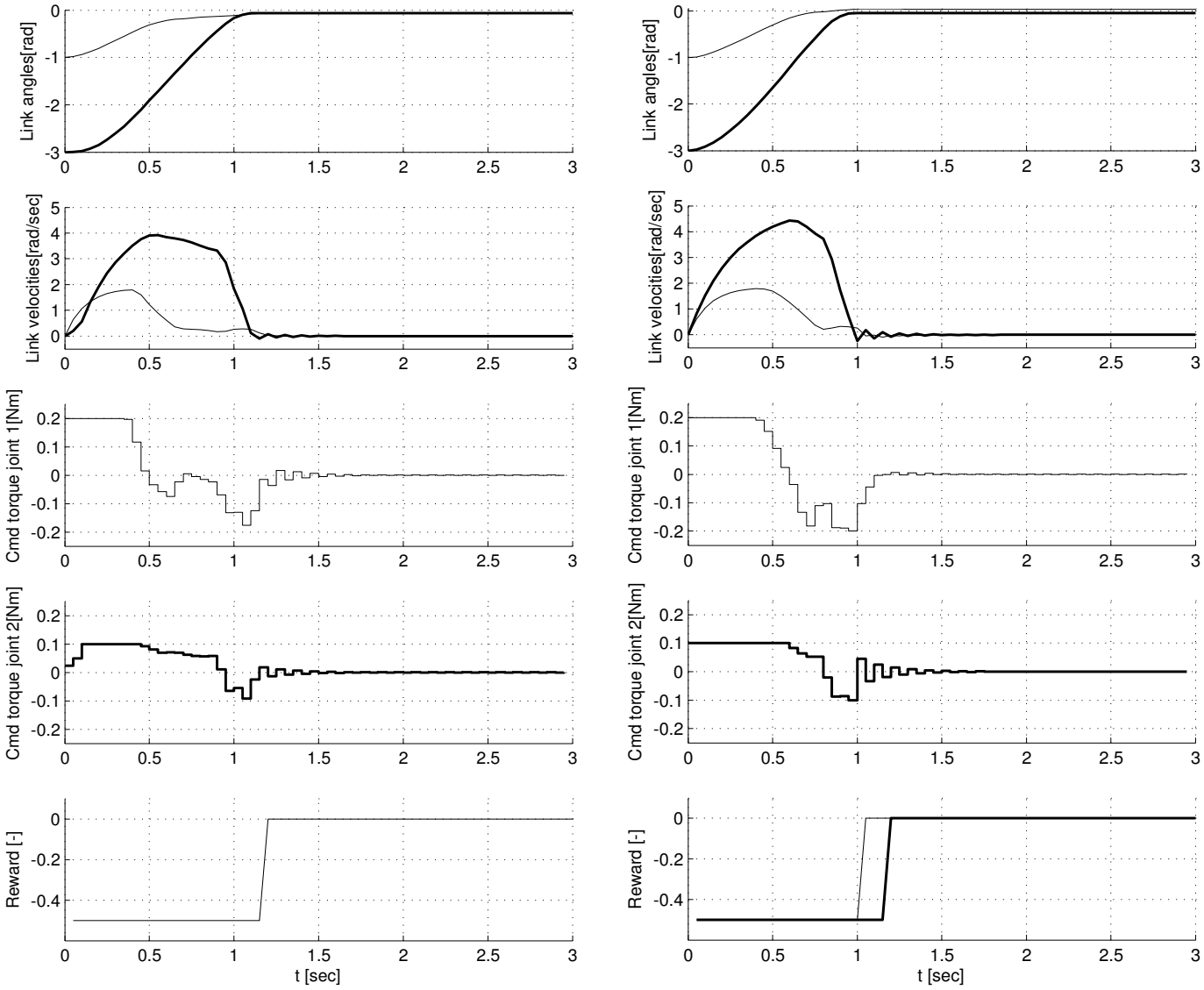
Therefore, the centralized learner uses a Q-table of the form $Q(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \tau_1, \tau_2)$.

The policy computed by value iteration is applied to the system starting from the initial state $x_0 = [-1, -3, 0, 0]^{\mathrm{T}}$. The resulting command, state, and reward signals are given in Fig. 3(a).

Decentralized RL. In the decentralized case, the rewards are computed separately for the two agents:

$$r_{i,k} = \begin{cases} 0 & \text{if } |\theta_{i,k}| \leq 5 \cdot \pi/180 \text{ rad} \\ & \quad \text{and } \left|\dot{\theta}_{i,k}\right| \leq 0.1 \text{ rad/sec} \\ -0.5 & \text{otherwise} \end{cases} \quad (17)$$

For decentralized control, the system (11) creates an asymmetric setting. Agent 2 can choose its action $\tau_{2,k}$ by only considering the second link's state, whereas agent 1 needs to take into account $\theta_{2,k}$ and $\tau_{2,k}$ besides the first link's state. If agent 2 is always the first to choose its action, and agent 1



(a) Centralized RL (thin line–link 1, thick line–link 2)

(b) Decentralized RL (thin line–link / agent 1, thick line–link / agent 2)

Fig. 3. State, command, and reward signals for RL control.

can learn about this action before it is actually taken (e.g., by communication) then the two agents can learn control policies of the following form:

$$\begin{aligned} \tau_2 &= h_2(\theta_2, \dot{\theta}_2) \\ \tau_1 &= h_1(\theta_1, \theta_2, \dot{\theta}_1, \tau_2) \end{aligned} \tag{18}$$

Therefore, the two agents use Q-tables of the form $Q_2(\theta_2, \dot{\theta}_2, \tau_2)$, and respectively $Q_1(\theta_1, \theta_2, \dot{\theta}_1, \tau_2, \tau_1)$. Value iteration is applied first for agent 2, and the resulting policy is used in value iteration for agent 1.

The policies computed in this way are applied to the system starting from the initial state $x_0 = [-1, -3, 0, 0]^{\mathrm{T}}$. The resulting command, state, and reward signals are given in Fig. 3(b).

C. Discussion

Value iteration converges in 125 iterations for the centralized case, 192 iterations for agent 1, and 49 iterations for agent 2. The **learning speeds** are therefore comparable for centralized and decentralized learning in this application. Agent 2 of course converges relatively faster, as it state-action space is much smaller.

Both the centralized and the decentralized policies stabilize the system in 1.2 seconds. The steady-state angle offsets are all within the imposed 5 degrees tolerance bound. Notice that in Fig. 3(b), the first link is stabilized slightly faster than in Fig. 3(a), where both links are stabilized at around the same time. This is because decentralized learners are rewarded separately (17), and have an incentive to stabilize their respective links faster.

The form of **coordination** used by the two agents is

TABLE II

COMPUTATIONAL REQUIREMENTS

	Q-table size	CPU time [sec]
Centralized	$(12 \times 7 \times 3)^2 = 63504$	≈ 18300
Agent 1	$12 \times 7 \times 12 \times 3 \times 3 = 9072$	≈ 1.7
Agent 2	$12 \times 7 \times 3 = 252$	≈ 1000
Agent 1 + Agent 2	9324	≈ 1000

indirect. The second agent can safely ignore the first (18). The first agent includes θ_2 and τ_2 in its state signal, and in this fashion accounts for the second agent's influence on its task. This is visible in Fig. 3(b) around $t = 0.8$s, when the first link is pushed counterclockwise ('up') due to the negative acceleration in link 2. Agent 1 counters this effect by accelerating clockwise ('down'). A similar effect is visible around $t = 1$s in Fig. 3(a).

The **memory and processing time requirements**[1] of value iteration for the two learning experiments are summarized in Table II. Both memory and CPU requirements are more than an order of magnitude higher for the centralized case. This is mainly because, as discussed in Section III, in the decentralized case the two agents were able to disregard state components that were not essential in solving their task, and thus reduce the size of their search space.

V. CONCLUSION AND FUTURE RESEARCH

We have pointed out the differences between centralized and multi-agent cooperative RL, and we have illustrated these differences on an example involving learning control of a two-link robotic manipulator. The decentralized solution was able to achieve good performance while using significantly less computational resources than centralized learning.

As can be seen in Table II, the memory (column 2) and time complexity (column 3) of the solutions scale poorly with the problem size. The multi-agent RL literature has not yet focused on the problem of scalability, although solutions for the centralized case exist (based mainly on generalization using function approximation to learn the value function). Such solutions might be extended to the decentralized case.

Another issue is that RL updates assume perfect knowledge of the task model (for model-based learning, e.g., value iteration (3)), or perfect measurements of the state (for online, model-free learning, e.g., Q-learning (4)). Such knowledge is often not available in real life. Studying the robustness of solutions with respect to imperfect models or imperfect observations is topic for future research.

ACKNOWLEDGEMENT

This research is financially supported by Senter, Ministry of Economic Affairs of the Netherlands within the BSIK-ICIS project "Interactive Collaborative Information Systems" (grant no. BSIK03024).

[1]The CPU times were recorded on a Centrino Dual Core 1.83 GHz machine with 1GB of RAM. Value iteration was run on Matlab 7.1 under Windows XP.

REFERENCES

[1] N. Vlassis, "A concise introduction to multiagent systems and distributed AI," University of Amsterdam, The Netherlands, Tech. Rep., September 2003, URL: http://www.science.uva.nl/~vlassis/cimasdai/cimasdai.pdf.

[2] P. Stone and M. Veloso, "Multiagent systems: A survey from the machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[3] M. J. Matarić, "Learning in multi-robot systems," in *Adaptation and Learning in Multi–Agent Systems*, G. Weiß and S. Sen, Eds. Springer Verlag, 1996, pp. 152–163.

[4] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine Learning*, vol. 33, no. 2–3, pp. 235–262, 1998.

[5] A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive load balancing: A study in multi-agent learning," *Journal of Artificial Intelligence Research*, vol. 2, pp. 475–500, 1995.

[6] S. Sen and G. Weiss, "Learning in multiagent systems," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. MIT Press, 1999, ch. 6, pp. 259–298.

[7] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, November 2005.

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

[9] G. Chalkiadakis, "Multiagent reinforcement learning: Stochastic games with multiple learning players," Dept. of Computer Science, University of Toronto, Canada, Tech. Rep., 25 March 2003, URL: http://www.cs.toronto.edu/~gehalk/DepthReport/DepthReport.ps.

[10] K. Tuyls and A. Nowé, "Evolutionary game theory and multi-agent reinforcement learning," *The Knowledge Engineering Review*, vol. 20, no. 1, pp. 63–90, 2005.

[11] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

[12] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[13] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: A critical survey," Computer Science Dept., Stanford University, California, US, Tech. Rep., 16 May 2003.

[14] R. Powers and Y. Shoham, "New criteria and a new algorithm for learning in multi-agent systems," in *Advances in Neural Information Processing Systems 17 (NIPS-04)*, Vancouver, Canada, 2004, pp. 1089–1096.

[15] M. Bowling, "Convergence and no-regret in multiagent learning," in *Advances in Neural Information Processing Systems 17 (NIPS-04)*, Vancouver, Canada, 13–18 December 2004, pp. 209–216.

[16] M. L. Littman, "Value-function reinforcement learning in Markov games," *Journal of Cognitive Systems Research*, vol. 2, pp. 55–66, 2001.

[17] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proc. 15th National Conference on Artificial Intelligence and 10th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-98)*, Madison, US, 26–30 July 1998, pp. 746–752.

[18] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Proc. 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02)*, Menlo Park, US, 28 July – 1 August 2002, pp. 326–331.

[19] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proc. Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, De Zeeuwse Stromen, The Netherlands, 17–20 March 1996, pp. 195–210.

[20] M. T. J. Spaan, N. Vlassis, and F. C. A. Groen, "High level coordination of agents based on multiagent Markov decision processes with roles," in *Workshop on Cooperative Robotics, 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02)*, Lausanne, Switzerland, 1 October 2002, pp. 66–73.

[21] C. Guestrin, M. G. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *Proc. Nineteenth International Conference on Machine Learning (ICML-02)*, Sydney, Australia, 8–12 July 2002, pp. 227–234.

[22] L. Buşoniu, B. De Schutter, and R. Babuška, "Multiagent reinforcement learning with adaptive state focus," in *Proc. 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC-05)*, Brussels, Belgium, October 17–18 2005, pp. 35–42.