Robotics and Autonomous Systems 58 (2010) 1306–1315

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Simultaneous learning of perception and action in mobile robots

Pablo Quintía [a], Roberto Iglesias [a,*], Miguel A. Rodríguez [a], Carlos V. Regueiro [b]

[a] Department of Electronics and Computer Science, Universidade de Santiago de Compostela, Campus Sur, Santiago de Compostela, Spain

[b] Department of Electronics and Systems, Universidade da Coruña, Fac. Informática, Campus del Elviña, A Coruña, Spain

ARTICLE INFO

Article history:
Available online 21 September 2010

Keywords:
Reinforcement learning
State representation
Fuzzy ART
Robot learning

ABSTRACT

One of the main problems of robots is the lack of adaptability and the need for adjustment every time the robot changes its working place. To solve this, we propose a learning approach for mobile robots using a reinforcement-based strategy and a dynamic sensor-state mapping. This strategy, practically parameterless, minimises the adjustments needed when the robot operates in a different environment or performs a different task.

Our system will simultaneously learn the state space and the action to execute on each state. The learning algorithm will attempt to maximise the *time before a robot failure* in order to obtain a control policy suited to the desired behaviour, thus providing a more interpretable learning process. The state representation will be created dynamically, starting with an empty state space and adding new states as the robot finds new situations that has not seen before. A dynamic creation of the state representation will avoid the classic, error-prone and cyclic process of designing and testing an ad hoc representation. We performed an exhaustive study of our approach, comparing it with other classic strategies. Unexpectedly, learning both perception and action does not increase the learning time.

1. Introduction

One of the key objectives in modern robotics is to have robots integrated into common life, take them out of the laboratories and put them to work in real environments while interacting with people. In order to have productive, safe and robust working robots, we need them to be able to cope with the dynamic nature of real environments: like humans or animals, robots should be able to adapt and learn from their own experiences instead of relying on predefined rules, models or hardwire controllers. The behaviour of a mobile robot is the result of the properties of the robot itself (physical aspects), the environment, and the control program the robot is executing. The same robot, running the same control programs, might act differently, simply because the wall covering has changed; hence, the performance and reliability of a robot controller operating continuously cannot be guaranteed, showing that *adaptability* is the main characteristic robots should possess. Traditional robot control code represents sensor-action couplings as rigid, usually pre-defined mappings that lack robustness in cases where novel situations occur, or where the robot control code does not cover the particular situation the robot is encountering.

This paper presents a system aimed at permitting robots to simultaneously learn how to perceive and how to act from their interaction with the environment (Fig. 1). The system we propose consists of two modules: one responsible for the *perception*, and the other for the *action*. The robot will cluster robot sensor readings into a finite number of distinguishable situations. We shall call these distinguishable situations *states*. Therefore, the *perception learning* module in Fig. 1 will achieve a sensor-state mapping that will dynamically increase to include new situations, detected in the stream of sensor inputs, and that have not been seen before. This dynamic sensor-state mapping will be combined with a learning strategy (action learning in Fig. 1) capable of adapting the robot's behaviour to those relevant situations (states) that are being identified by the perception module.

We already tried to achieve this same goal in the past [1]: nevertheless, we ended up with a system which, although promising, involved too many parameters and slow learning. We then investigated alternatives to speed up reinforcement learning [2,3], so that now we can propose an innovative alternative where the number of parameters is reduced to the bare minimum, and the learning times are very short.

2. Action learning

One way of getting a robot to learn from its interaction with the environment is through reinforcement: according to psychology theories, learning is strengthened if it is followed by positive reinforcement—pleasure—and weakened if it is followed by punishment—pain [4]. This is something that is clearly described

* Corresponding author.
E-mail addresses: pablo.quintia@usc.es (P. Quintía), roberto.iglesias.rodriguez@usc.es (R. Iglesias).

0921-8890/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2010.08.009

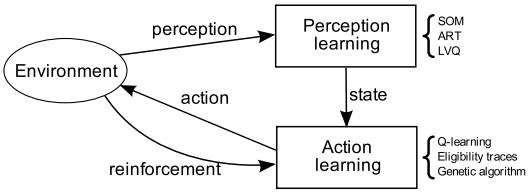

Fig. 1. General schema of our proposal.

in Thorndike's *Law of Effect* [5]: actions followed by good or bad outcomes have their tendency to be reselected altered accordingly. Modern reinforcement learning grew inspired by these psychological theories (trial and error learning), together with optimal control theory and its solution using value functions [6–8], and the development of temporal difference methods [9–12]. Reinforcement learning is a machine learning paradigm that determines how an agent ought to take actions in an environment so as to maximise some notion of long-term reward:

$$E\left[\sum_{t=0}^{\infty}\gamma^t r_t\right], \tag{1}$$

r_t is the reinforcement received at time t, and $\gamma \in [0, 1]$ is a discount factor which adjusts the relative significance of long-term rewards versus short-term ones.

Therefore, when reinforcement learning is applied in robotics, all a robot needs is a reinforcement function which tells the robot how well or how poorly it has performed, but nothing about the set of actions it should have carried out. Through a stochastic exploration of the environment, the robot must find a control policy which maximises the expected total reinforcement described above (Eq. (1)).

Reinforcement learning is learning what to do, i.e. how to map situations to actions. In general, reinforcement learning algorithms attempt to improve the agent's decision-making policy over time. Formally, a policy is a mapping from relevant and distinguishable states to actions. Any of the most common reinforcement learning algorithms can be used in the system we propose (Fig. 1); hence, we include a brief description of some of them in this paper and we test them in the experimental application of our system (experimental results described in Section 5). Nevertheless, since we wish to minimise the number of parameters that need to be determined by the user, we shall propose our own algorithm (*increasing time before failure*). This algorithm will be described later in this paper (Section 2.3).

2.1. Q-learning

Q-learning [12] is one of the most popular reinforcement learning algorithms, through which the robot will learn a utility function of states and actions, termed Q-function. This function estimates the expected discounted reward for every state-action pair, i.e. $Q(s, a)$ is the expected sum of future rewards obtained by taking action a in state s and following an optimal policy thereafter,

$$Q(s, a) = E\left[r(s, a) + \sum_{t=1}^{\infty}\gamma^t r_t | s_0 = s, a_0 = a\right] \tag{2}$$

where $r(s, a)$ is the reinforcement the robot receives when it executes action a in state s, and $E\left[\sum_{t=1}^{\infty}\gamma^t r_t | s_0 = s, a_0 = a\right]$ represents the expected total reinforcement the robot will receive starting from the state s, by the executing action a, and then following the best possible policy.

Once these Q-values have been learnt, the optimal action for any state is the one with the highest Q-value; this is called the greedy policy.

During the learning process, the system updates the estimation of how good or bad the execution of action a_t in the state s_t seems to be:

$$\Delta Q(s_t, a_t) = \beta_L(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)), \tag{3}$$

where $\beta_L \in [0, 1]$ is a learning rate. Eq. (3) describes what is called the temporal difference error (TD-error), i.e., the difference between the expected consequences of executing a_t in s_t and what is really observed. This is also the simplest version of Q-learning, as it only updates Q-values based on the transition between two consecutive states. Hence, this algorithm is considered as one of the 1-step temporal difference methods [13,14]. Nevertheless, when rewards occur infrequently, it can take many learning trials for these values to propagate to previous states. Accordingly, multi-step TD-learning [13] attempts to expedite this process by simply providing the system with additional memory. One of the most common and well-known multi-step TD methods consists in the use of *eligibility traces* [15].

2.2. Q-learning improved with eligibility traces

An eligibility trace [15] is a temporary record of the occurrence of an event, such as the visiting of a state or the performance of an action. An eligibility trace is a variable associated with each pair (state, action). The eligibility trace for the state s, action a, at time t is denoted as $e_t(s, a) \in \mathbb{R}^+$. This value represents the time elapsed since the last time the robot visited state s and performed action a. Eligibility traces are a basic mechanism for temporal credit assignment: when a TD error occurs (Eq. (3)), only the eligible states or actions are assigned credit or blame for the error.

Three different methods have been proposed for combining eligibility traces and Q-learning: Watkins $Q(\lambda)$, Naive $Q(\lambda)$, and *Peng* $Q(\lambda)$ [12,13]. We have considered the use of the first two of the aforementioned methods since they are the most commonly used.

In Watkins $Q(\lambda)$, the Q-values corresponding to the states that have been visited by the robot not long before (eligible states) will be updated according to the current TD-error (Eq. (3)). Nevertheless, this backpropagation of the consequences of the execution of a_t in s_t will not take place if the robot performs an exploratory action, i.e., the robot performs an action a_t whose Q-value is not the maximum one for the current state, s_t. Owing to this, the trace update is carried out in two stages: first, the traces for all state–action pairs are either decayed by $\gamma\lambda$ or, if an exploratory action was taken, set to 0. Second, the trace corresponding to the current state and action is incremented by 1.

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ & \text{and } a_t = \arg\max_a(Q(s_t, a)) \\ 0 & \text{if } s = s_t \text{ and } a = a_t \\ & \text{and } a_t \neq \arg\max_a(Q(s_t, a)) \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise} \end{cases} \tag{4}$$

$\lambda \in [0, 1]$ is the trace-decay parameter. This trace will be used to backpropagate the TD error:

$$Q(s_t, a_t) = Q(s_t, a_t) + \beta_L\delta e_t(s, a), \tag{5}$$

where

$$\delta = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t).$$

Naive $Q(\lambda)$ is like Watkins $Q(\lambda)$, except for the fact that the traces are not set to zero on exploratory actions:

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{if s = } s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise.} \end{cases} \tag{6}$$

The update equation of the Q-values is the same as for Watkins $Q(\lambda)$ (Eq. (5)). It is important to notice the number of parameters that are present in both learning algorithms: γ, λ, and β_L.