ELSEVIER

Robotics and Autonomous Systems 37 (2001) 53–68

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Symbolic structures in numeric reinforcement for learning optimum robot trajectory☆

Getachew Hailu

Electrical and Computer Engineering, Villanova University, 800 Lancaser Avenue, Villanova, PA 19085, USA

Received 12 April 2001

Communicated by F.C.A. Groen

Abstract

This article contributes to the understanding of how symbols can profitably be integrated into a numeric reinforcement learning approach so as to realize a learning robot in a continuous problem setting. An initial experiment clearly revealed that the underlying task could not be learned merely on the basis of numeric reinforcement values. The unstructured utilization of sensory values of the reinforcement approach is dealt with by incorporating readily available domain knowledge. The domain knowledge pre-labels the task space into different regions and builds a connection between the direct sensory experience and the high level features. Teams of networks, each dedicated to a particular feature, exploit the differential characteristics of the regions to rapidly learn the sub-tasks. In addition, a pre-wired fuzzy controller whose function is to restrict the set of actions from which the learner composes a control policy is integrated into the learner. The task faced by the robot has interesting characteristics that are unknown initially, and must be learned. Experimental results indicate that after the robot has been placed in the task space and given the externally motivated needs, it has been able to learn, with a handful of trials, the right policy that fulfills the external needs. © 2001 Published by Elsevier Science B.V.

Keywords: Domain knowledge; Numeric reinforcement; Radial basis functions; Fuzzy behavior

1. Introduction

An intelligent agent presented with a learning task has two sources of information available: the instantaneous training instant drawn from experience, and a prior knowledge about the task that may have been derived from earlier experience. These two sources of information are compatible and complementary. Prior knowledge guides and organizes the process of learning from new experience and new experience is the basis for incremental addition to knowledge.

Depending on how they make use of prior knowledge, learning methods fall anywhere in the continuum from unbiased to highly biased, see [5,14,34]. Since reinforcement learning is *numeric* (i.e., works with unstructured raw sensory data), and *inductive* (i.e., learns and generalizes from experience alone), it is near the unbiased end. Consequently, it requires a large number of training episodes that lead to impractical memory requirement. The problem is not just the memory space, but more importantly the *time* needed to fill the memory with data and to subsequently learn from the data.

Several shortcomings that hinder the practicability of reinforcement learning have been reported in the literature. For example, reinforcement learning fails to

☆ This work is a part of the author's dissertation work that is done at the University of Kiel, Germany.

0921-8890/01/$ – see front matter © 2001 Published by Elsevier Science B.V.
PII: S0921-8890(01)00142-7

recognize and identify new situations quickly [11,31], cannot pass back rewards to other states, except to states of the current path [4], is unable to cope with large states and actions space [9,35], does not know how to maintain a balance between exploration and exploitation [23,50], is unable to attach suitable reactions to incoming new situations [37], etc. For few of them, solutions have been suggested; but to a greater degree, the reason that hinders its applicability is hinged on the *reward feedback*, which gives little guidance for feature extraction or action selection.

For example, when a reinforcement learning system fails by choosing the wrong action, the reward feedback does not specify which of the output nodes were wrong. If the system has a hidden layer of feature detectors, acting properly will depend on both identifying the current context, as well as selecting an action appropriate to the context. The feedback signal does not indicate which of these processes is at fault: it does not distinguish between the case where the system rightly identified its context but selects the wrong response, and the case where the system learned responses were correct, but its feature detectors misidentified the context. In network terminology, the system needs to know whether it should tune its feature detector, or the weights placed on the outputs of those feature detector, or both.

This article advocates the need for bottom-up information or some type of domain knowledge to supplement the top-down reward feedback so as to realize a reinforcement-based learning on physical robots. It integrates two kinds of biasing schemes into the learning network. The first bias is aimed at overcoming the inability of reinforcement learning to identify outstanding features at once in the task space. In many practical problems, there is hardly a genuine lack of knowledge, domain knowledge after all is available. Therefore, the scheme utilizes this available knowledge to construct a heuristic that aids the learner to detect and classify key features [19,30,31]. In this article, a feature is a subset of input space that is considered to be the same for the purpose of learning and generating action. Under this bias, learning takes place by a team of networks [21,22,52], each specializing on individual features. The second bias is a reflex, introduced and applied by Millán [37], that restricts the action search space. While the primary advantage of restricting the search space is to focus exploration where it is mostly needed, its added advantage is to eliminate undesirable actions from consideration and make the robot functional at the early stage of its learning process [37].

We begin the article by describing the task, the robot is facing. Subsequent to that, the robot sensory system and the environment in which the robot operates are presented. We then go on by defining the constituents (inputs and outputs) of the learning network, and the reinforcement signal used to teach the network. After presenting the two built-in knowledge bases in detail, the learning architecture and the adaptation algorithms are overviewed. Afterwards experimental results obtained by the path taken from an early development in the simulation to the actual testing on a physical robot are presented. Finally, the work of this article and the result obtained are compared to relevant reinforcement works done in the past.

2. Problem statement

Reinforcement learning has been used to solve the shortest path problem in maze-like structures, for example [6,34,47]. Furthermore, algorithms have been developed to extend it to non-maze-like structures [39,41]. These algorithms find solution paths without ever attempting to optimize the path. In this article, the interest is still searching for a path in a non-maze-like structure, but with additional constraint of optimizing the path.

Formally, the problem consists of two Cartesian coordinate systems specifying the start and the target locations of the robot. The task faced by the robot is to build a self-adaptive controller that searches for a trajectory, which (when followed by the robot) would lead to a minimum cost. We define the cost of a path $\mathcal{P}$ as, $C(\mathcal{P}) = C_{\mathrm{D}} + C_{\mathrm{S}}$. The first term represents the cost associated with the path duration and the second term is the cost associated with its safety. Thus, among all the feasible paths $\mathcal{Q}$ that lead to the target, the problem searches for the minimum cost path $\mathcal{P}$ that is both safe and short, i.e., $\mathcal{P} = \arg\min_{\mathcal{L}\in\mathcal{Q}} C(\mathcal{L})$.

3. The experimental system

In this section, both the robot and the environment in which the learning experiment is carried out are presented.

3.1. The robot

The B21 robot from the Real World Interface (RWI), Fig. 1, is used as our experimental platform. The robot is a cylindrical four-wheeled synchronous drive with two parts: a base and an enclosure. The base carries 32 infrared (IR) and 32 tactile sensors, whereas the enclosure has a belt of 24 tactile, 24 IR, and 24 sonar sensors, each placed evenly around the robot's periphery. A two finger manipulator with six DOF and a binocular CCD camera are mounted on the top of the enclosure. In addition, it is equipped with an encoder that provides the position of the robot with an accuracy of 0.254 cm, though the actual accuracy is dependent on the slippage between the robot's wheels and the floor.

The 24 sonar sensors define the robot's view of the environment and form a part of the input space of the learning system. The IR and tactile sensors are used in emergency to detect real or virtual collisions. Whereas a real collision is detected by the tactile sensors, a virtual collision is detected by the IR sensors, see Section 5 for detail.

3.2. The environment

Fig. 2 shows the top view of the robot's environment consisting of an indoor space of 25 m^2 and a corridor of width 1.8 m. The origin of the robot is the black dot inside the room and the big circle in the corridor is the goal location. As mentioned in Section 2, the task is to take the robot from the origin, $\boldsymbol{p}_r(t = 0)$ to the goal location, $\boldsymbol{p}_g$. Both the robot and the goal positions are specified in a Cartesian coordinates system, and the coordinate of the center of the goal relative to the origin is

$$\boldsymbol{p}_g = \begin{pmatrix} x_g \\ y_g \end{pmatrix} = \begin{pmatrix} 4.00\,\text{m} \\ -2.00\,\text{m} \end{pmatrix} \quad (1)$$

At first glance, both the task and the environment seem relatively simple compared to what we would like our robots be able to do. However, when one tries to implement it on present day robots, it becomes clear that this seemingly simple task is no longer easy. First and foremost, the task is performed using *local* sensory information; the robot has neither access to the global view of the environment nor a comprehensive world model. This is controversial; there is no universal consensus if biological systems also learn from local sensory information alone. But this will not hinder us; as Landelius [27] put it: in the same way it is possible to build machines that fly but do not flutter their wings, machine learning is aimed at designing machines that show intelligent behavior but lack the full perceived process found in biological systems. Second, it is a high-dimensional continuous learning task and successful goal reaching requires a non-linear mapping from this space to the space of continuous real-valued actions. In

Fig. 1. The experimental robot equipped with sonar, IR, tactile sensors, and an odometry system. The six DOF manipulator and the binocular CCD camera are reserved for visual processing.

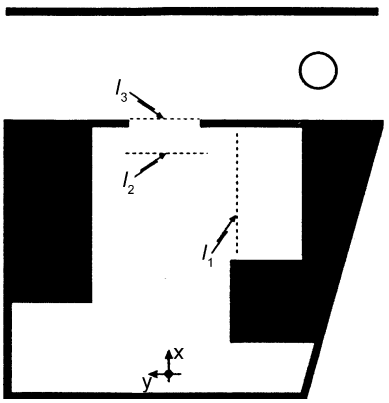

Fig. 2. Top view of the robot world. The three lines l_1, l_2, and l_3 encode domain-specific knowledge, see Section 6.1.

general, it is not easy to train a network on large spaces.

4. Inputs and outputs

The learning system builds its input from the external as well as the internal sensors of the robot. Among the three types of external sensors used, only the sonar sensors form a part of the input space, i.e., the controller learns an action map from the sonar sensors. It also uses the IR and tactile sensors, but solely in emergency conditions that require a fixed and prior mapping.

Before the sonar sensors are fed to the controller, their values are normalized so that each falls in the interval [0, 1], i.e., $s_i \in [0, 1], \forall i \in \{0, \ldots, 23\}$. Apart from this normalization process, no attempt is made either to process or collapse the sensory data as done in [18]. The main reason for retaining the whole sensory data in the input is to capture as much information about the environment as possible so that the problem of hidden states can be minimized [54].

In addition to the sonar data, the controller gets the relative distance of the robot from the goal, i.e., $\|\boldsymbol{p}_{\mathrm{r}}(t) - \boldsymbol{p}_{\mathrm{g}}\|$. In order to make this scalar distance value comparable to the dimension of the sonars, Millán's codification scheme [37] is applied to the scalar relative distance. The scheme involves eight localized processing units, whose activation values depend on how far the normalized relative distance is away from the respective center positions of the Gaussian processing units, i.e., $\phi_i = \exp(-(\mu_i - \rho)^2)$. Here the $\mu_i, i \in \{0, \ldots, 7\}$, are the pre-assigned center positions of the units, which are placed evenly along the abscissa spanning the interval [0, 1], and ρ is the normalized relative distance between the robot and the goal. Hence, it is these values, $\phi_i \, \forall i \in \{0, \ldots, 7\}$ forming a vector $\bar{\rho}$, that are used as part of the input space of the controller. Thus, the overall input vector to the controller is a vector of 32 continuous value elements,

$$x = \begin{pmatrix} \boldsymbol{s} \\ \bar{\rho} \end{pmatrix} \tag{2}$$

The B21 robot has multiple motor parameters that one can use to efficiently control its motion. But learning multi-motor parameter is extremely difficult. First, the reward does not tell us which of the motor parameters are at fault. Second, if common internal representations (such as common neurons or weight vectors) are used, adapting the internal representations for one of the parameter often have a damaging consequence on the values of the other parameters. Therefore, it is more effective to restrict the repertoire of motor parameters. Consequently, the robot's angular rotation, which determines its next direction, is the only motor parameter chosen to control the robot. However, for every rotation, two motor actions are initiated; first the robot completely rotates by the specified angle, after rotation has ceased it will move to a new location by translating forward a fixed distance, $l = 20\,\mathrm{cm}$.

5. Reinforcement functions

The controller receives an immediate reinforcement (carved by and passed from the environment) every time the robot executes an action. This does not mean that it is an immediate reinforcement learning task. Rather, the controller computes and optimizes the long term desirability of states after taking into account the states that are likely to follow and the immediate reinforcements available in those states.

The reinforcement function has two components. The first component penalizes the robot whenever it collides with or approaches an obstacle. To detect these events, two flags: `collision` and `close` are defined. The collision flag is set either when the body of the robot is in contact with obstacles (real collision) or when the reflectance of any of the IR sensors [1] is greater than 70 (virtual collision). As pointed out in [1,37], the virtual collision is included to make the learning process safe. When collision occurs, the controller receives a fixed penalty and reverse the robot to clear the collision. The close flag is set when any of the sonars measures a distance less than a given specified value, $d_{\mathrm{c}} = 30\,\mathrm{cm}$. In this case, the generated penalty increases as the distance between the robot and the obstacle decreases. Altogether, the component of the reinforcement that teaches the robot to avoid obstacles

[1] The IR sensors detect objects that are within the range of 0.5 m by emitting light and measuring the intensity of the reflection bounced from the objects. The reflectance values range from 0 to 100 and are highly dependent on the color of the object.

has the form

$$f_1 = \begin{cases} -3 & \text{if collision} \\ -1 + \min_j \left(\dfrac{d_j}{d_c}\right) & \text{if } \min_j d_j < d_c \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The other component teaches the robot how to approach the goal point by first computing the acute angle between the robot heading θ_h, and the line connecting the goal and the robot location θ_{gr}. This angle measures the divergence of the robot from the goal — if the angle is increasing then the robot is moving away from the goal. Hence, the second component of the reward function is proportional to this angle and is normalized to lie between $[-1, 0]$, i.e.,

$$f_2 = -\frac{\text{acute}(|\theta_h - \theta_{gr}|)}{\pi} \quad (4)$$

The total immediate reinforcement is the sum of Eqs. (3) and (4). This reinforcement function does not teach the robot directly how to reach the goal; it only trains the robot how to approach the goal without collision. Approaching and reaching are quite different; the robot can approach a goal without ever reaching the goal (e.g., if the goal is enclosed). Therefore, the above reinforcement function presupposes that the environment satisfies the constraint that it has at least one free way or path through which the robot can reach the goal without collision.

5.1. Inconsistent reinforcement

Properties of the physical hardware of a robot, which are often constrained by various sensory, mechanical, and computational limitations, impose restrictions not only on the control strategies that can be applied but also on the type of tasks and experiments that can be used. One of these hardware limitations is the robot's dead reckoning system. Most mobile robot controllers rely on reasonably accurate dead reckoning for localization or spatial learning. However, overtime slippage between the robot's wheels and the floor results in errors, both in the position and orientation of the robot. The robot's rotational error tends to be more serious than the translational error, since small errors in rotation lead to large errors in translation at a location far away from the origin of the coordinate frame.

In this article, the robot position is used to decode the relative distance between the robot and the goal and to provide implicitly (by computing the goal angle) a part of the reinforcement function, Eq. (4). From the two, the latter one is more sensitive to the inaccuracy of the robot position, because it leads to an inconsistent reinforcement that makes learning difficult or even impossible. [2] But, in order to guarantee that the measured position is close to the true robot position, we have exploited the only crucial property of the robot's dead reckoning system. Dead reckoning performs satisfactorily provided that the robot does not move for an extended period of time without reaching the goal. This characteristic has directly restricted how far and how hidden the goal should be placed away from the origin.

6. Built-in knowledge

This section is devoted to the two forms of built-in knowledge used to realize reinforcement learning on our robot.

6.1. Environment model

Much work has been performed with discrete state-space particularly in an MDP domain known as grid world, see example [6,47]. Most useful learning applications, however, take place in multi-dimensional continuous state space. The obvious way of transforming such a state space into a discrete problem involves a uniform quantization. But it is rarely the case that the entire space requires a fixed quantization; since there are significant sub-spaces of the state space that are either unimportant or for which the optimal response is the same throughout. On the other hand, it is often the case that some critical areas require high resolution. Hence, it would be inefficient to represent all the space at high level of resolution. Different methods have been tried to improve the representation [26,31,35,37,41,48].

[2] Noting this [38] has eliminated the dependence of the reinforcement value on the odometry reading by building *goal sensors* that are capable of detecting the goal explicitly.

Millán's method [37] uses an on-line adaptive state construction algorithm within the reinforcement learning. His construction algorithm apportions states not only according to the perceived distribution of the input but also according to the *variability* of the target function (Section 8.1). For the algorithm to work without falling pray to the curse of dimensionality, he has exploited the unique feature of the Nomad 200 robot: the turret motor. Since the turret motor orients the sensors independent of the robot heading, sensor readings were made independent of the robot heading, i.e., $\boldsymbol{x} = \boldsymbol{s}(\boldsymbol{p}_{\mathrm{r}}(t), \theta_{\mathrm{r}}) = \boldsymbol{s}(\boldsymbol{p}_{\mathrm{r}}(t))$. In this case, the state space has been contained, and appropriate states have been constructed directly from the raw sensory data.

Unfortunately, these types of robots are the exception rather than the rule; most robots do not align their sensors independent of the base. Consequently, the perceived sensory data would be different every time a robot visits a given location at different headings, i.e., $\boldsymbol{x} = \boldsymbol{s}(\boldsymbol{p}_{\mathrm{r}}(t), \theta_{\mathrm{r}}(t))$. Therefore, in the absence of prior knowledge, the adaptive state construction algorithm either requires a training period that would challenge the most patient teacher or fails to cope with the states and identify key structures. The only way that we can give leverage to the algorithm is by introducing some domain-specific knowledge that determines how much the robot knows about the different parts of its environment. Hence, prior to learning, non-overlapping features, whose union covers the all state space, were extracted by cutting the world at its joints. Choosing features appropriate to the task is an important way of adding domain knowledge to a learning system [24]. Earlier works of this kind where the task space is partitioned a priori to enforce specific goal sequence or behavior selection include [11,31,44].

Our chosen features are: `concave`, `door`, `corridor`, and `room`, which are delineated by the lines l_1: $y = -1.0\,\mathrm{m}$, l_2: $x = 3.50\,\mathrm{m}$, and l_3: $x = 3.75\,\mathrm{m}$ (see Fig. 2). The features correspond to the natural constituents (components) of the task space along which generalization is likely. Based on the x and y intercepts of the lines and the robot position $\boldsymbol{p}_{\mathrm{r}}(t)$, a set of heuristic rules are written. The rules serve as an interface between the low level sensory signals and the high level cognitive knowledge by singling out a unique feature where the robot is currently found.

In the heuristic below, l_{1y} is the y intercept of the line l_1, l_{2x} and l_{3x} are the x intercepts of lines l_2 and l_3, respectively, see Fig. 2

$$\texttt{corridor} = l_{3x} < p_x(t)$$
$$\texttt{concave} = p_x(t) < l_{3x} \&\& p_y(t) < l_{1y}$$
$$\texttt{room} = p_x(t) < l_{2x} \&\& \texttt{!concave}$$
$$\texttt{door=!corridor\&\&!concave \&\&!room}$$

Heuristic signal-to-symbol mapper.

The heuristic bias decomposes the global controller into four separate and non-interacting learning components. In lieu of tackling the problem by a monolithic network that poorly covers everything, the problem is now shared by a team of networks, each dedicated to a particular region. Once the heuristic identifies a unique symbol, it completely cuts off the sensory input of all the networks, except the one chosen. In so doing, it carves up the state space into mutually exclusive and exhaustive regions, and each network learns the action map of the subset of the input variables relevant to its specific region. The architecture of the controller looks much like the *mixture of experts* [22,52] with the heuristic bias replacing the gated network. The essential difference is while in the former case, the partition of the output is due to the result of partitioning the input, in the latter case it is due to the weighted average of the output of each network. The bias has also the advantage of minimizing state aliasing. Since each network stores its own state history information; a single state *can* infer two or more different actions from different networks without causing any ambiguity.

Although we are aware that such a bias is generally ad hoc, causes sub-optimality in performance, and trades with autonomy, all these drawbacks are out-weighted by the benefit it brings to the learning system. Further, we argue that the provision of this domain-specific knowledge is not a large sacrifice to the robot autonomy; because this is only a coarse partition that is not adequate to learn the task. Other aspects of the task faced by the robot are still to be learned inductively and numerically by splitting the initial partitions.

6.2. Fuzzy behaviors as reflex

The other stumbling block that limits the practical applicability of reinforcement learning is that the robot does not know how to act when it gets into a new situation. Often, it either collides with obstacles that terminate the learning process or wanders aimlessly without ever reaching the goal. Observing this, Millán [36,37] has introduced and applied a new prior knowledge called *reflex*, which not only ameliorates the problem but also brutally cuts down the learning trials. Unlike the earlier bias, the reflex is independent of a particular environment, and therefore does not have to be re-wired in a new environment. Subsequently, Hailu and Sommer have applied the reflex both on a simulated [18] and a physical [17] robot. The reflex actions, though eventually overridden by more accurate learned actions, keep the agent safe and direct it in the right direction while it is trying to learn. The added advantage of the reflex bias is its silency; it intervenes only when the learning system needs help [20].

Our reflex consists of two fuzzy behaviors; while the first one is a reactive `obstacle avoidance`, the other one is a purposive `goal following`. The behaviors are implemented by a set of fuzzy rules that have fuzzy sets in the antecedent and conclusion parts [32,45,58]. Since the outputs of the behaviors are combined one to one, the number of output fuzzy sets of each behavior and the form of their membership functions are identical; but their input fuzzy sets and fuzzy variables are different. The output fuzzy sets that decode the robot heading are `left`, `forward`, and `right`, and have overlapping triangular membership functions spanning the interval $[-\pi, \pi]$. The `obstacle avoidance` behavior receives the sonar data as input variables and outputs a three-dimensional vector $\boldsymbol{\alpha}_a$, whose elements indicate the activation levels of the output fuzzy sets. Likewise, the `goal following` behavior inputs the acute angle between the robot heading and the vector connecting the current robot and goal locations and outputs a similar three-dimensional vector $\boldsymbol{\alpha}_g$. Note that, since this behavior seeks a particular goal that cannot be sensed by the robot's perceptual sensors, it utilizes the robot's internal representation to indirectly sense the goal.

The outputs of the behaviors are fused using Payton et al. [42] architecture of combining outputs of multiple behaviors (Fig. 3). Since it is our desire that the reflex points always to the direction of the goal, this scheme of combining outputs guarantees that the final command is goal directed. The other approach, the command arbitrator scheme, where a single behavior is chosen based on behavior priorities, *may* not always be goal directed, since information regarding, say goal following, would not be available once the command arbitrator selects the collision avoidance behavior.

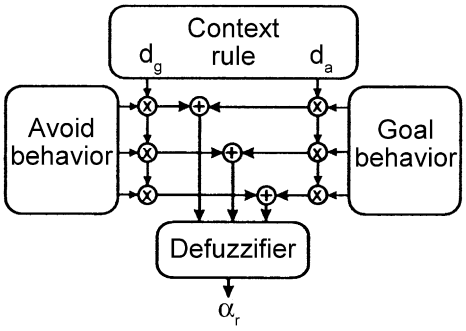

Fig. 3. The reflex bias realized as two fuzzy behaviors.

The outputs of the behaviors are combined by assigning different desirability functions for each behaviors [42,43,57]. Generally, desirability functions are complex and vary with the context. However, since the reflex is used to provide only an initial search location, it suffices to consider a less rigorous blending scheme, where the behaviors, irrespective of their context, have fixed desirability functions. Following this, two constant desirability values $\boldsymbol{d} = (d_a, d_g)$, one for each behavior, are chosen. Fusion is done by combining the activation strengths of the corresponding nodes of the behaviors with their respective desirability values, i.e.,

$$\boldsymbol{\alpha}_f = \boldsymbol{d}\begin{pmatrix}\boldsymbol{\alpha}_a \\ \boldsymbol{\alpha}_g\end{pmatrix} = d_a\boldsymbol{\alpha}_a + d_g\boldsymbol{\alpha}_g, \tag{5}$$

where $\boldsymbol{\alpha}_f$ is the fused vector. Following fusion is the defuzzification process that decodes the fused vector to a crisp equivalent $\boldsymbol{\alpha}_r$.

7. Learning architecture

In continuous space reinforcement learning, most states encountered will never have been experienced exactly before. The only way to learn and successfully

cope with such a space is when the system has the ability to generalize from previously experienced states to the one that has never been seen before. One simple way to transform such space is to quantize the space into a multi-dimensional uniform grid and treat each cell within the grid as an atomic state. Although this can be effective in certain problem domains, such as [7], a uniform grid approach generally leads to unmanageable states [9,12].

The other method is the use of a global approximator such as a multi-layer sigmoidal neural network. This method has booked successful results [3,16,28,49] in approximating the value function in a variety of tasks. But there is no reason to believe that such networks are well suited to reinforcement learning. First, they tend to forget episodes unless they are frequently retrained for those episodes. Second, the need to make small gradient descent steps makes learning very slow, particularly in the early stages.

7.1. Localized AHC architecture

Instead of a global approximator, the other alternative is a sparse and coarse-coded local approximator known as radial basis functions (RBFs) network [40]. Although most works of RBF network are either supervised (e.g., estimating regression functions from noisy data [29,51]) or unsupervised (e.g., building feature maps [10,25,33]), it has recently been applied by Millán [37] in the framework of reinforcement learning to locally generalize states and through them state values. His architecture, called TESEO, operates similar to the way the global adaptive heuristic critic [53] operates; namely, the critic network guides how the action network is to be adapted. Architecturally, however, there is a subtle difference between them. In the latter case, two distinct networks are adapted simultaneously; whereas in the former, the actor and critic networks are lumped together and only a single network is adjusted.

Here TESEO's architecture is employed; however, as a direct consequence of the a priori partitioning of the task space, four separate and non-interacting function approximating networks are employed. Fig. 4 shows one of the four functions approximating network working in tandem with a neural "reinforcement-learning" unit, which is shared by all the approximating networks. The approximating network

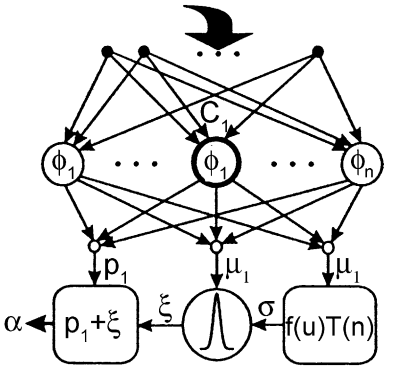

Fig. 4. A localized AHC architecture. A state generalization input layer and a "reinforcement-learning" output layer working in tandem.

(upper part of Fig. 4) is a single layer RBF network where the input neurons are fully connected to the continuous input vector, $\boldsymbol{x}$, through the excitatory connection vector, $\boldsymbol{c}$. The network generalizes states and through them state values by covering the continuous space with finite and overlapping receptive fields to produce features representations for the final mapping where reinforcement learning takes place.

The strength of the activation value of the basis function is determined from the distance between the sensory input and the excitatory connection vectors, i.e., $\phi_j(\boldsymbol{x}) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_j\|^2/\varsigma^2)$ where $\boldsymbol{c}_j$ is the excitatory connection vector of the jth RBF neuron. While TESEO adapts the width of receptive fields of each RBF neurons (provided that their widths do not fall below a fixed minimum, at which point adaptation ceases), here the receptive widths of the neurons are kept fixed to ς, i.e., $\varsigma = \sigma_j = 0.1\ \forall j$. The rationale behind this is varying both parameters ($\boldsymbol{c}_j$ and σ_j) upsets the initial formed cluster (input coverage) and subsequently the state values.[3] Therefore, in order to avoid this "high" variation in cluster formation, the receptive widths are kept fixed and only the excitatory connection weights are variable.

Apart from the parameters $\boldsymbol{c}_j$, which hold the location (in the continuous space) of the feature, additional three adjustable parameters are associated to each feature state of Fig. 4. Earlier it has been mentioned that the approximating network gradually partitions

[3] It is an experimental observation.

the high-dimensional real-valued state into finite discrete features space. Hence, much in the spirit of *Q*-learning that defines action and utility values for discrete states, the first two parameters represent the action p_j and utility u_j values. The other parameter μ_j controls the mean location where the output layer performs exploration when this particular feature is chosen.

8. Network growth, exploration, and adaptation algorithms

In this section, we will be discussing: the mechanisms of growing the network, the strategy of exploration, and the algorithms that adapt the network parameters. Since the mechanisms and algorithms are based on the known methods, our discussion is qualitative, highlighting the main aspects only. Details can be found in [15,37,46,55].

8.1. Network growth

Initially, the function approximating network is empty (i.e., no situation is encoded) but gradually grows like the works of Refs. [2,10,26,37] as it starts partitioning the sub-region. When a new situation arrives from the robot's sensory module, existing neurons (if any) compete to win the situation. The winning neuron is the one with the highest activation value or with the closest distance to the situation, i.e., $i = \arg\max_j \phi_j(\boldsymbol{x}) = \arg\min_j (\boldsymbol{x} - \boldsymbol{c}_j)^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{c}_j)$. If the distance between the winning neuron and the situation is larger than the width of the receptors, the situation can not be generalized. In this case, a new neuron is introduced into the network with its center position equals to the current perceived input. This way of growing the network is called *distance driven*.

The other method of growing the network involves the mechanism introduced and applied by Millán [37] to identify states that deliver inconsistence reinforcements. The idea is, if the TD error is larger than a specified threshold, then the past situation is wrongly classified. Because, even if the situation is close to the past winning neuron, after action is taken it is discovered that the estimated utility is quite different from the stored utility — as attested by a large TD error. Therefore, the controller "undone" the previous association by creating a new neuron at the location of the past perceptual input. In this way, sensory states that initially look similar and categorized in the same feature, will gradually be split according to their consequences. This is the other method of growing the network and is called *error driven*.

8.2. Utility-based exploration

As indicated earlier, the lower part of Fig. 4 is the output layer that is shared by the four approximating networks to explore and generate a continuous action. The layer has a "reinforcement-learning" unit developed by Gullapalli [15] for use in continuous action space. It is a stochastic unit that draws actions from a Gauss distribution and adjusts the parameters based on the experience. The distribution parameters (μ and σ) that control the Gauss process are derived from the learning parameters of the winning neuron of the approximating network (Fig. 4) by

$$\mu = \mu_i, \qquad \sigma = f(u_i)T(n) = \frac{T(n)}{1 + \exp(u_i)}, \tag{6}$$

where $T(n)$ is a computational temperature that decreases with trial n. While the mean, as mentioned earlier, determines the location *where* the unit explores, the variance controls its exploratory behavior.

We have related the extent of exploration to the utility value through a sigmoid function. If the utility is small, meaning the chosen action is not performing well, the variance becomes high, resulting in exploration of the range of choice. On the other hand, when an action performs well (meaning the utility is high) the mean moves in that direction and the variance decreases, resulting in a tendency to generate action values near the successful one. Once the parameters are computed, similar to [37] a random value is drawn from the distribution that modulates the action p_i, and generates the final stochastic action $\alpha = p_i + \xi$. The computational temperature is intended to control the necessary trade-off between exploration and control. It progressively narrows the extent of exploration (σ of Eq. (6)) around the mean, and gradually produces a deterministic action by controlling the strength of modulation. Both these are achieved by cooling

the temperature after each trial[4] until a pre-selected minimum value is reached.

8.3. Parameters adaptation

The learning parameters are defined when a new situation is discovered and an RBF neuron (representing the situation) is created.

The four adjustable parameters (defined for each feature) are first initialized as follows: the center position $\boldsymbol{c}_j$ is equated to the current perceived situation, the prototypical action p_j is set to the action received from the reflex component, the utility u_j is estimated by computing the reinforcement function for the current robot states and sonar readings, and the mean μ_j is set to zero. These initial values are subsequently adapted during the course of learning through the reinforcement algorithm.

The basic error source used to adapt the parameters is the temporary difference [46] between the actual and the expected utility values. Since adaptation takes place after the receipt of a reinforcement value, the controller adapts the parameters of the past winning neuron, before it generates action for the present sensation. The utility value of the winning neuron $u_j(t)$ is updated by the TD method. Williams' REINFORCE algorithm [55] is employed to adapt the center of exploration μ_j. Depending on the sign of the TD error of the winning neuron, its center position $\boldsymbol{c}_j$ is either shifted toward the previous sensation or left unchanged and, action p_j is replaced by a more accurate learned action when the robot reaches the goal through a trajectory whose total payoff is greater than the maximum payoff so far obtained.

9. Experimental results

In this section, results obtained by the path taken from an early development in the simulation to the actual testing on the real robot is presented. The controller described so far is the one that was implemented on the physical environment. Prior to its implementation, however, another controller that worked on a simulated robot was developed [18]. Despite the fact that some parts of the simulated controller were carried over to the real world, the difference between the simulated and the real world has necessitated modifications to be made before transferring it to the physical robot.

First, the controller of the simulator had only the reflex bias (Section 6.2) and key states were disambiguated by the state construction algorithm alone. Nevertheless, when this controller was used on the physical robot, it was no longer possible to quickly split key world states. Consequently, the controller had been modified to accommodate additional bias that substantially altered the architecture. Second, the input data of the simulator was highly pre-processed by hashing the sonars into regions and throwing all the sonars in the region, except the one that read the minimum. The rationale behind this technique was to keep the dimension of the input space low. The method when applied to the robot, however, has caused frequent occurrences of a state aliasing (hidden state) problem that is difficult to deal with. Consequently, instead of hashing the processed sonars, the unprocessed raw sensory data is used on the real robot (Section 4). Third, as soon as a real or virtual collision occurred, the agent was immediately terminated and a new trial was initiated by placing the robot back to its home location. Although this scheme worked well on the simulation, it was inefficient in terms of time when applied to the real robot. Therefore, instead of terminating the agent and aborting the trial altogether, an emergency handler was incorporated into the controller that rescues the robot and lets the robot resume its trial.

9.1. Simulation results

9.1.1. A note on the simulation

At the time of simulation, a TRC robot was the only platform available in our laboratory, and the simulator was built keeping this robot in mind as the target robot. However, after the simulation was completed, we acquired a B21 robot that has far better sensor-motor characteristics and development software. To exploit the advantages of the new platform, instead of working on the TRC, for which the simulator was built, we began working on the B21. Therefore, while the simulation results were obtained from the TRC

[4] A trial is defined as a time interval that begins as soon as the robot starts moving from the origin toward the goal and stops immediately after the robot reaches the goal.

simulator, the real results have been obtained from the B21 robot.

At this point, we want to stress that the reason why a major architectural adjustment was needed when transferring to the physical robot was not due to the change in the platform. As indicated in [13], the motivation for using simulation is not to finish the design stage there and transfer the components directly to the physical robot. Rather, it is to allow us to come up with a working (often coarse, however) version of the controller. Hence, architectural corrections were inevitable even if the platform had not been changed, see [56] for an example.

9.1.2. Results

Fig. 5 (top and bottom-left) shows the ghost paths of the robot at different sampled trials of its learning phase. A ghost is placed when the controller toggles

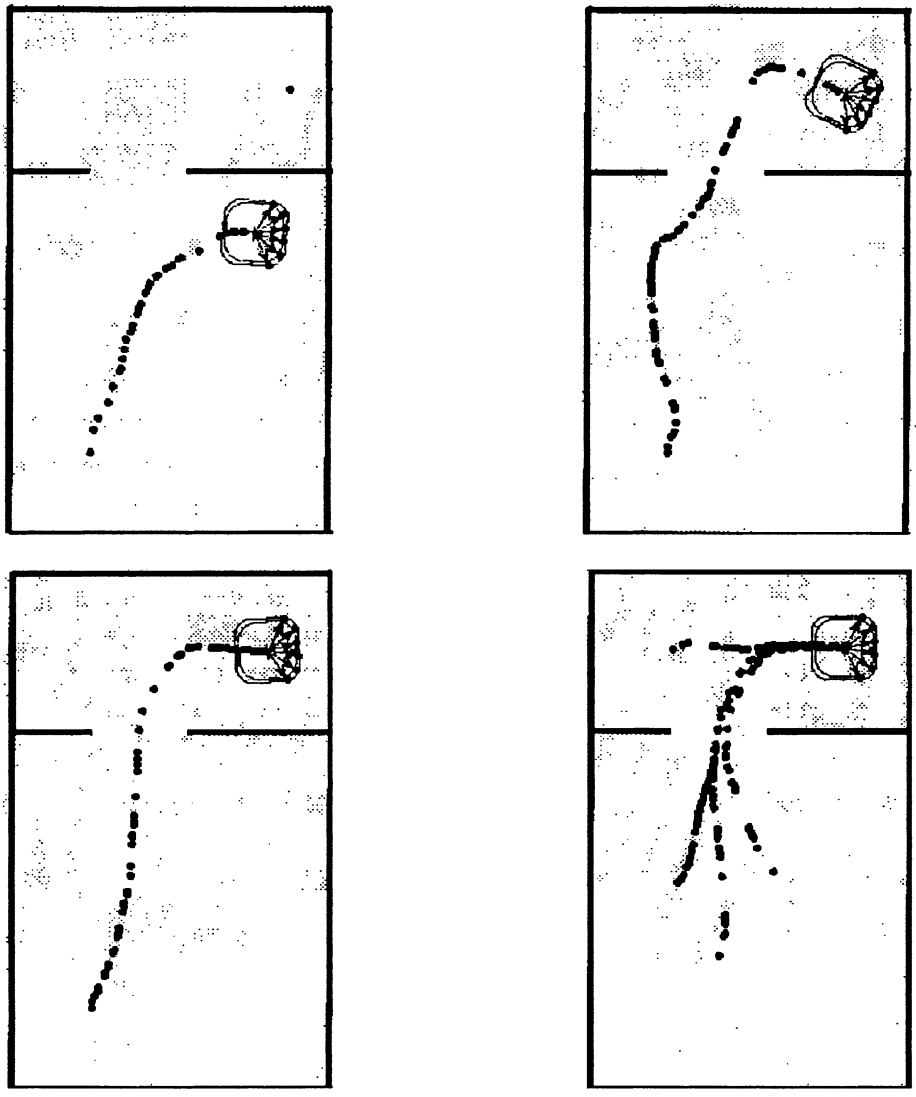

Fig. 5. Trajectories of the simulated robot at the first trial (top-left), the first time the robot arrives at the goal (top-right), and at the final trial (below-left). The behavior of the final controller for four sampled starting locations (below-right).

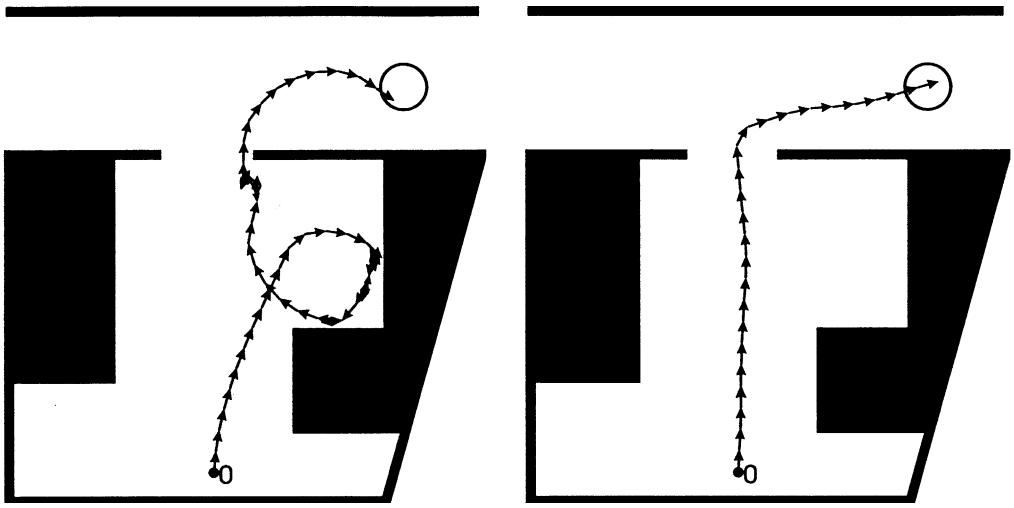

Fig. 6. Trajectories taken by the robot during the first (left) and final (right) trials.

neurons, so as to represent pictorially where along the route neurons are concentrated. During the initial few trials, it was observed that the robot failed to reach the target; this is attributed to the initial state of the controller, which is empty and has to encode enough situation–action pairs. At trial nine, the robot reached the goal for the first time. But the path followed, Fig. 5 (top-right), does not look like a planned path; it is more of a haphazard motion. The plot of Fig. 5 (below-left) shows the final trajectory obtained after the robot has made 50 trials. Comparing this final path with that of Fig. 5 (top-right) it can been seen that the robot had rapidly adapted the coarse and instinct skill acquired from the reflex component to get a smooth, short (10% shorter than Fig. 5, top-right) and planned-like path.

Also, the behavior of the final controller was tested to see if it could still take the robot to the target when the starting location of the robot is altered. Fig. 5 (below-right) shows trajectories obtained from four different starting locations. As can be seen, the controller is able to produce feasible trajectories even when the robot is started from other locations for which it is not trained. But we observed that this behavior is true for restricted regions; if the chosen starting location is within the region that the robot had already explored during the learning phase. Outside of this region, the controller fails. This is because situation–action pairs are decoded locally. As a consequence, if the robot is placed at a location that has not been explored before, either no neuron would be active or, even worse, the learned action of the active neuron is quite different from the one needed in that location.

9.2. Real experiments

We will now present results obtained from the physical robot. As pointed out at the beginning of this section, the results were obtained after the necessary

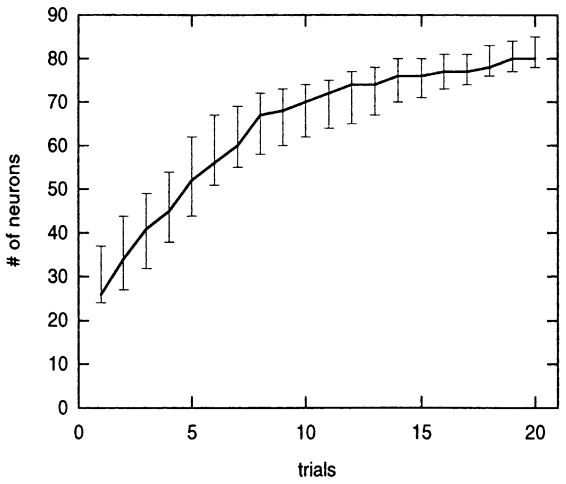

Fig. 7. Number of neurons present in the network at the end of each trials.

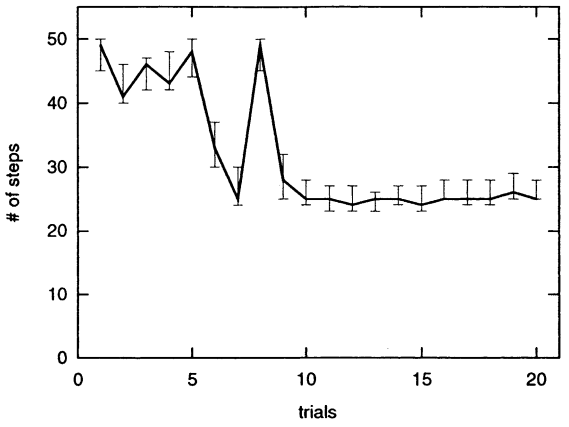

Fig. 8. Number of steps the robot has required to reach the target.

adjustments on the controller of the simulator had been made.

Fig. 6 shows sample robot trajectories and Figs. 7–9 are plots of the learning curves. During the initial trial, the controller produced a trajectory that was no better than what the reflex would have produced, if it had been directly connected to the robot. Furthermore, it was during the first few trials the robot incurred a high payoff (Fig. 9) and a sharp growth in the network size was registered (Fig. 7). Both of these signify that the robot was exploring the environment during its initial learning phase. As the trials proceeded, however, the robot gradually started to unfold its path. In addition, the size of the network has begun to increase at a much

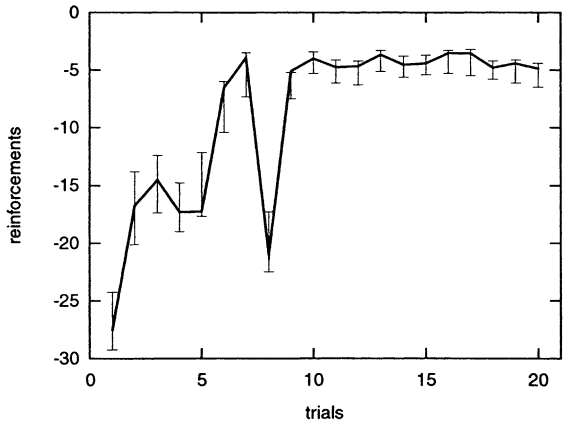

Fig. 9. The total payoff (cost), the robot received at each trial.

Table 1
Final network performance

Quantities	Mean	Variance
Number of neurons	82.5	4.7286
Number of steps	27.7	1.9000
Total reinforcements	−6.24	0.8752

reduced slope than the earlier trials. By the sixth trial and afterwards the robot had practically straightened its path, except in the eighth trial where it took a different action and depart from the already learned path. In the following trials, however, the robot has returned to its previous performance and followed the same path, without any significant divergence through out the remaining trials. A similar phenomena was observed in the [37] work, too. Comparing the initial and final robot trajectories (Fig. 6), we observe that the robot learned three useful abilities: (1) to short circuit the concave region that causes the robot to fold its path, (2) to avoid colliding with the door edges by passing through the middle of the door, and (3) to head-on directly to the goal after it had passed the door.

The repeatability of the controller was also tested by conducting 10 sets of experiments, each consisting of 20 trials. On the learning curves of Figs. 7–9, the vertical error bars indicate the minimum and maximum variations of the network performance values at each trial. Generally, as the trial continues, the variation of the values tends to decrease. Table 1 characterizes these variations by computing the mean and the variance of the network performance values at the last trial in the set of 10 experiments.

10. Related work

Mahadevan and Connell [31] employed *Q*-learning to learn a global box pushing task. They employed a prior domain-specific knowledge to learn the task. The difference from the work of this article is in the way the prior knowledge was used. While in Mahadevan and Connell it is used to define the applicability conditions, in this work it is used to directly decompose the task space. But these are different sides of the same coin — defining the applicability conditions indirectly entails task decomposition. The other difference is that while they used a statistical

clustering technique for state representation and learned *Q*-values, here an RBF network was used for state representation and state values were learned.

Matarić [35] applied implicit domain knowledge to facilitate the learning process in a multi-agent environment. The domain knowledge was primarily utilized to shape the reward function and to transform states and actions to conditions and behaviors. Conditions and behaviors, however, are more suitable to subsumption like architecture [8] and are not directly applicable to connectionist architecture that works very close to the raw sensory data.

Millán [37] realized reinforcement learning on a Nomad 200 mobile robot for a similar goal reaching task. Our work is close to his, but differs in the following: The major difference, as mentioned in Section 6.2, is the physical capability of the robot. While the Nomad robot has a turret motor that enables it to focus its sensors on the goal, the B21 robot lacks this ability; consequently it has to deal with the resulting large space. To cope with this, symbolic knowledge about the environment, absent in the work of Millán [37], is integrated into the controller. Though the method works satisfactorily, it does not "contain" the state space as the turret did in [37]. The net effect is that in the last few trials the size of the network did not abate to grow (Fig. 7), and the variance values of the final network performance (Table 1) were a bit higher than that reported in [37].

11. Conclusions and remarks

The article has presented a physically based learning robot that is realized by endowing the robot with appropriate and sufficient prior knowledge. On the robot we worked, the use of domain-specific knowledge that pre-labels and decomposes the environment has been found necessary in order to ease the construction of the state space. The other built-in knowledge is a hard-wired controller that focuses exploration where it is mostly needed. With the help of these two biases, the robot has able to correct, after a handful of trials, the initial acquired actions, and learned the short and safe path to the goal by unfolding its original trajectory.

The paper has two main drawbacks or limitations. Both limitations are a direct consequence of the way the learning system uses the position information. The first limitation has already been discussed in Section 5.1. There, we have emphasized how the position information (due to inaccurate odometry) leads to inconsistent reinforcement and distorts the learning process.

Likewise, the domain specific bias presented in Section 6.1 uses the position information and is, therefore, subject to the same odometry error. That is, the heuristic before routing the sensor information compares the odometry reading with the stored data base to figure out which environment feature needs to be considered. Therefore, in the event of significant mismatch between the odometry reading and the true robot position, the heuristic may select the incorrect feature and subsequently channels the sensory data to the wrong controller — this too damages the overall learning process. The obvious method to eliminate this problem is to directly identify the key features, similar to [38], without knowing the robot position. One such mechanism, for instant, is to use unique beacons in each region, that are easily detectable and identifiable by robust sensors.

Experiments were conducted to determine if the final learned controller was still able to take the robot to the goal location, when the robot was started from different locations or when obstacles were placed along its path. In both cases, however, negative results are obtained that indicate the need for further investigation. The other point that needs improvement is the symbols extraction method. Though symbols extraction and modularization have remained to be a strong technique of biasing [24], to date they are entirely done on a problem basis (e.g., [31,44]). It would be helpful if some principled way of symbols extraction and modularization are investigated.

Acknowledgements

The author greatly acknowledges José R. Millán for the useful discussion and support he has obtained during the early stage of the work as well as for the encouragement he has received in the writing of this paper.

References

[1] I. Ahrns, Ultraschallbasierte navigation und adaptive hindernisvermeidung eines autonomen mobilen roboters, Master's

Thesis, Institut für Informatik und Praktische Mathematik, CAU zu Kiel, 1996.

[2] E. Alpaydin, Networks that grow when they learn and shrink when they forget, Technical Report 91-032, Computer Science Institute, Berkeley, CA, 1991.

[3] C.W. Anderson, Learning and problem solving with multilayer connectionist systems, Ph.D. Thesis, University of Massachusetts, Amherst, MA, 1986.

[4] M.R. Andrew, Using transitional proximity for faster reinforcement learning, in: Proceedings of the Ninth International Machine Learning Conference, Aberdeen, 1992, pp. 316–321.

[5] A.G. Barto, Artificial intelligence, neural network, and control, in: G.A. Bekey, K.Y. Goldberg (Eds.), Neural Networks in Robotics, Kluwer Academic Publishers, Dordrecht, 1992, pp. viii–xi.

[6] A.G. Barto, S.J. Bradtke, S.P. Singh, Learning to act using real time dynamic programming, Artificial Intelligence 72 (1) (1995) 81–138.

[7] A.G. Barto, R.S. Sutton, C.W. Anderson, Neurolike elements that can solve difficult learning problems, IEEE Transactions on Systems, Man, and Cybernatics 5 (13) (1983) 834–846.

[8] R.A. Brooks, A layered intelligent control system for mobile robot, IEEE Transactions on Robotics and Automation RA-2 (1986) 14–23.

[9] R.A. Brooks, M.J. Matarić, Real robots, real learning problems, in: Robot Learning, Kluwer Academic Publishers, Dordrecht, 1993, pp. 193–213.

[10] J. Bruske, G. Sommer, Dynamic cell structure learns perfectly topology preserving map, Neural Computation 7 (4) (1995) 834–846.

[11] D. Chapman, L.P. Kaelbling, Input generalization in delayed reinforcement learning: an algorithm and performance comparison, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, 1991.

[12] T. Dean, K. Basye, J. Shewchuk, Reinforcement learning for planning and control, in: S. Minton, Machine Learning Methods for Planning and Scheduling, Morgan Kaufmann, Los Altos, CA, 1992.

[13] M. Dorigo, Introduction to the special issue on learning autonomous robots, IEEE Transactions on Systems, Man, and Cybernetics 26 (3) (1996) 361–364.

[14] G. Drastal, Learning in abstraction process, in: S.J. Hanson, G.A. Drastal, R.L. Rivest (Eds.), Computational Learning Theory and Natural Learning Systems, Vol. I: Constraints and Prospects, MIT Press, Cambridge, MA, 1994, pp. 163–211.

[15] V. Gullapalli, A stochastic reinforcement learning algorithm for learning real valued function, Neural Networks 3 (1990) 671–692.

[16] V. Gullapalli, Skillful control under uncertainty via reinforcement learning, Robotics and Autonomous Systems 15 (1995) 237–246.

[17] G. Hailu, G. Sommer, Embedding knowledge in reinforcement learning, in: International Conference on Artificial Neural Networks, Skövde, Sweden, 1998, pp. 1133–1138.

[18] G. Hailu, G. Sommer, Learning by biasing, in: Proceedings of the IEEE Conference on Robotics and Automation, Leuven, Belgium, 1998, pp. 2168–2173.

[19] J. Hampshire, A. Waibel, The meta-pi network: Building distributed knowledge representation for robust pattern recognition, Technical Report CMU-CS-89-166, Carneige Mellon University, Pittsburgh, PA, 1989.

[20] D.A. Handelman, S.H. Lane, Fast sensorimotor skill acquisition based on rule-based training of neural networks, in: G.A. Bekey, K.Y. Goldberg (Eds.), Neural Networks in Robotics, pp. 255–270.

[21] R.A. Jacob, M.I. Jordan, Hierarchical mixtures of local experts and the EM algorithm, Neural Computation 6 (1994) 181–214.

[22] R.A. Jacob, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1991) 79–87.

[23] L.P. Kaelbling, Learning in Embedded Systems, MIT Press, Cambridge, MA, 1993.

[24] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: A survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.

[25] T. Kohonen, Self-organized formation of topologically correct feature maps, Biological Cybernetics 43 (1982) 59–69.

[26] B.J.A. Kröse, J.W.M. van Dam, Adaptive state space quantisation for reinforcement learning of collision-free navigation, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Raleigh, NC, 1992, pp. 1327–1331.

[27] T. Landelius, Reinforcement learning and distributed local model synthesis, Ph.D. Thesis, Department of Electrical Engineering, Linköping University, 1997.

[28] L. Lin, Reinforcement learning for robots using neural networks, Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1993.

[29] D. Lowe, Radial basis function networks, in: M.A. Arbib (Ed.), Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, 1995.

[30] P. Maes, R.A. Brooks, Learning to coordinate behaviors, in: Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), Boston, MA, 1990, pp. 796–802.

[31] S. Mahadevan, J. Connell, Automatic programming of behavior-based robots using reinforcement learning, Artificial Intelligence 55 (1992) 311–365.

[32] E.H. Mamdani, S. Assilian, An experiment in lingustic synthesis with a fuzzy logic controller, International Journal of Man–Machine Studies 7 (1975) 1–13.

[33] T. Martinetz, K. Schulten, Topology representing networks, Neural Network 7 (3) (1993) 507–522.

[34] M.J. Matarić, A comparative analysis of reinforcement learning methods, Technical Report, MIT Artificial Intelligent Laboratory, Cambridge, MA, 1991, p. 1322.

[35] M.J. Matarić, Reward functions for accelerated learning, in: W.W. Cohen, H. Hirsh (Eds.), Proceedings of the 11th International Conference on Machine Learning, New Brunswick, NJ, 1994.

[36] J.R. Millán, Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot, Robotics and Autonomous Systems 15 (1995) 275–299.
[37] J.R. Millán, Rapid, safe and incremental learning of navigation strategies, IEEE Transactions on Systems, Man, and Cybernetics 26 (3) (1996) 408–420.
[38] J.R. Millán, Incremental acquisition of local networks for the control of autonomous robots, in: Proceedings of the Seventh International Conference on Artificial Neural Networks, Lausanne, Switzerland, 1997, pp. 739–744.
[39] J.R. Millán, C. Torras, A reinforcement connectionist approach to robot path finding in a non-maze-like environment, Machine Learning 8 (3–4) (1992) 363–395.
[40] J.E. Moody, C.J. Darken, Fast learning in networks of locally tuned processing units, Neural Computation 2 (1) (1989) 281–294.
[41] A.W. Moore, Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces, in: Proceedings of the Eighth International Machine Workshop, Ithaca, NY, 1991.
[42] D.W. Payton, J.K. Rosenblatt, D.M. Keirsey, Plan guided reaction, IEEE Transaction on Systems, Man, and Cybernetics 20 (6) (1990) 1370–1382.
[43] A. Saffiotti, E.H. Ruspini, K. Konolige, Using fuzzy logic for mobile robot control, in: D. Dubois, H. Prade, H.J. Zimmermann (Eds.), Handbook of Fuzzy Sets and Possibility Theory, Kluwer Academic Publishers, Dordrecht, 1997.
[44] S.P. Singh, Scaling reinforcement learning algorithms by learning variable temporal resolution models, in: Machine Learning — Proceedings of the Ninth International Workshop (ML'92), Aberdeen, 1992, pp. 406–415.
[45] M. Sugeno, M. Nishida, Fuzzy control of a model car, Fuzzy Sets and Systems 16 (1985) 103–113.
[46] R.S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning 3 (1) (1988) 9–44.
[47] R.S. Sutton, First result with Dyna, an integrated architecture for learning, planning and reacting, in: W. Thomas, R.S. Sutton, P.J. Werbos (Eds.), Neural Networks for Control, MIT Press/Bradford Books, Cambridge, MA, 1992.
[48] R.S. Sutton, Generalization in reinforcement learning: successful examples using sparse coarse coding, in: Advances in Neural Information Processing Systems, 1996, pp. 1048–1044.
[49] G. Tesauro, TD-gammon, a self-teaching backgammon program, achieves master-level play, Neural Computations 6 (2) (1994) 215–219.
[50] S.B. Thrun, The role of exploration in learning control, in: Handbook of Intelligent Control Neural, Fuzzy and Adaptive Approaches, Van Nostrand Reinhold, New York, 1992.
[51] A.R. Webb, Functional approximation by feed-forward networks: a least-squares approach to generlization, IEEE Transactions on Neural Networks 5 (3) (1994) 363–371.
[52] A.S. Weigend, M. Mangeas, A.N. Srivastava, Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting, International Journal of Neural Systems 6 (4) (1995) 373–399.
[53] P.J. Werbos, A menu of design for reinforcement learning over time, in: W. Thomas, R.S. Sutton, P.J. Werbos (Eds.), Neural Networks for Control, MIT Press/Bradford Books, Cambridge, MA, 1992.
[54] S.D. Whitehead, D.H. Ballard, Learning to perceive and act by trial and error, Machine Learning 7 (1991) 45–83.
[55] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine Learning 8 (1992) 229–256.
[56] B. Yamauchi, R. Beer, Spatial learning for navigation in dynamic environments, IEEE Transactions on Systems, Man, and Cybernetics 26 (3) (1996) 496–504.
[57] J. Yen, N. Pfluger, A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation, IEEE Transactions on Systems, Man, and Cybernetics 25 (6) (1995) 971–977.
[58] L.A. Zadeh, A fuzzy-set-theoretic interpretation of lingustic hedges, Journal of Cybernetics 3 (2) (1972) 4–34.

Getachew Hailu received his Ph.D. (1999) in Computer Science from the University of Kiel, Germany. Prior to that, he was a Visiting Researcher (1994–1995) at Advanced Robotics Laboratory, Tokai Univeristy, Japan. From 1990 to 1993, he was a Junior Fellow at the International Center for Theoretical Physics (ICTP), Italy. Currently, Dr. Hailu holds a post-doctoral position at Villanova University, Villanova, PA, USA.