Appeared in:
IEEE International Conference on Robotics and Automation (ICRA'98)
Leuven, Belgium, May 16-21, 1998.

Learning by Biasing

G. Hailu, G. Sommer
Christian Albrechts University
Department of Cognitive Systems
Preusserstrasse 1-9, D-24105 Kiel, Germany

Abstract

In the quest for machines that are able to learn, reinforcement learning (RL) is found to be an appealing learning methodology. A known problem in this learning method, however, is that it takes too long before the robot learns to associate suitable situation - action pairs. Due to this problem, RL has remained applicable only to simple tasks and discrete environment. To accelerate the learning process to a level required by real robot tasks, the traditional learning architecture has to be modified. We propose a modified reinforcement based robot skill acquisition and adaptation architecture. The architecture has two components: a bias and a learning components. The bias component imparts to the learner coarse a priori knowledge about the task. Subsequently, the learner refines the acquired actions through reinforcement learning. We have validated the architecture and the learning algorithm on a simulated TRC mobile robot for a goal reaching task.

1 Introduction

Programming an autonomous robot to reliably carry out its task demands a complete knowledge of the task and the environment. Systems designed with complete knowledge are called *expert systems* and have no learning ability. Instead they are equipped with a large amount of data base that requires careful tuning. However, because of the complexity and uncertainty of the real world, it is prohibitive to create an expert system with large data base. Besides, it is argued that if the robot some how possesses a *self-learning* ability, an enormous amount of human effort would be saved from tuning the data base.

In the past many machine learning techniques have been proposed. Most of the learning techniques assume the presence of teacher provided training instances in the form of stimuli and desired response. These types of learning techniques are known as *supervised learning* and successful applications have been booked in: function approximation, pattern recognition and, navigation of mobile robots [5, 12].

However, for many real world systems such as mobile robots working in dynamic environments, training instances in the form of stimuli and desired response are not easily available. Therefore, the robot has to learn for every stimulus the optimal response directly by interacting with its environment. This type of learning method falls into a class of learning methodology called *reinforcement learning* (RL). In RL the robot learns to associate the right responses to different stimuli of the world. It involves four components: the robot, its environment, a learner[a] (controller) to be trained and a trainer that provides *only* a scalar reinforcement signal.

Although RL method fits very nicely to robot learning, it is a slow learning process - it takes too long for the controller to converge toward the desired performance. There are many reasons [2, 8, 9] that contribute to the slow convergence of RL. The major one, however, is that the controller does not know before hand where to search in action space for suitable reactions. This problem stems from the definition of RL: *reinforcement based learning robots learn by doing and*

[a] In this paper we use learner and controller interchangeably.

do not require a teacher. To overcome the problem, we have lifted up the above *unsupervised learning* restriction by providing the learner with a *bias component*. The bias component can be compared with a teacher in supervised learning. However, it does not supply the learner the desired response, hence we still demand the desired response to emerge from RL. Apart from accelerating the learning process, biasing enables the learner to avoid those actions that takes the robot to undesirable locations, thereby making the learning process safe [11].

The paper is organized as follows. Section 2 presents briefly the architecture of the bias component, from which the controller gets a rough action. Section 3 describes the proposed learning architecture and adaptation algorithms. Section 4 presents the trajectory and the learning curves of the robot. At last, a conclusion is drawn from the experimental results.

2 Bias Component

The agent, for which the simulator is built, is a two wheeled $60cm$ square and $40cm$ high TRC mobile robot, figure 1. On the front periphery of the robot there are tactile and sonar sensors. The sonar sensors are programmed for a time out distance of $2m$. Our simulator assumes sonar values are repeatable and motor actions are invertible. In addition, the simulated robot is given a capability to access global information - such as its position.

Figure 1. TRC robot

The architecture of the bias component is similar to [15] and is shown in figure 2. It consists of two purposive fuzzy behaviors: `obstacle avoidance` and `goal following`. As the purpose of the bias component is to deliver a rough estimate of the optimal action, it suffices to have few fuzzy rules and coarse input/output granulation levels.

To come up with few fuzzy rules, the sonars are first grouped into five regions: `right corner`, `right`, `front`, `left` and, `left corner` corresponding to their physical location on the TRC. Subsequently, from each region only the sonar that has the minimum reading is considered[b], i.e.,

$$D_j = \min_i(S_{i,j}) \quad i = 1, \ldots, N_j \ ; \ j = 1, \ldots, 5 \qquad (1)$$

where D_j is the depth value of region j, $S_{i,j}$ is the reading of sonar i located inregion j and, N_j is the number of sonars in the region.

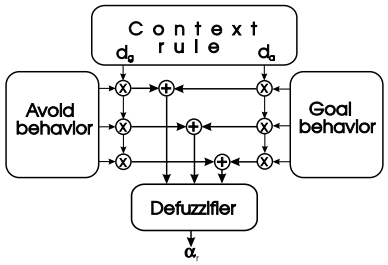

Figure 2. Bias component architecture

The obstacle avoidance behavior has a set of 32 fuzzy rules that are build by granulating the depth values of each regions into two fuzzy sets: `n` (near) and `f` (far). Whereas the goal following behavior has a set of three fuzzy rules that are constructed by fuzzifying the acute angle θ between the robot heading and the vector connecting the current robot and goal locations into three fuzzy levels: `l` (left), `f` (front) and `r` (right). The output of the bias component is the turn rate[c] of the robot that is fuzzified into three fuzzy sets: `L` (Left), `F` (Front) and `R`(Right).

The obstacle avoidance and goal following behaviors output vectors $\boldsymbol{\alpha}_a$ and $\boldsymbol{\alpha}_g$ respectively. The elements of the vectors indicate the activation levels of the output fuzzy sets. To combine the outputs of the behaviors a simple behavior blender with constant *desirability functions* $d_a = 0.9$ and $d_g = 0.1$ is employed. The blender blends the outputs of the behaviors using Eqn. (2) and passes the fused vector $\boldsymbol{\alpha}_f$ to the defuzzifier, which decodes $\boldsymbol{\alpha}_f$ to a crisp value α using centroid technique.

$$\boldsymbol{\alpha}_f = d_g\boldsymbol{\alpha}_g + d_a\boldsymbol{\alpha}_a \qquad (2)$$

Note that when the robot is controlled by the bias component, most of the time it either collides or follows non-optimal trajectories. It is only for very simple tasks and carefully chosen desirability values that the bias component produces smooth and short trajectories. An example is shown in figure 3 where the

[b] On real robot this assumption does not hold true and another technique must be sought, see [6] for a possible method.

[c] The velocity of the robot is kept constant and only the turn rate is controlled within the range of $[-\pi/9, \pi/9]$ rad/sec.

robot has failed to follow optimal trajectory, though it reached the goal[d]. It is also very simple to find yet another example where the robot would fail to reach the goal point at all!

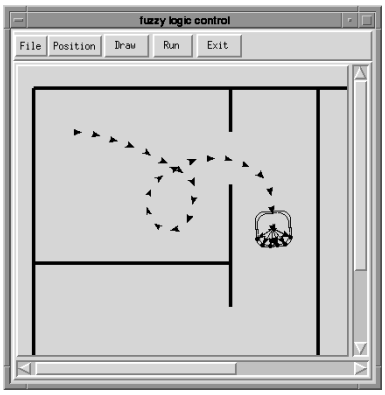

Figure 3. Non optimal trajectory

3 Learning Component

The task of the learner is to take the robot from a home location $\boldsymbol{p}_r(0)$ to some goal location $\boldsymbol{p}_g$. We assume that both the robot and goal locations are specified in a Cartesian coordinate system. Furthermore, at any time the robot determines its position by *dead reckoning* method.

The learner has eight inputs and one output. The first five inputs are depth information of each regions, Eqn. (1) and the remaining three inputs are the robot's current heading θ and position $\boldsymbol{p}_r(t)$. However, before these inputs are applied to the learner they are converted to "primed" quantities by normalizing them appropriately. Hence the input, commonly called situation in connectionist, is characterized by a vector,

$$\boldsymbol{x} = \left(\acute{D}_1, \acute{D}_2, \acute{D}_3, \acute{D}_4, \acute{D}_5, \acute{\theta}, \acute{x}_r, \acute{y}_r\right)^T \qquad (3)$$

Similar to the bias component, the learning component maintains the vehicle velocity constant and controls the turn rate.

3.1 Trainer

Our trainer has two terms that penalize the learner immediately for every bad actions chosen. The first term f_1 penalizes the learner whenever the robot collides with or moves close to obstacles. If the robot collides, it is penalized by a fixed value, otherwise if the minimum depth reading is less than a certain threshold, the trainer penalizes the learner proportional to the inverse of the minimum reading with D_n as a proportional constant. Therefore, the term that teaches the robot to keep away from obstacles is:

$$f_1 = \begin{cases} -3 & : \text{ if collision} \\ -D_n / \min_j(\acute{D}_j) & : \text{ if close} \\ 0 & : \text{ otherwise} \end{cases} \qquad (4)$$

The other term f_2 teaches the robot how to *approach* a goal point. It computes first the acute angle θ between the robot heading and the vector connecting the current robot and goal locations. Then as long as $|\theta| < \Theta$, for some positive Θ, the trainer penalizes the robot proportional to $|\theta|$. Beyond Θ, however, the robot is penalized as if a collision has occurred. This forces the robot to explore only the space which lies between $\pm\Theta$ from the goal direction, there by bounding both the network size and the exploration space.

$$f_2 = \begin{cases} -|\theta|/\Theta & : \text{ if} -\Theta \leq \theta \leq \Theta \\ -3 & : \text{ otherwise} \end{cases} \qquad (5)$$

The total immediate reinforcement r is the sum of the two terms, $r = f_1 + f_2$. Note that the trainer does not teach the robot directly how to *reach* the goal. It trains only how to approach (f_2) the goal without collision (f_1). Therefore, the above reinforce function presupposes that the environment satisfies the constraint that it has a free way (path) through which the robot can reach the goal without collision.

3.2 Learner

The learner architecture is a feed forward neural network consisting of RBF neurons in the input layer and a stochastic neuron in the output layer, figure 4. The architecture is an *actor-critic* type. The critic element is a one step ahead predictor of the expected future discounted sum of reinforcement values (utility). And the actor element is a multi-parameter stochastic unit that generates actions stochastically from a given distribution [4]. In the architecture, all neurons are tied up to the input and only a winning neuron is connected to the output. Each neuron represents a localized receptive field of width Σ that covers a hyper-sphere in the input space. In the present architecture the width of the receptive fields are all the same and kept fixed.

The learner is initially empty but grows gradually, similar to the work of [3], as it learns and explores the environment. When a new situation is presented to the learner, existing neurons (if any) compete to win the situation. If a winning neuron exists, it will beconnected to the output layer to generate an action.

[d] The bias component has been implemented on the real robot and the same result is obtained for the environment depicted.

Otherwise, a new neuron j is introduced and the following four learning parameters are attached to it: a) utility, u_j, b) prototypical action, p_j, c) output weight, w_j and, d) center position, $\boldsymbol{c}_j$. Each ofthese parameters are initialized first and evolve later through RL.

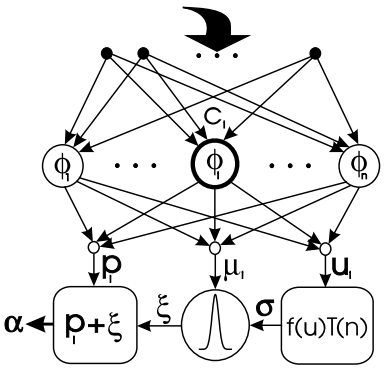

Figure 4. The architecture of the learner

The winning neuron generates an action by exploring only a restricted area around its prototypical action p_j. Restricted exploration accelerates the learning speed by focusing the search area to a fraction of the total action space. To enforce exploration, a *Gaussian stochastic unit* with parameters (μ, σ) is introduced at the output layer [14]. The parameters of the unit are directly determined from the learning parameters of the currently winning neuron using,

$$\mu = w_j \quad \text{and} \quad \sigma = T(n)f(u_j) \tag{6}$$

Where μ is the mean of the distribution, σ is the extent to which the stochastic unit searches for a better action, $T(n)$ is the search-range temperature, n is trial number, $f()$ is the logistic function that take values between $[0, 1]$, Once the parameters are determined, the unit draws a random number $s = \mathcal{N}(\mu, \sigma)$ and generates the final action by modulating the prototypical action with the random number, i.e.,$a = p_j + s$. The temperature T is cooled down[e], similar to [1], every time a trial[f] is started, so that the stochastic unit produces progressively deterministic actions.

Before learning starts the robot is located at origin $\boldsymbol{p}_r(t = 0)$. At this location, the robot perceives a situation $\boldsymbol{x}$. Since the learner is empty (has no neurons), it can not generalize the situation. Therefore, it invokes the bias component. Upon request, the bias component sends its action to the learner. The learner receives the action, adds a neuron, attaches the above learning parameters to the new neuron and initializes the parameters as described below.

To every new neuron j, the algorithm initializes the learning parameters as follows: the center position $\boldsymbol{c}_j$ is equated to the perceived situation $\boldsymbol{x}$, the prototypical action p_j is set to the action received from the bias component, the utility u_j is estimated by computing the terms of the reinforce function for the current states of the robot and sonars readings and finally, the weight w_j is set to zero.

After initializing the parameters, the learner explores and generates action that moves the robot to a new location $\boldsymbol{p}_r(t + 1)$. At this location, the trainer computes the immediate reinforcement $r(t + 1)$ for the action that brought the robot from $\boldsymbol{p}_r(t)$ to $\boldsymbol{p}_r(t + 1)$ and the robot perceives a new situation $\boldsymbol{x}$. The new situation is presented to the learner that identifies first the winning neuron closest to the situation, i.e.,

$$\begin{aligned} winner &= arg \min_i(d_i) \\ d_i &= (\boldsymbol{c}_i - \boldsymbol{x})^T(\boldsymbol{c}_i - \boldsymbol{x}) \end{aligned} \tag{7}$$

If the distance of the winning neuron is larger than Σ, the situation is regarded as novel and the learner invokes the bias component and adds a neuron as discussed above. This way of adding neurons is called *distance driven*. Otherwise the situation is not new and can be generalized.

Next the learner adapts the learning parameters of the previous winning neuron using the immediate reinforcement received and the utility value of the current winning neuron. Thereafter, the learner explores and generates an action for the new situation. If the new action results in collision or takes the robot to the goal, the current trial is terminated, the robot is relocated to home location and, a new trial is started. Otherwise, adaptation and exploration continue until the robot collides or reaches the goal.

3.3 Adaptation

Before the learner generates an action for the present situation, it adapts the learning parameters of the previous winning neuron. Each of the learning parameters are adapted using different adaptation algorithms and error sources.

The utility value of the previous winning neuron $u_j(t)$ is updated by *temporal difference* (TD) method [13]. Assuming neuron i is the present winning neuron with an associated utility $u_i(t + 1)$, $r(t + 1)$ is the immediate reinforcement, and γ is a real value between

[e] Millán[10] has reported that his learner has determined suitable reactions without employing annealing techniques. However, this is only true if the learner starts near to the optimal actions and utility values - a case which is difficult to meet in general.

[f] A trial is a trajectory that starts at the home location and terminates when the robot collides or reaches the goal.

$[0, 1]$, the estimation error of u_j (commonly called TD error) between the estimates at $t+1$ and t is,

$$\delta(t+1) = r(t+1) + \gamma u_i(t+1) - u_j(t) \qquad (8)$$

During learning $\delta(t+1)$ is different from zero either because the utility values do not yet converge or the robot has chosen a non optimal action. If $\delta(t+1) < \Delta$, where Δ is some negative constant, then the situation at time t is incorrectly classified to neuron j. Because, even if the situation is close to neuron j as measured by Eqn. (7), it is found to have quite a different utility value from u_j. Therefore, the learner splits this situation from neuron j by creating and adding a new neuron at that situation. This is a second way of adding neuron and is called *error driven*, where the error is the TD error. Otherwise, if $\delta(t+1) > \Delta$, then u_j is adapted by:

$$u_j(t+1) = u_j(t) + \Delta u_j(t+1) \qquad (9)$$

$$\Delta u_j(t+1) = \begin{cases} \eta_r \ \delta(t+1) & \delta(t+1) > 0 \\ \eta_p \ \delta(t+1) & \delta(t+1) < 0 \end{cases} \qquad (10)$$

where η_r and η_p are two learning rates with $\eta_r > \eta_p$. The utility u_j is adapted less intensively when the TD error is negative than when it is positive. This is because a negative TD error is probably caused by bad action selection that results in a less utility estimate [11].

The output weight w_j directly controls the mean μ of the output stochastic unit and is updated in a direction that lies along the gradient of the expected utility. Williams' REINFORCE algorithm [14] is employed to update the weight w_j,

$$w_j(t+1) = w_j(t) + \Delta w_j(t+1) \qquad (11)$$

$$\Delta w_j(t+1) = \begin{cases} \beta_r \delta(t+1) e_j & \delta(t+1) > 0 \\ \beta_p \delta(t+1) e_j & \delta(t+1) < 0 \end{cases} \qquad (12)$$

where $\delta(t+1)$ is the TD error given by Eqn. (8) and e_j is the characteristic eligibility of w_j that measures how influential w_j was in determining the stochastic action [14]. Similar to utility update, the weight w_j is updated less intensively when the TD error is negative than when it is positive, i.e. $\beta_r > \beta_p$.

Finally, the center position $\boldsymbol{c}_j$ of the winning neuron is shifted to ward $\boldsymbol{x}$ using,

$$\Delta \boldsymbol{c}_j = \epsilon(\boldsymbol{x} - \boldsymbol{c}_j) \qquad (13)$$

where ϵ is the learning rate. Our present architecture prevents neurons from collapsing in a region of high data density, since it activates only one neuron for every situation and the widths of all neurons are constant.

Every time the robot moves, the learner keeps track of the winning neuron $j(t)$, its associated utility value $u(j(t))$ and, the immediate reinforcement $r(t)$ along the trajectory. If a trajectory leads to the goal, the learning algorithm *back up* the utility values of all neurons that lie along this trajectory [7, 11]. While utility, output weight and, center position are adapted at each move, prototypical actions are *replaced* by the actual actions if the robot reaches the goal with the best total reinforcement. We define total reinforcement as the sum of immediate reinforcements the learner receives till the robot reaches the goal point.

$$R = \sum_{t=0}^{T} r(t) \qquad (14)$$

Where T is total number of moves required to reach the goal. If the robot reaches the goal through a trajectory whose total reinforcement is greater than the maximum R_{max} so far obtained, the algorithm replaces the prototypical actions of all neurons that lie along the trajectory by their respective actual actions.

4 Results

We validate our architecture and learning algorithm on a simulator made for the TRC robot. The simulator has simplified dynamics and assumes noise free sensors. However, it takes into account the physical dimensions of the robot by reducing its size proportionally and places each sensors at the same locations as in the real robot. Besides, the fuzzy rules wired for the real robot are directly transfered without tampering to the simulator.

Figure 5 shows the final trajectory of the robot and figure 6 shows the number of neurons and total reinforcement value against the number of trials. It is observed that during the first eight trials the robot failed to reach the goal. This is not surprising, because the learner is empty and has to acquire enough situation-action pairs. Note that the total reinforcement Eqn.(14) is not defined if the robot fails to reach the goal, hence no data is available to plot.

At the ninth trial the robot reached the goal for the first time. It is during this trial that the learner received the lowest total reinforcement (figure 6 below). Furthermore the path followed during this trial looks more of haphazard motion. After the robot has reached the goal at the ninth trail, it has chosen eight times non optimal actions that ultimately lead to real or virtual collisions (figure 6 below). This is due to the exploratory nature of the actor element and is common in any reinforcement learning [11]. From the twelfth

trial and afterwards the bias component has practically stopped intervening. Besides the size of network (figure 6 above) has become saturated (between trials only few neurons are added). This indicates that the learner has already started operating in reinforcement mode.

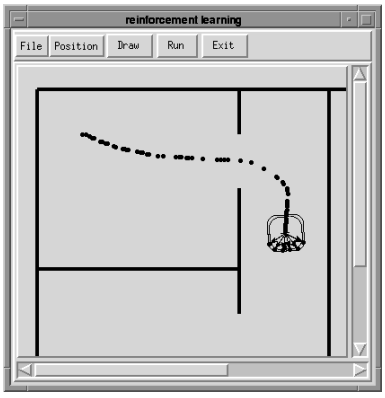

Figure 5. Robot trajectory

After trial thirty the robot has visited the goal constantly, the total reinforcement remains stable within ± 3, except at trial 41, where the learner explore other actions from the currently known optimal values (figure 6 below). In subsequent trials, however, it has quickly discovered its previous performance. The final result (figure 5) demonstrates that the robot has indeed adapted quickly the coarse and instinct skill acquired from the bias component to get smooth and planned like trajectory.

5 Conclusion

We have proposed a feasible robot learning architecture that learns quickly from reinforcement signal alone. The architecture has two components: a bias component and a learning component. Initially the bias component intervenes in the learning process frequently to resolve unknown situations. As learning proceeds, however, it stops intervening and the learner optimizes (refines) the acquired situation-action pairs using the reinforcement signal. The architecture has been tested on a simulated TRC mobile robot for goal reaching task. The final planned like trajectory and the number of trials required validates our approach. Work is going on to transfer the obtained result on the real robot.

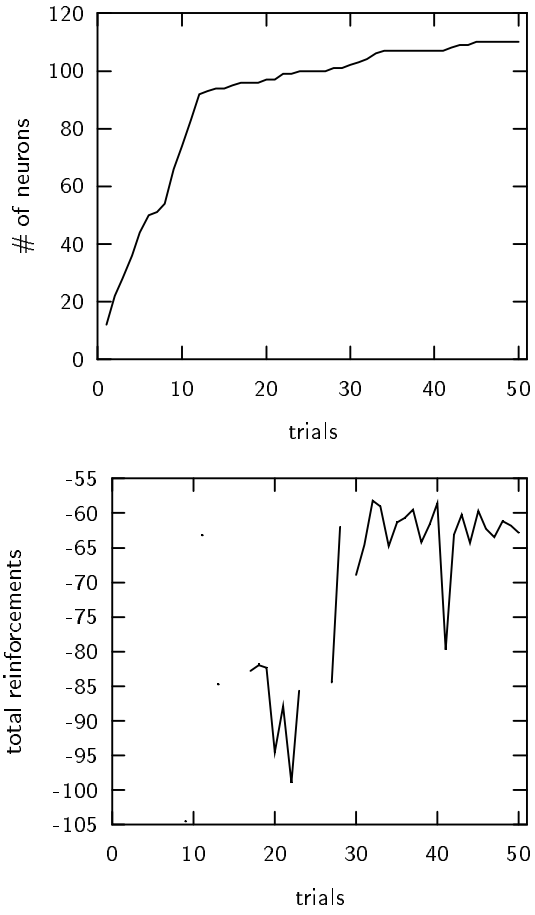

Figure 6. Top: Network size, Bottom: Total Reinforcements.

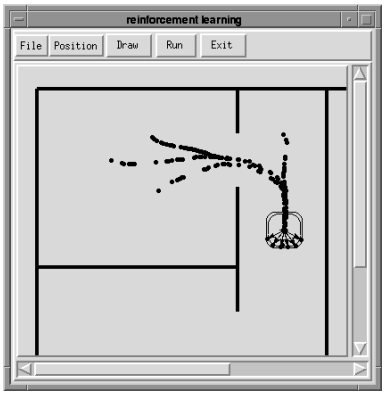

Figure 7. Trajectories of the robot for other sample start points

6 Acknowledgment

We would like to thank J. R. Millán for clarifying some points on reinforcement learning. The support

given to the first author by DAAD under grant code 413/ETH-4-BOA is greatly acknowledged.

References

[1] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[2] Rodney A. Brooks and Maja J. Matáric. Real robots, real learning problem. *Robot Learning*, pages 193–214, 1993.

[3] Jörg Bruske and Gerald Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):834–846, 1995.

[4] Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real valued function. *Neural Networks*, 3:671–692, 1990.

[5] Getachew Hailu. Distributed fuzzy and neural network based navigational behaviours. Technical Lab. Report H/696, CAU, Cognitive Systems Laboratory, 1996.

[6] Getachew Hailu, Jörg Bruske, and Gerald Sommer. Fuzzy logic control of a situated agent. In *Seventh International Fuzzy System Association - World Congress*, pages 494–500, Prague, 1997.

[7] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[8] Maja J. Matáric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1994.

[9] R. Andrew McCallum. Using transitional proximity for faster reinforcement learning. In *Machine Learning - Proceedings of the Ninth International Workshop (ML92)*, pages 316–321, Aberdeen, 1992.

[10] José R. Millán. Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, 15:275–299, 1995.

[11] José R. Millán. Rapid, safe and incremental learning of navigation stratagies. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):408–420, 1996.

[12] Dean A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Publishers, 1993.

[13] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[14] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[15] John Yen and Nathan 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):971–977, 1995.