In International Conference on Intelligent Robots and Systems (IROS03), Las Vagas, Navada, US, October 2003. IEEE-RSJ.

Using Policy Gradient Reinforcement Learning on Autonomous Robot Controllers

Gregory Z. Grudic
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
USA

Vijay Kumar
GRASP Lab
University of Pennsylvania
Philadelphia, PA 19104-6228
USA

Lyle Ungar
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
USA

Abstract— **Robot programmers can often quickly program a robot to approximately execute a task under specific environment conditions. However, achieving robust performance under more general conditions is significantly more difficult. We propose a framework that starts with an existing control system and uses reinforcement feedback from the environment to autonomously improve the controller's performance. We use the Policy Gradient Reinforcement Learning (PGRL) framework, which estimates a gradient (in controller space) of improved reward, allowing the controller parameters to be incrementally updated to autonomously achieve locally optimal performance. Our approach is experimentally verified on a Cye robot executing a room entry and observation task, showing significant reduction in task execution time and robustness with respect to un-modelled changes in the environment.**

I. Introduction

Building continuous controllers with a provable global performance is well known to be very difficult in all but the simplest of cases. Our philosophy is to develop simple controllers whose dynamic characteristics are well understood, and switch between these controllers depending on the task and on sensory feedback. This essentially means that the state space is divided into discrete partitions, and the behavior of the robot system changes when it leaves one partition and enters another partition. This paradigm can be viewed in a hybrid systems framework where there are many discrete modes, each of which represents a continuous dynamic system, and the hybrid robot controller switches between the discrete modes [1], [2], [3], [4], [5]

In this context, learning can be used in two ways. First, learning can be used to improve the performance of the controller in each mode. This generally reduces to a parametric learning problem. Second, learning can be used to determine the conditions for mode-switching, or the boundaries that characterize each partition. These conditions are algebraic equations for the *invariants* characterizing each mode, and the *transitions* that characterize switches between modes. Learning at this level can fundamentally change the behavior of the controller.

An attractive alternative to hand coding robot controllers is to instead code learning algorithms which allow the robot to autonomously learn to appropriately interact with their environment. Reinforcement Learning (RL) is a paradigm by which agents learn to improve their behaviour through interaction with their environment [6]. RL starts with the assumption that it is easy to specify under what conditions an agent (robot) has failed or succeeded in a specific task. For example, with a mobile robot we can determine relatively easily when it has collided with an obstacle, or when it has reached its goal state. Such high level feedback from the environment is termed reinforcement reward or feedback, and it is typically intermittent, often with long periods of time passing between successive rewards. The aim of RL is to use such intermittent reinforcement feedback to design a controller that acts optimally in a specific environment.

Although there have been a number of successful applications of RL [7], these applications are typically characterized by relatively small discrete state spaces, and millions of learning episodes. In contrast, robotic systems are typically characterized by large continuous state spaces and environments where millions of learning runs are not feasible. As a result, there have been relatively few published examples of RL on real robotic systems. One example is [8], where a robot's controller is specified by a set of behaviors, and learning is done by exploring the order in which these behaviours are executing, then choosing the ordering which gives best performance. Another is [9], where learning is bootstrapped by demonstration runs supplied by a human operator, effectively directing search during learning and allowing relatively quick convergence to successful control policies.

In both of the above examples of RL in robotics (as

well as other examples [10], [11], [12], [13], [14], [15], [16]), effective learning is accomplished by building prior knowledge into the learning system. Prior knowledge can be encoded in the choice of behaviours and an initial order of execution [8], or it can be added by a human operator who supplies sample robot trajectories [9].

In this paper we propose to incorporate prior knowledge in the form of a controller specification. Our objective is to develop a framework that can take any standard control specification and apply RL to improve the controller's performance. The motivation for this is twofold. First, although it is difficult to code a controller that performs robustly under a wide range of conditions, programmers can write controllers that effectively work under limited conditions. Second, even if a controller is theoretically guaranteed to perform robustly, it is likely that in practice it will not behave exactly as predicted because theory cannot completely describe the actual dynamics of a real robot. Therefore, much is to be gained for any real control system, if RL techniques can be used to autonomously improve the controllers performance on the actual task it is meant to accomplish. Simply put, our objective is to shift the burden of tuning and refining a complex controller from the designer to an RL algorithm.

Our framework assumes that the controller can be represented by a set of K real parameters $\Theta = (\theta_1, ...\theta_K)$, which define how the robot acts in its environment. Changing one of these θ_k parameter values changes the robot's controller, thus affecting the robot's performance. The goal is to improve the robot's reward as specified by some reward function $\rho(\Theta)$, which is also a function of the controller parameters Θ. In mobile robotics, a possible example of a reward function is one which gives negative reward whenever the robot collides with an obstacle, and a positive reward whenever it attains a goal position. Because the robot's control policy is completely defined by a parameter vector Θ, the Policy Gradient Reinforcement Learning (PGRL) [17], [18], [19] framework can be directly applied to modifying parameter values to improve controller performance. In addition, PGRL algorithms are well suited to continuous problem domains and are guaranteed to converge to locally optimal policies. In PGRL learning occurs by estimating a *performance gradient*, $\widehat{\partial\rho/\partial\Theta}$ in the direction of increased reward. The control parameters are updated according to gradient ascent as follows:

$$\Theta_{i+1} = \Theta_i + \alpha \widehat{\frac{\partial \rho}{\partial \theta}} \tag{1}$$

where α is a small positive step size and Θ_{i+1} specifies the updated policy. PGRL algorithms are guaranteed to converge to a locally optimal control policy, and are therefore ideally suited for real world control problems where globally optimal solutions are rarely realizable, and any improvement in controller performance is beneficial.

In Section II we describe the theoretical framework used and the Deterministic Policy Gradient (DPG) algorithm which is used to estimate performance gradients. In Section III experimental results are given on a Cye robot executing an room entry, observation, and exit task. Section IV concludes with a brief discussion.

II. The Reinforcement Learning Formulation

A. POMDP Formulation

We formulate the learning problem as an agent (robot) interacting with a Partially Observable Markov Decision Process (POMDP) [7]. Each time the robot makes a sensor reading, it observes a set D of continuous valued readings (or information variables) symbolized by $\mathbf{z} = (z_1, ..., z_d) \in D \subseteq \Re^d$. Note, that these sensor readings $\mathbf{z}$ are not the same as the actual physical state $\mathbf{x}$ of the robot, which cannot be fully observed. However, we assume that the actual robot's state is partially observable because an infinite time sequence of observations of sensor readings $\mathbf{z}$ can be used to exactly infer $\mathbf{x}$. At $t = 0$, the robot observes an initial set of sensor values denoted by $\mathbf{z}_0$ and continues to interact with the environment for a maximum duration of time T. The paths followed by the agent are continuous in time $0 \leq t \leq T$ and are symbolized by observations $\mathbf{z}(t) = (z_1(t), ..., z_d(t))$. During each episode the expected reward the agent receives at time t, after $\mathbf{z}(t)$ is observed and action a_t is executed, is symbolized by $r(\mathbf{z}(t), a_t) \in \Re$.

The robot's controller is uniquely defined by a set of Q functions $\mathbf{g}(\mathbf{z}, \Theta) = (g_1(\mathbf{z}, \Theta), ..., g_Q(\mathbf{z}, \Theta))$, which are bounded continuous functions defined on $\mathbf{z} \in D$ such that $\Theta = (\theta_1, ..., \theta_K) \in \Re^K$, and $\frac{\partial g_q}{\partial \theta_k}$ exists and is bounded $\forall q = 1, ..., Q$ and $\forall k = 1, ..., K$.

The robot's goal is to incrementally modify the parameters Θ in $\mathbf{g}$ to locally optimize the reward:

$$\rho(\Theta) = \int_0^T \left(\int_D \gamma^t r(\mathbf{z}(t), a_t) \, p(\mathbf{z}|t, \Theta) \, dD \right) dt \tag{2}$$

where $0 < \gamma < 1$ is a discount factor and $p(\mathbf{z}|t, \Theta)$ is the probability the agent enters state $\mathbf{z}$ at time t under the policy specified by Θ. The discount factor γ in (2) implies that the robot receives greater reward if it reaches positive values of reinforcement feedback ($r(\mathbf{z}(t), a_t)$) more quickly. In this sense it is similar to the the standard discounted reward formulation in discrete state spaces [6]. We further assume that $\rho(\Theta)$ is continuous with respect to Θ.

B. A DPG Algorithm

We use the Deterministic Policy Gradient (DPG) algorithm (proposed in [20]) to estimate a gradient ($\widehat{\partial\rho/\partial\Theta}$ in equation (1)) in control parameter space of the reward function given in equation (2). This algorithm is based

a) Robot at Start Position (entry to room)

b) Robot Going Towards Goal

c) Robot at Goal Position (ready to leave room)

d) Robot's Internal Obstacle Representation

Fig. 1. Robot Task.

on the following theorem which allows $\widehat{\partial\rho}/\partial\Theta$ to be estimated (proof proven in [20]):

Theorem: *Let $V(t,\Theta)$ be the remaining part of the reward function (2) after t seconds has passed:*

$$V(t,\Theta) = \int_t^T \left(\int_D \gamma^\tau r(\mathbf{z}(\tau))\, p(\mathbf{z}|\,\tau,\Theta)\, dD \right) d\tau \quad (3)$$

Then, given the assumptions in Section II-A and further assuming that $\frac{\partial V(t,\Theta)}{\partial g_q(\mathbf{z}(t),\Theta)}$ exists and is bounded, then the exact expression for the performance gradient with respect to θ_k, $\forall q = 1,...,Q$ and $\forall k = 1,....,K$, is given by:

$$\frac{\partial \rho}{\partial \theta_k} = \int_0^T \left(\sum_{q=1}^{Q} \frac{\partial V(t,\Theta)}{\partial g_q(\mathbf{z}(t),\Theta)} \frac{\partial g_q(\mathbf{z}(t),\Theta)}{\partial \theta_k} \right) dt \quad (4)$$

The motivation behind the DPG algorithm is to create an online learning framework that continuously updates the hybrid control parameters to steadily improve performance. We are not interested in finding an optimal control policy because, in essence, it is not possible to do this in the complex uncertain environments we are interested in. Our aim is to quickly and efficiently improve performance until a locally optimal controller has been attained. Thus, the algorithm is designed to quickly identify those control parameters that, if changed, will most effectively improve performance. Using the above theorem, the relevance of parameter θ_k can be directly observed by evaluating the term $\frac{\partial g_q(\mathbf{x}(t),\Theta)}{\partial \theta_k}$ in (4) as the robot executes it's task. If this term is zero for the entire episode, then small changes in θ_k are likely to have no affect on the policy and need not be perturbed. Once identified in this way, relevant parameters can be slowly modified in a direction of increased reward, allowing the robot to quickly improve controller performance.

III. Experimental Results

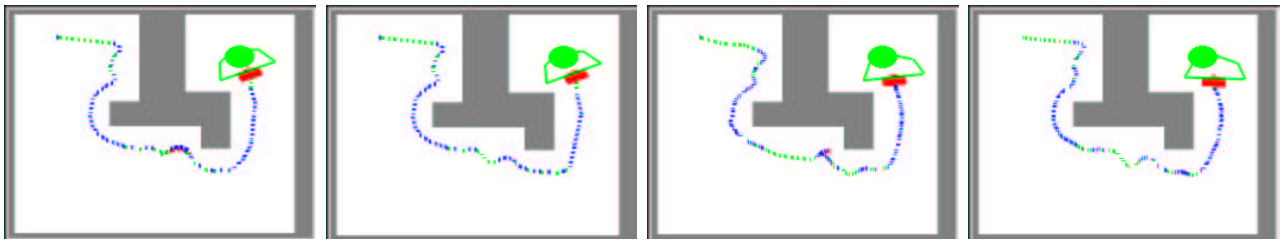
A. The Robot and Task Definition

Our experimental setup consists of a Cye Robot controlled by an on-board laptop computer through an RS232 serial link. The robot's task is to enter a room, follow a path to a goal position, and then exit the room from where it entered. Figure 1a shows the robot where it enters the room at the initial position, Figure 1b shows the robot navigating around the obstacles on its way to the goal position, and Figure 1c shows the robot at the goal position. The room dimensions are 17 feet by 17 feet, and there are four obstacles within the room (see Figure 2 for a representation of these obstacles): an 'L' shaped obstacle behind which the goal is located (at the right end of the room), and two square obstacle. The robot has an internal model of the 'L' shaped obstacle and one of the square obstacles as shown in Figure 1d. Therefore, the robot's internal model differs from the real world in two ways: the square obstacle the robot knows about has moved, and a new square obstacle has appeared which the robot did not know about. The task of the learning system is to learn to compensate for these differences, as well as to compensate for the usual un-modelled dynamics of the robot interacting with the environment.

B. The Controller

We use a mode switching controller, with three modes: *follow potential field*, *avoid obstacle*, and *recovery from collision*. Only one of these control modes is active at any one time. The controller begins in the *follow potential field* mode described below.

The *follow potential field* mode assumes that there exists a rough map of where the stationary obstacles are located, where the goal position is, and the entry and exits of the room. Figure 1d shows the map used in this paper. Given this information, the grassfire algorithm [21] is used to calculate a numerical potential field which the robot can use to navigate to the goal. Whenever the *follow potential field* mode is active, the robot is directed along a direction of lowest potential (when it reaches zero potential the robot is at the goal position). In our implementation of this mode, the room is divided into a 20 by 20 grid, and the grassfire algorithm is used to assign a potential to each grid point. If m denotes the grid index that has the minimum potential of all grids adjacent to the grid the robot is currently occupying, then the *desired* direction the robot is directed to more is given by $\phi = atan2(y,x)$,



a) Simulated Path Before Learning | b) Simulated Path After Learning | c) Actual Path Before Simulator Learning | d) Actual Path After Simulator Learning

Fig. 2. Typical paths towards goal. The light dotted lines (in green) indicate that the controller is in the *follow potential field* mode, and the darker dotted lines (in blue) indicate the *avoid obstacle* mode. Finally, *recovery from collision* mode is indicated by the darkest (in red) dotted line in part c.

where $x = x_m - x_c$, $y = y_m - y_c$, and (x_m, y_m) are the grid coordinates with the minimum potential, and (x_c, y_c) is the current estimate of the robot's position. The *follow potential field* mode uses two potential gradients: one for going toward the goal, and one for returning the the initial position after the goal state has been reached.

If an obstacle is detected within a polygon shaped area, called the obstacle detection region, around the robot, the robot switches from the *follow potential field* mode to the *avoid obstacle* mode. Figure 1d shows the shape of the obstacle detection region used in this paper. This mode switching policy is defined by seven parameters $\Theta = (\theta_{m1}, ..., \theta_{m7})$ which are all initially set to 2 feet. These parameters define seven line segments, each starting from the robot's center and radiating outward in a forward direction at $180/7$ degree intervals. The *avoid obstacle* mode redirects the robot around the obstacle, and once the obstacle is no longer within the obstacle detection region, the *follow potential field* mode is activated once more.

The *recovery from collision* mode is activated whenever the robot detects a collision. This mode controls the robot by applying a negative velocity v to the drive wheels and a desired steering direction ϕ_c that will move the robot away from obstacle it is assumed to have collided with. The *recovery from collision* mode is active for a fixed period of time (T), after which *follow potential field* mode is reactivated.

C. The Simulator

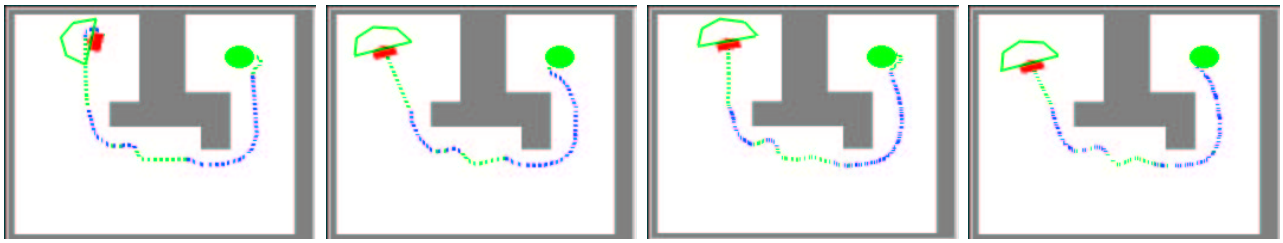
The potential path calculated by the grassfire algorithm assumes a point robot which can holonomically move in any direction. However, the Cye robot is rectangular (see Figure 1) and cannot instantaneously move in any direction. Therefore, the goal of the simulator is to roughly model the robot's geometry and to limit the rate of change in the robot's orientation to 5 degrees each time a control signal is passed to the robot. The simulator also uses the map containing the obstacle positions, the goal position, and the initial position. The mode switching controller described above is also simulated. However, the simulator only incorporates kinematic constraints, and no robot dynamics are considered. Noise in the simulator is modeled as an uncertainty in current robot's position, and is calculated using a uniform distribution in x and y of 4 inches from the actual robot position.

D. Learning Results

We investigated how RL can be used to modify the initial potential field generated by the grassfire algorithm, as well as the mode switching parameters between the *follow potential field* mode and the *avoid obstacle* mode. All other parts of the controller definitions are held fixed (i.e. the calculation of ϕ_c in the the *avoid obstacle* and *recovery from collision* modes, as well as the duration T for which the *recovery from collision* mode is active remains unchanged). The reward function to be maximized is given in (2), where the discount factor is set to $\gamma = 0.99$. The reward for reaching the goal, and for reaching the initial position after the goal is attained, is $r = 1$. A negative reward of $r = -1$ is given for an obstacle collision. Therefore, the robot receives most reward by taking the shortest path between goal and initial states, while still avoiding obstacles. Finally, if the robot doesn't reach the goal within a fixed period of time, then it is given a reward of $r = -1$.

The gradient field generated by the grassfire algorithm has 1600 control parameters: 400 (x,y) grid pairs for the gradient field towards the goal, and 400 (x,y) grid pairs for the gradient field towards the initial position. We denote these parameters as $\Theta = (x_1, y_1, ..., x_{800}, y_{800})$, and the DPG algorithm described in Section II-B, is used to modify these 1600 parameters along a gradient of increased reward. The functions $\mathbf{g}(\mathbf{x}, \Theta)$ used by the DPG algorithm (see Section II-A) are set to $g_m = \theta_m - x_c$ for odd m and $g_m = \theta_m - y_c$ for even m, where θ_m is defined such that $\Theta = (\theta_1, ..., \theta_{1600}) = (x_1, y_1, ..., x_{800}, y_{800})$ and (x_c, y_c) is the current estimated position of the robot. If the robot never uses parameter θ_m during an episode, then $g_m = \frac{dg_m}{d\theta_m} = 0$. Therefore, learning occurs by warping the grid locations of the potential field, and not the values of the potential field at the grid locations.

The Mode Switching controller between the *follow*



a) Simulated Path Before Learning | b) Simulated Path After Learning | c) Actual Path Before Simulator Learning | d) Actual Path After Simulator Learning

Fig. 3. Typical paths from goal to start position.. The light dotted lines (in green) indicate that the controller is in the *follow potential field* mode, and the darker dotted lines (in blue) indicate the *avoid obstacle* mode.

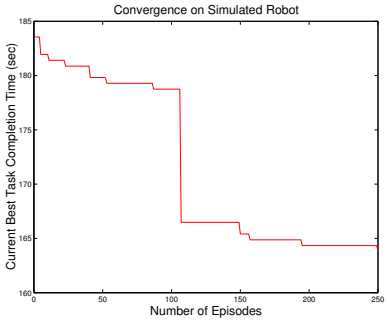


Fig. 4. Convergence results on the simulated Robot.

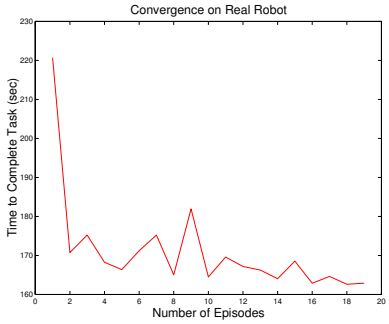


Fig. 5. Convergence results on the actual Cye robot.

potential field mode and the *avoid obstacle* mode is defined by two functions (g_{m1}, g_{m2}). The g_{m1} function is defined by: $g_{m1} = \frac{1}{2}\left[1+\tanh\left(\eta \sum_{i=1}^{7} h_i\right)\right]$ where $h_i = \frac{1}{2}\left[1+\tanh\left(\eta\left(d_i - \theta_{mi}\right)\right)\right]$ and d_i is the minimum distance to an obstacle along section i, η is a positive number(set arbitrarily to 1.0 in this paper), and θ_{mi} are the seven mode switching parameters as defined in Section III-B. We define $g_{m2} = 1 - g_{m1}$. Note that g_{m1} will only go above 0.5 if an obstacle intersects one of the seven pi sections, thus the controller switches from the 'follow potential field' mode, to the 'avoid obstacle' mode. Similarly, when g_{m2} goes above above 0.5, the obstacle has been cleared and the controller switches from the 'avoid obstacle' mode to the 'follow potential field' mode.

The gradient step size (i.e. α in Equation (1)) for the policy update is $\alpha = 0.05$. The DPG (Section II-B) algorithm search step size is set to $P = 0.5$ feet. The cut-off for search for a given policy parameter θ_k is defined by $D^{\theta}_{max} = 0.5$, and in a typical robot run through the room only about 100 of the 1607 parameters (counting both the mode switching and grassfire controller parameters) satisfy these conditions (and thus will be used to estimate a performance gradient).

Figure 4 shows typical convergence of the DPG algorithm on the simulation. Results given show the reward obtained by the best learned control policy as a function of the number of times the robot goes through the room (i.e. the number of episodes). The algorithm converged to a locally optimal policy in about 200 runs through the simulated room, learning to avoid the obstacle that moved as well as the obstacle which it did not know about. A typical path followed by the simulated robot after learning in simulation is shown in Figure 2b for the initial position to goal position phase, and in Figure 3b for the goal position to initial position phase. Corresponding runs of the actual robot are shown in 2d and Figure 3d respectfully. For both the simulated and real robot's, the control policy learned by the DPG algorithm reduces the overall time the robot spends in the room by about 14 percent, completely eliminating obstacle collisions.

Figure 5 shows the convergence results of the DPG algorithm running on the real robot for a total of 20 episodes through the room. The initial jump in the reward between episode 1 and 2 reflects the learning done in simulation (i.e episode 1 is before learning in simulation and episode 2 is after learning in simulation). The best policy continues to generally improve over the next 18 robot passes.

IV. Conclusion

We have demonstrated that reinforcement learning can be used to effectively improve the performance of mode switching hybrid controllers. Our framework simultaneously modifies both the control parameters within modes, as well as the parameters that govern when the controller

switches between modes. The Policy Gradient Reinforcement Learning (PGRL) framework is used to calculate a gradient of increased reward in controller space, allowing the robot to autonomously update its controller to locally optimal policies. PGRL allows learning to be seamlessly incorporated into the robot's hybrid controller, thus allowing the control policy to be continually refined as conditions change.

V. ACKNOWLEDGMENTS

Thanks to Ben Southall and Joel Esposito for implementing the Grassfire Algorithm. This work was funded by the GRASP Lab, the IRCS at the University of Pennsylvania, and by the DARPA ITO MARS grant no. DABT63-99-1-0017.

VI. REFERENCES

[1] R. Arkin and T. Balch, *Artificial Intelligence and Mobile Robots*, ch. Cooperative Multiagent Robot Systems. MIT Press, 1998.

[2] M. Mataric, "Issues and approaches in the design of collective autonomous a gents," *Robotics and Autonoumous Systems*, vol. 16, pp. 321–331, Dec 1995.

[3] J.Lygeros, C.J.Tomlin, and S.Sastry, "Multiobjective hybrid control synthisis," in *Proceedings of hybrid and realtime systems*, vol. 1201 of *Lecture Notes in Computer Science*, Grenoble: Springer-Verlag, March 1997.

[4] D. Liberzon and A. S. Morse, "Basic problems in stability and design of switched systems," *IEEE Control Systems*, vol. 19, pp. 59–70, Oct. 1999.

[5] M. Branicky, *Studies in Hybrid Systems: Modeling, Analysis and Control*. PhD thesis, MIT, Cambridge, MA, 1995.

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

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[8] F. Michaud and M. J. Mataric, "Representation of behavioral history for learning in nonstationary conditions," *Robotics and Autonomous Systems*, vol. 29, no. 2, pp. 187–200, 1999.

[9] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces," in *Proceedings of the Seventeenth International Conference on Machine Learning*, vol. 17, pp. 903–910, Morgan Kaufmann, June 29 - July 2 2000.

[10] S. Mahadevan, "Enhancing transfer in reinforcement learning by building stochastic models of robot actions," in *Proceedings of the Ninth International Conference on Machine Learning*, vol. 9, pp. 290–299, Morgan Kaufmann, 1992.

[11] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[12] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behaviour aquisition for a real robot by vision-based reinforcement learning," *Machine Learning*, vol. 23, pp. 279–303, 1996.

[13] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *IEEE Int. Conf. on Robotics and Automation, ICRA 02*, 2002. IEEE Intl. Conf. on Robot. and Automat., 2002.

[14] A. S. E. Martinson and R. C. Arkin, "Robot behavioral selection using q-learning," in *In the Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[15] A. H. F. Y. Wang, B. Thibodeau and R. Grupen, "Learning optimal switching policies for path tracking tasks on a mobile robot," in *In the Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[16] S. Mahadevan, "Continuous-time hierarchical reinforcement learning," in *Proceedings of the Eighteenth International Conference on Machine Learning*, vol. 18, pp. 186–193, Morgan Kaufmann, 2002.

[17] G. Z. Grudic and L. H. Ungar, "Localizing search in reinforcement learning," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, vol. 17, pp. 590–595, Menlo Park, CA: AAAI Press / Cambridge, MA: MIT Press, July 30 - August 3 2000.

[18] L. Baird and A. W. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems* (M. I. Jordan, M. J. Kearns, and S. A. Solla, eds.), vol. 11, (Cambridge, MA), MIT Press, 1999.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[20] G. Z. Grudic, V. Kumar, and L. H. Ungar, "Refining autonomous robot controllers using reinforcemnt learning," *Submitted*, 2003.

[21] D. Lee, *The map-building and exploration strategies of a simple sonar-equipped robot : an experimental, quantitative evaluation*. Cambridge ; New York: Cambridge University Press, 1996.