INTERACTIVE REFINEMENT OF CONTROL POLICIES FOR AUTONOMOUS ROBOTS

Vinay Papudesi and Manfred Huber
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019-0015
{papudesi, huber}@cse.uta.edu

ABSTRACT
The automation of various aspects of life through robotics is a promising and useful mechanism to the general end-user. Robots are required to accept human guidance, and in its absence, have to operate autonomously while ensuring safety and optimality. This paper presents an approach to variable autonomy that extends reinforcement learning with the capability of integrating user guidance at varying levels of abstraction into its control policies. This permits the modification of robot behavior based on the preferences of the user and faster policy acquisition. User commands are filtered to satisfy a priori constraints and task requirements. The applicability of the approach is illustrated with its operation in a task of navigation.

KEY WORDS
Reinforcement Learning, Human-Robot Interaction

1. Introduction

Robotic automation in daily chores is a promising technology for the general end-user, with a large area of applicability. Specifically, tasks that involve repetition or dangerous environments could be automated. Further, providing such agents with the capability to operate with variable autonomy permits the end-user to monitor and control its operation with minimal effort. This is especially important to extend the applicability of such systems to include the population of end-users that are unskilled in operating robots. Consequently, the human-robot interface should facilitate the incorporation of user commands at different levels of abstraction.

While recent research efforts in this direction have been numerous, they were conducted with only the skilled operator in mind. In contrast, this work integrates potentially unreliable user commands into the control policies while ensuring that both robot safety and task completion are achieved. To achieve this, user commands at different levels of abstraction are filtered prior to their integration into an autonomous learning component.

Reinforcement learning [10, 5] has been used extensively as a formal mechanism for autonomous agents to learn optimal solutions to a variety of real-world tasks. This strategy allows the agent to learn from its interaction with the environment, through which it obtains numerical feedback, called reinforcements, which measure the progress of the agent with respect to a particular task. Reinforcements are typically obtained intermittently, when the agent achieves a goal or sub-goal. Further, reinforcements measure the utility of a single action and not the preceding potentially sub-optimal sequence of actions. These qualities of the feedback make learning from reinforcements slow, and a more continuous feedback structure is desired from the reward function. However, it is generally non-trivial to construct such reward functions without imposing further restrictions on the nature of the task; similarly, simplifications in its design might lead to incorrect executions of the task.

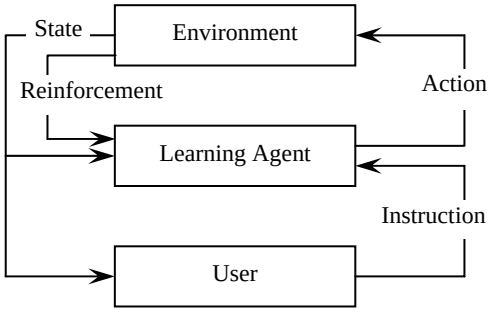

Figure 1. Learning from reinforcements and advice

To avoid these restrictions, this paper presents an advisor-based solution [4] to the problem of modifying the feedback pattern. We define a composite reward function with a static, task-specific component and a tunable user-based component. In the context of this work, a task is defined as reaching one or more accepting states. Positive task rewards are assigned to all actions that lead to such accepting states. Variable autonomy is achieved by integrating user commands into the user-based reward component. This approach adopts the advisor-based extension to Reinforcement learning as depicted in Figure 1. The user commands are incorporated into the user-based reward component, thereby modifying the learned control policy. While the general notion behind this framework is that humans are better trained at most

practical tasks that require automation, separating and filtering the user commands through the user-based reward component permits the provision of unreliable and incorrect user commands.

The advantages of learning from advice were identified by McCarthy [8] in his proposal of the *advice taker*. Clouse [1] defines a training agent that provides advice and an associated feedback that is applied to the learning agent regardless of the true feedback from the environment. Similarly, Lin [6] suggests the use of advice in the form of a sequence of actions that yield some predetermined reinforcement. Placing the onus of computing such reinforcements on the advisory entity, specifically humans, could produce unpredictable and potentially incorrect policies. Maclin and Shavlik [7] describe an approach to learning from reinforcements and advice that uses knowledge-based neural networks; advice is applied as weight-changes, thereby producing a modified policy.

In the next section, we describe learning with composite reward functions. In Section 3, we define an example task of robot navigation and illustrate the modification of control policies with human advice. Finally, Section 4 presents the conclusions.

2. Composite Reward Functions

The learning problem is modeled as a Markov Decision Process $M(S, A, T, R)$ with a state space S, a set of actions A, a probabilistic transition function $T : S \times A \to S$ and a composite reward function $R : S \times A \to \Re$ defined as the unweighted sum of the task reward R_τ and user reward R_υ components. Q-learning [12] is used to evaluate the state-action utility function $Q : S \times A \to \Re$, called the Q-value function.

The task reward component is defined as a static reward structure that mirrors the task requirements. The construction and verification of a task reward function is generally non-trivial. Often, the agent derives a large positive reward when it reaches an accepting state. More complex reward structures can be created by assigning rewards to intermediate states that represent specific sub-goals or check-points.

However, adding intermediate reinforcements to the task reward function can unpredictably result in the formation of a policy that does not execute the task correctly. Such problems have been well documented in relation to *reward shaping* [11], in which external rewards are added to R_τ while maintaining an invariant optimal policy. A typical scenario is the formation of a state-action loop involving one or more states; the agent derives more reward from moving through these state configurations *ad infinitum* than performing a more distant task goal.

We counter this problem by imposing bounds on all intermediate task reinforcements. In particular, intermediate task rewards must maintain the monotonicity of the Q-value function; states encountered in optimal paths to an accepting state must have strictly increasing Q-values. Task reward functions that do not satisfy this property can be modified by enforcing bounds that are similar to those imposed on user rewards (Section 2.1). There is, however, no guarantee that the task is accomplished in the same manner. Furthermore, the bounds imposed on task and user reward functions do not prohibit changes to the optimal policy as with shaping [9]; instead, they prevent the formation of self-sufficient loops while allowing for modified policies.

2.1 Formulating R_υ from Human Advice

Advice is typically in the form of recommended states or state-action pairs. The recommendation of state-action pair (s, a) leads to a preference for action a when the agent is in state s. The recommendation of state s leads to a preference for the path of state-action pairs starting at the current state and ending at s. The ambiguity of selecting an appropriate sequence of state-action pairs is typically handled using a model-based or utility function-based approach. The model-based solution computes the sequence with the maximum probability or involving the minimum number of intermediate actions. The utility function-based approach uses the current Q-function to determine the best sequence of state-action pairs.

The set of recommended state-action pairs is mapped onto a numerical function, $bias : S \times A \to \aleph$. If the total number of actions available at state s is represented by n_s, the numerical *bias* is defined as:

$$bias(s,a) \leftarrow \begin{cases} bias(s,a) + n_s - 1 & \text{if } (s,a) \text{ is recommended,} \\ bias(s,a) - 1 & \text{if } (s,a) \text{ is an alternate.} \end{cases}$$

This definition ensures that the sum of the numerical biases at any state is always zero. The recommendation of all n_s actions at state s increases $bias(s,a)$ by $n_s - 1$ when action a is recommended and decreases it by 1 for each of the remaining $n_s - 1$ actions, for a net change of zero. This allows random advice (such as when the advisor is unsure of the true best action) to ultimately be rejected by the agent.

R_υ is derived as a generic transformation of the *bias* function, a trivial example of which is an *equal-to* mapping:

$$R_\upsilon(s,a) \leftarrow bias(s,a)$$

A severe problem with this mapping is the independence of the user reward structure from the Q-value function. For states with arbitrarily small utilities, this would result in the formation of a self-loop (a loop with a single action that leads to the agent remaining in the same state forever). Handling the dependency on the utility function results in mappings similar to:

$$R_\upsilon(s,a) \leftarrow bias(s,a) \cdot k \cdot \left((1-\gamma) \cdot Q(s,a) - R_\tau(s,a)\right)$$

where $k > 0$. The final factor in this formula determines the maximum acceptable user reward without resulting in

a self-loop. By ensuring that the magnitude of $bias(s,a)$ remains below k^{-1}, we can guarantee that erroneous self-loops do not form.

With negative user rewards, however, it may be determined [10] that erroneous loops do not form unless a cycle of state-action pairs with utilities of zero are formed. By ensuring that the *Q*-value of any state-action pair never drops to zero, we can guarantee that negative user rewards do not create erroneous loops. An example of such a mapping is:

$$R_\upsilon(s,a) \leftarrow \max\left\{\begin{array}{c} bias(s,a)\cdot k\cdot\left((1-\gamma)Q(s,a)-R_\tau(s,a)\right), \\ -\max_{b\in A-\{a\}} Q(s,b)-R_\tau(s,a)+\varepsilon \end{array}\right\}$$

where ε is a small positive number. Similarly, mappings that are non-linear in the *bias* function, such as the mapping used to generate the results of Section 3, may also be defined.

2.2 Learning with Composite Rewards

We describe an offline *Q*-learning algorithm that makes use of a probabilistic model of the environment and a composite reward function. *Q*-values of state-action pairs are updated according to their probabilities of success. The updated *Q*-value of a successful state-action pair (s,a) is:

$$\delta_{succ}(s,a) \leftarrow R(s,a)+\gamma\cdot Q(s', \arg\max_{b\in A} Q(s',b))$$

where s' represents the state to which the agent transfers with the successful execution of action *a*. Under the simplifying assumption that unsuccessful actions do not change the current state of the agent, the updated *Q*-value of a failed state-action pair (s,a) is:

$$\delta_{fail} \leftarrow \gamma\cdot Q(s,a)$$

The resultant *Q*-value of the state-action pair (s,a) is:

$$Q(s,a) \leftarrow P(s'\mid s,a)\cdot\delta_{succ}(s,a) + \left(1-P(s'\mid s,a)\right)\cdot\delta_{fail}(s,a)$$

The algorithm is shown in Table 1. The iterative computation of the user reward component R_υ in step 1(*b*) of the algorithm may be better understood from the following discussion.

Consider a state-action pair (s,a) with a 100% probability of success. There is a direct dependence between $Q(s,a)$ and $R_\upsilon(s,a)$—the value assigned to (s,a) increases with the user reward assigned to it, while the user reward itself is proportional to the difference in values of the best next state and (s,a). Initially, as the user reward is zero, $Q(s,a)$ is at its natural discounted maximum. When (s,a) is recommended externally, a user reward proportional in magnitude to the difference in values of the best next state and the state-action pair itself is assigned. This user reward causes an increase in $Q(s,a)$. This then causes a decreased difference in the value of the best next state and (s,a). This would resultantly decrease the user reward, and so on. While this causal chain does eventually converge to the desired user reward and *Q*-value, the oscillations may cause abnormal behavior in the interim, such as the selection of a sub-optimal action. An incremental user reward update scheme would therefore seem essential for oscillation-free convergence of the user reward function.

Table 1. Learning with Composite Rewards

LEARN-WITH-COMPOSITE-REWARDS:

Initialize:

$i \leftarrow 0$

For each state-action pair (s,a), do $R_\upsilon^i(s,a) \leftarrow 0$

Repeat forever:

$i \leftarrow i+1$

For each state-action pair (s,a), do:

1. Determine the composite reward $R(s,a)$:
   - *a.* Compute $R_\upsilon(s,a)$ according to $bias(s,a)$
   - *b.* $R_\upsilon^i(s,a) \leftarrow R_\upsilon^{i-1}(s,a)+\alpha_\upsilon\cdot\left(R_\upsilon(s,a)-R_\upsilon^{i-1}(s,a)\right)$
   - *c.* $R(s,a) \leftarrow R_\tau(s,a)+R_\upsilon^i(s,a)$
2. Compute the components of $Q(s,a)$:
   - *a.* $\delta_{succ}(s,a) \leftarrow R(s,a)+\gamma\cdot Q(s', \arg\max_{b\in A} Q(s',b))$
   - *b.* $\delta_{fail} \leftarrow \gamma\cdot Q(s,a)$
3. $Q(s,a) \leftarrow P(s'\mid s,a)\cdot\delta_{succ}(s,a) + \left(1-P(s'\mid s,a)\right)\cdot\delta_{fail}(s,a)$

This notion is further illustrated in Figure 2. The red plot is due to the direct assignment of user rewards, while the oscillation-free blue plot is due to the iterative update scheme. Its monotonous increase prevents abnormal user reward assignments prior to convergence. This aspect is captured by the algorithm in table 1 through the user reward update rate α_υ. An assignment of $0 \le \alpha_\upsilon < 1$ leads to a weighted sum of the old and new estimates with a weight ratio of $\alpha_\upsilon : 1-\alpha_\upsilon$.

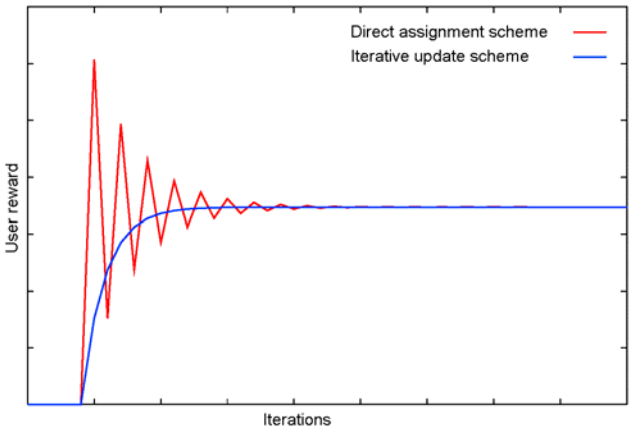

Figure 2. Comparison of direct assignment and iterative update schemes for user rewards

3. Robot Navigation

Learning with composite rewards is a general learning algorithm applicable to a wide variety of tasks for which task reward functions can be correctly defined. Complex tasks can be divided into a set of sub-tasks, each of which may be modeled by a task reward function, and all of which are linked together in a connected graph.

To demonstrate the functionality of this approach to learning from reinforcements and advice, we consider the task of mobile robot navigation. Activmedia Robotics' Pioneer 2, shown in Figure 3, was used to learn to navigate an environment in order to reach a functional goal, such as its proximity to a specific geometrical location or a particular visually identifiable feature. The environment consists of a set V of *via-points* on a set W of maps. Via-points represent geometrical locations that are of interest to the learning task, reducing the state space to be considered to an important few. The maps are linked together through a subset of V representing the *linkage via-points*, which are via-points that are present on two neighboring maps and serve as links when traversing between maps.

The size of a map is an important design criterion. Specifically, it is related to the control strategy used for inter-via-point navigation (represented by the set of actions). This work uses harmonic potential-based motion controllers [2, 3] for this purpose. Advantages of a harmonic path planner include its computation of a path (called a *streamline*) that avoids obstacles and that is suitable for wheeled mobile robots. A typical implementation assigns a high potential to obstacles and a low potential to the goal; the robot follows the path of minimum potential in reaching the goal. To allow for real-time updates of the grid potentials due to the presence of dynamic obstacles, maps were empirically determined to be 50×50 grids of square cells.

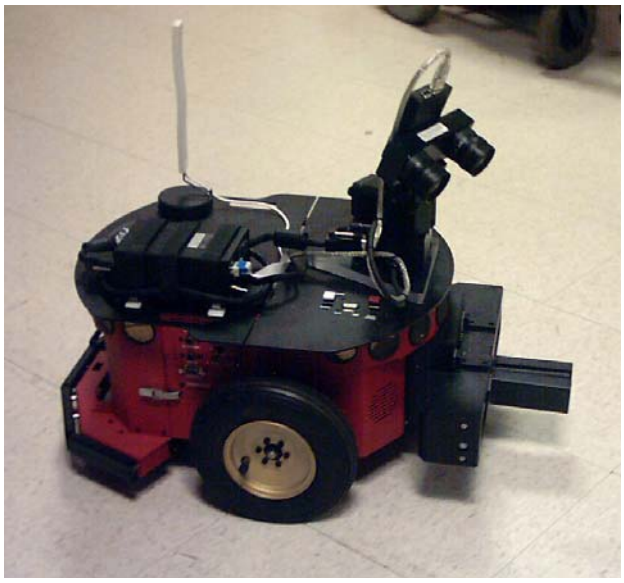

Figure 3. Activmedia Robotics' Pioneer 2 mobile robot

In addition to via-points, visually identifiable features also contribute to the state space. For this work, blobs of three colors—red, blue and green—may be distinctly identified by the robot.

Figure 4 shows a sample environment consisting of four maps for a grid of 100×100 cells. The via-points and visual features are shown linked together to identify the probabilistic model over which learning takes place. Three types of visual features are used in this environment, identified by red, blue and green objects.

3.1 Modifying the Control Policy

For the experiments presented in this paper, a user reward transformation that is non-linear in the *bias* function is used. For positive biases, the user reward is:

$$R_U(s,a) \leftarrow \frac{bias(s,a)}{1+bias(s,a)}\left((1-\gamma)\cdot Q(s,a) - R_\tau(s,a)\right)$$

The first factor is always less than one, and as a result, the user reward never exceeds the maximum allowable user reward as determined by the second factor. Multiple recommendations of the same state-action pair increase the first factor closer to one.

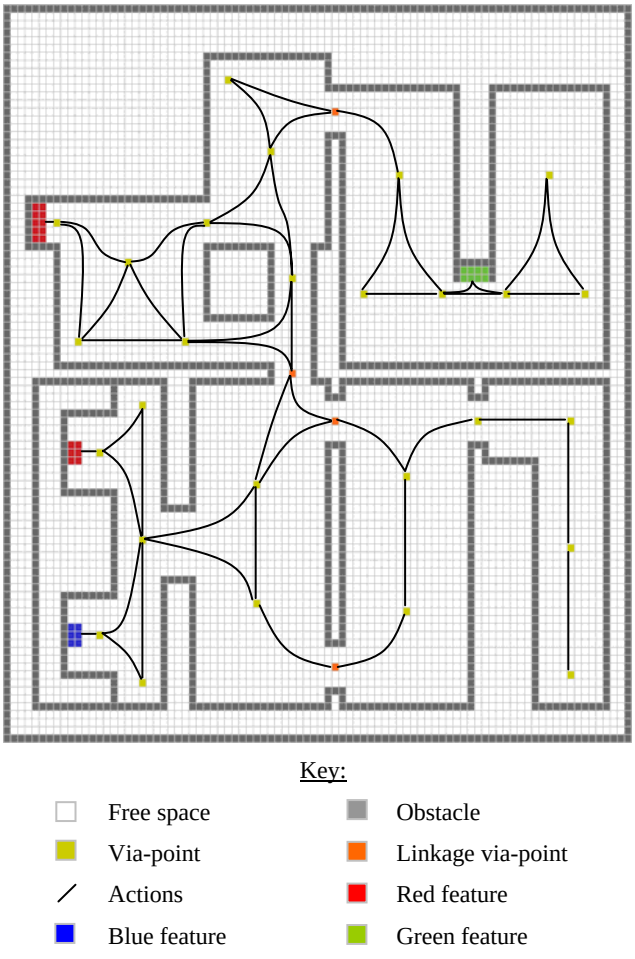

Figure 4. An environment for the task of robot navigation

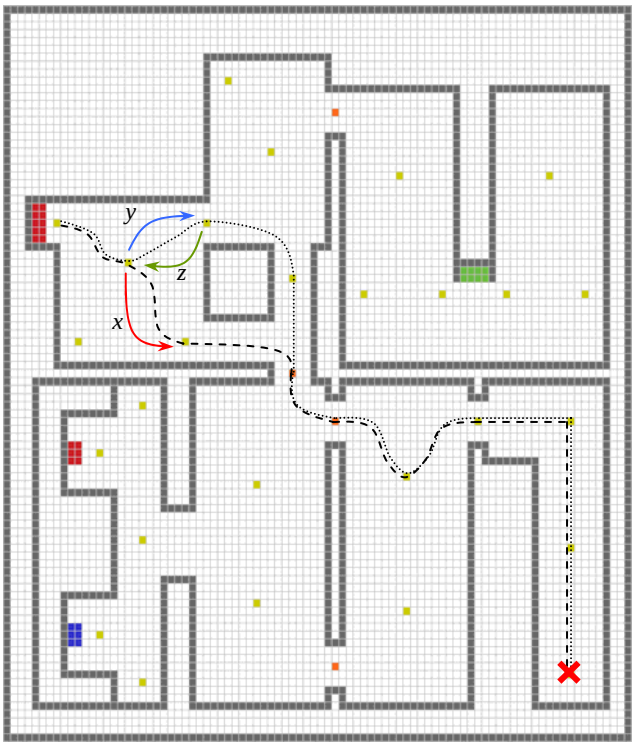

Figure 5. Modifications in the control policy due to advice

For negative biases, the user reward function, together with the lower bound to prevent the values from dropping to zero, is:

$$R_{\upsilon}(s,a) \leftarrow \max\left\{\begin{array}{c} \frac{1}{10} bias(s,a), \\ -\max\limits_{b \in A-\{a\}} Q(s,b) - R_{\tau}(s,a) + \varepsilon \end{array}\right\}$$

where ε is a small positive number.

The control policy may be modified by both recommending and prohibiting states and state-action pairs. Figure 5 presents the results of recommending a sub-optimal state-action pair so as to include it in the control policy. We consider the task of reaching the via-point marked by the red ×. The initial control policy is shown as a dashed line, and includes action *x* (shown in red). When action *y* (shown in blue) is recommended, it is included in the new control policy as shown by the dotted line. The changes in the user reward and utility functions are shown in Figure 6 and Figure 7 respectively.

3.2 Prevention of Loops

The prevention of loops is ensured through the upper and lower bounds on the user reward component. To demonstrate the manner in which loops are prevented, we recommend a set of actions that together form a loop. As illustrated in Figure 5, the actions *y* and *z* together constitute a loop. When action *z* is also recommended, the agent avoids entering the loop and instead selects the sequence of actions that were used by the initial unaltered control policy (shown as a dashed line). The *Q*-value of

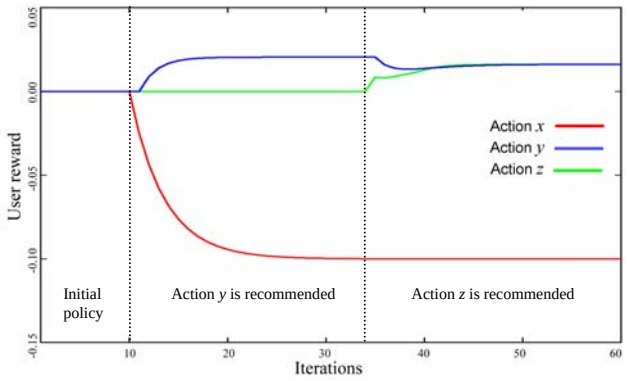

Figure 6. Changes in the user reward function with advice

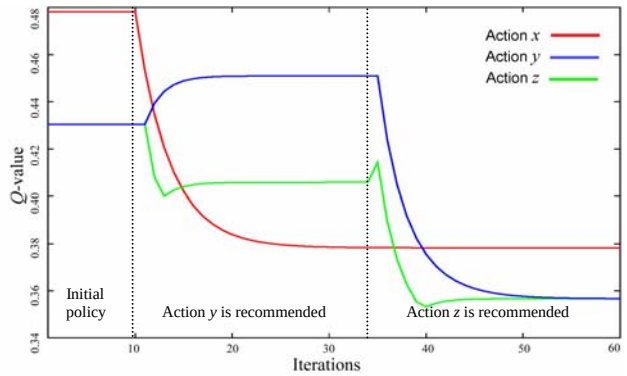

Figure 7. Changes in the utility function with advice

action *y* drops below that of action *x*, and resultantly, action *x* is preferred over action *y*. This is shown in the utility function plot of Figure 7.

4. Conclusions

The automation of tasks through robots that learn form their interaction with the environment is applicable to various domains. By providing such robots with the ability to perform with variable autonomy, domain experts can further refine the autonomously learned policies to incorporate environmental or task-based information to improve efficiency, reduce learning time and/or fine-tune the policies.

The automation of tasks that are repeatedly encountered by the general end-user poses restrictions on the design of such systems. Operational safety must be maintained in its operation by filtering out harmful or prohibitively expensive user commands. Further, the implications of user commands must be analyzed to verify their utility. While various measures of utility may be used, probably most relevant amongst them is task completion. The prevention of spurious loops in the control policy through the bounding of user rewards provides the mechanism with this property.

More generally, learning with composite rewards allows external advice at different levels of abstraction to be integrated with reinforcement learning techniques. The

filtering of the advice in the formation of the user reward structure provides an additional layer of protection to the task definition. This prevents advice from redefining the task, and prevents the formation of erroneous control structures. Furthermore, strategically provided advice can be used to accelerate the learning process, while incorrect advice is ultimately ignored, as its effects diminish over time.

5. Acknowledgements

This work was supported in part by NSF ITR-0121297 and EIA-0203499.

References:

[1] J.A. Clouse, Learning from an Automated Training Agent. In *G. Weiβ, S. Sen*, Adaptation and Learning in Multiagent Systems, Springer Verlag, Berlin, 1996.

[2] C.I. Connolly and R.A. Grupen, Harmonic Control. In *Proceedings of the International Symposium on Intelligent Control*, 1992.

[3] C.I. Connolly and R.A. Grupen. Applications of Harmonic Functions to Robotics. In *Proceedings of the International Symposium on Intelligent Control*. 1992.

[4] F. Hayes-Roth, P. Klahr, and D.J. Mostow, Advice-taking and knowledge refinement: An iterative view of skill acquisition. Technical Report, Rand Corporation, 1980.

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

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

[7] R. Maclin and J.W. Shavlik, Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994, MIT Press, 694–699.

[8] J. McCarthy, Programs with common sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, volume 1, London, 1958, 77–84.

[9] A.Y. Ng, D. Harada, and S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping. In *Machine Learning: Proceedings of the Sixteenth International Conference*, 1999, 278-287.

[10] V.N. Papudesi, Integrating Advice with Reinforcement Learning. M.S. thesis, Department of Computer Science, Univ. of Texas at Arlington, 2002.

[11] J. Randløv, and P. Alstrøm, Learning to Drive a Bicycle using Reinforcement Learning and Shaping. In *Machine Learning: Proceedings of the Fifteenth International Conference*. MIT Press, 1998, 463-471.

[12] C.J.C.H. Watkins, Learning from delayed rewards. PhD thesis, Psychology Department, Univ. of Cambridge, 1989.