Learning Policies for Embodied Virtual Agents Through Demonstration

Jonathan Dinerstein
DreamWorks Animation
jondinerstein@yahoo.com

Parris K. Egbert **Dan Ventura**
CS Department, Brigham Young University
{egbert,ventura}@cs.byu.edu

Abstract

Although many powerful AI and machine learning techniques exist, it remains difficult to quickly create AI for embodied virtual agents that produces visually lifelike behavior. This is important for applications (e.g., games, simulators, interactive displays) where an agent must behave in a manner that appears human-like. We present a novel technique for learning reactive policies that mimic demonstrated human behavior. The user demonstrates the desired behavior by dictating the agent's actions during an interactive animation. Later, when the agent is to behave autonomously, the recorded data is generalized to form a continuous state-to-action mapping. Combined with an appropriate animation algorithm (e.g., motion capture), the learned policies realize stylized and natural-looking agent behavior. We empirically demonstrate the efficacy of our technique for quickly producing policies which result in lifelike virtual agent behavior.

1 Introduction

An *Embodied Virtual Agent*, or EVA, is a software agent with a body represented through computer graphics [**?**; **?**]. If given sufficient intelligence, these characters can animate themselves by choosing actions to perform (where each action is a motion). Applications of these agents include computer games, computer animation, virtual reality, training simulators, and so forth.

Despite the success of EVAs in certain domains, some important arguments have been brought against current techniques. In particular, programming the agent AI is difficult and time-consuming, and it is challenging to achieve natural-looking behavior. We address these issues by learning a policy for controlling the agent's behavior that is:

1. **Scalable** — The policy learning must scale to problems with continuous and high-dimensional state spaces.
2. **Aesthetic** — The virtual agent behavior must appear natural and visually pleasing.
3. **Simple** — The policy learning must be directable by a non-technical user.

Reinforcement learning (RL) is a powerful scheme and an obvious choice for policy learning. However, it does not meet our requirements — RL does not always scale well to continuous state spaces of high dimensionality (which are common for EVAs), and it is challenging to encode aesthetic goals into a reward structure. We take a different approach.

We present a technique for automatic learning of a policy by mimicking demonstrated human behavior. This technique not only provides a natural and simple method for the construction of a policy, but it also allows an animator (or other non-technical person) to be intimately involved in the construction of the policy. We learn a reactive policy rather than more advanced behavior because this learning is simple and nearly automatic, thereby allowing for AI creation by non-technical users. This approach is empirically shown to be effective at quickly learning useful policies whose behavior appears natural in continuous state (and action) spaces of high dimensionality.

2 Related Work

Designing and developing embodied virtual agents is fundamentally a multi-disciplinary problem [**?**; **?**]. Indeed the study of EVAs overlaps with several scientific fields, including artificial intelligence, computer graphics, computer-human interfaces, etc.

The computer graphics community has been interested in EVAs since Reynold's seminal work on flocking behavior [**?**]. One of the most well-known examples in film is the *Lord of the Rings* trilogy [**?**], depicting epic-scale battle scenes of humanoids. Such large-scale animation would be implausible if the characters were animated manually. Several notable techniques for constructing EVAs have been developed (e.g., [**?**]) but the explicit programming of agent AI remains a difficult task.

Game-oriented EVA research includes the development of action selection and learning architectures [**?**]. Computational models of emotion have also been proposed [**?**]. Recent work has examined rapid behavior adaption such that an EVA can better interact with a given human user [**?**; **?**]. In other work, a scheme has been developed whereby a non-embodied agent in a text-based computer game learns through interaction with multiple users [**?**].

As discussed in the introduction, designing and programming EVA AI is often challenging. Several authors have ad-

dressed this through reinforcement learning (a survey is given in [**?**]). RL is a natural choice since it has long been used for agent policy learning. However, RL does not meet our requirements in this paper. For example, it is quite challenging to achieve natural-looking behavior since these aesthetic goals must be integrated into the fitness function. Also, many EVAs live in continuous virtual worlds and thus may require continuous state and/or action spaces, such as flocking EVAs [**?**]. Traditional RL is intractible for large or continuous state/action spaces, though recent research has begun to address this problem [**?**; **?**]. Nevertheless, the learning is still computationally expensive and limited. Our technique learns stylized EVA behaviors quickly as empirically shown later.

The agents and robotics communities have long recognized the need for simplified programming of agent AI. There has been some interesting work performed in *programming by demonstration*, such as [**?**; **?**; **?**; **?**; **?**; **?**], where an agent is instructed through demonstrations by a user. Our technique fits in this category but is specifically designed for EVAs trained by non-technical users. In contrast to these existing techniques, our approach does not require significant effort from a programmer and domain expert before demonstration can be performed. The existing technique that is most related to our work is "Learning to Fly" [**?**]. However, our technique is unique in many aspects, in particular because it performs conflict elimination, can operate in continuous action spaces, and can be operated by non-technical users. There has been some previous work in conflict elimination through clustering [**?**], but that approach quantizes the training examples and culls many potentially useful cases.

Contribution: We present a novel scheme for programming EVA behavior through demonstration. While many of the components of our scheme are pre-existing, our overall system and application are unique. Moreover, some notable and critical components of our scheme are novel. Specific contributions made in this paper include:

- The application of agent programming-by-demonstration concepts to policy learning for humanoid EVAs.
- An intuitive interface for non-technical users to quickly create stylized and natural-looking EVA behavior.
- A novel conflict elimination method (previous techniques in agents/robotics have usually ignored or blended conflicts).

3 EVA Programming by Demonstration

Learning an EVA policy through demonstration involves the following steps (see Figure **??**):

1. *Train:*
   (a) Observe and record state-action pairs.
   (b) Eliminate conflicts.
   (c) Generalize state-action pairs into a policy μ.
2. *Autonomous behavior:*
   (a) Use μ to compute decisions for the agent, once per fixed time step Δt.

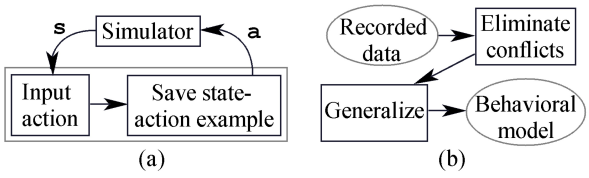


Figure 1: The workflow of our technique. (a) The simulator, which drives the animation, provides a current state $\mathbf{s}$. This is visualized for the demonstrator. The demonstrator responds with an action $\mathbf{a}$, which is recorded and used to update the simulation. (b) The state-action examples are processed to eliminate conflicts and then generalized into a continuous policy function. This policy is used online to control the agent.

This is a process of learning to approximate intelligent decision making from human example. We formulate decision making as a *policy*:

$$\mu : \mathbf{s} \rightarrow \mathbf{a} \tag{1}$$

where $\mathbf{s} \in \mathbb{R}^n$ is a compact representation of the current state of the character and its world, and $\mathbf{a} \in \mathbb{R}^m$ is the action chosen to perform. Each component of $\mathbf{s}$ is a salient feature defining some important aspect of the current state. If the character has a finite repertoire of possible actions, then $\mathbf{a}$ is quantized.

A state-action pair is denoted $\langle \mathbf{s}, \mathbf{a} \rangle$. The demonstrator's behavior is sampled at a fixed time step Δt, which is equal to the rate at which the character will choose actions when it is autonomous. Thus the observed behavior is a set of discrete cases, $B = \{\langle \mathbf{s}, \mathbf{a} \rangle_1, \ \ldots, \ \langle \mathbf{s}, \mathbf{a} \rangle_q\}$. Each pair represents one case of the target behavior. There is no ordering of the pairs in the set. We construct μ by generalizing these cases. Specifically, to ensure that the character's behavior is smooth and aesthetically pleasing, we construct μ by interpolating the actions associated with these cases. Because $\mathbf{s} \in \mathbb{R}^n$ and $\mathbf{a} \in \mathbb{R}^m$, we can theoretically use any continuous real-vector-valued interpolation scheme. In our implementation, we either use ε-regression SVM or continuous k-nearest neighbor, as detailed later. If the action set is symbolic, we use voting k-nearest neighbor.

For generalization of cases to succeed, it is critical that the state and action spaces be organized by the programmer such that similar states usually map to similar actions. In other words, $\|\mathbf{s}_i - \mathbf{s}_j\| < \alpha \Rightarrow \|\mathbf{a}_i - \mathbf{a}_j\| < \beta$, where α and β are small scalar thresholds and $\|\cdot\|$ is the Euclidean metric. Certainly, this constraint need not always hold, but the smoother the mapping, the more accurate the learned policy μ will be.

3.1 Training

In training mode, both the demonstrator and programmer need to be involved. First, the programmer integrates our technique into an existing (but thus far "brainless") character. This integration involves designing the state and action spaces such that μ will be learnable. Once integration is complete, the demonstrator is free to create policies for the character at will. In fact, the creation of multiple policies for one agent may be interesting to achieve different stylized behaviors, etc.

Demonstration proceeds as follows. The character and its virtual world are visualized in real-time, and the demonstrator has interactive control over the actions of the character. Note that the continuous, real-time presentation of state information to the demonstrator is critical to making the character training process as natural as possible, as this is analogous to how humans naturally perceive the real world. As the simulation-visualization of the character and its world proceeds in real-time, the demonstrator supplies the character with the actions it is to perform. This information is saved in the form of state-action pairs. Once enough state-action examples have been collected, all conflicting examples are automatically eliminated, as described below. We now have a discrete but representative sampling of the entire policy function μ. Moreover, because the demonstrator has had control of the character, she has forced it into regions of the state space of interest — therefore the density of the sampling corresponds to the importance of each region of the state space.

Elimination of conflicting examples is important because human behavior is not always deterministic, and therefore some examples will likely conflict. This is an important issue, because the machine learning schemes we utilize will "average" conflicting examples. This can result in unrealistic or unintelligent-looking behavior. Note that conflict resolution might also be possible by augmenting the state with context (e.g. the last n actions performed). However, this increase in the state-space dimensionality will only make the learning μ more challenging and require additional training examples. Thus we have designed our technique to directly eliminate conflicts.

Conflicting examples are formally defined as:

$$\text{if } \|\mathbf{s}_i - \mathbf{s}_j\| < \nu \text{ and } \|\mathbf{a}_i - \mathbf{a}_j\| > \upsilon, \text{ then } \texttt{conflict}, \quad (2)$$

where ν and υ are scalar thresholds. To eliminate conflicts, we cannot arbitrarily delete cases involved in a conflict — this can lead to high frequencies in the policy. Our goal is to remove those examples that represent high frequencies.

Pseudo-code for our conflict elimination technique is given in Figure **??**. To complement this pseudo-code, we now describe our technique. In brief, each state-action pair is tested in turn to determine whether it is an outlier. First, the l neighbors (according to state) of the current example are found and their median action $\mathbf{a}_{vm}$ is computed, using the following vector-median method [**?**]:

$$\mathbf{a}_{vm} = \arg \min_{\mathbf{a}_i \in \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l\}} \left(\sum_{j=1}^{l} \|\mathbf{a}_i - \mathbf{a}_j\| \right) \quad (3)$$

In other words, $\mathbf{a}_{vm}$ is equal to the action of the neighbor that is the closest to all other actions of the neighbors according to the Euclidean metric. Finally, if $\|\mathbf{a}_{vm} - \mathbf{a}'\| > \eta$, then the case is an outlier and is marked for deletion. Marked cases are retained for testing the other cases, and then are all deleted as a batch at the conclusion of the conflict elimination algorithm. In our experiments, we have found that it works well to use $l = 5$. We use a threshold η of about 10% of the possible range of component values in $\mathbf{a}$.

For each recorded pair $\langle \mathbf{s}, \mathbf{a} \rangle'$ ...
  Find l closest neighbors of $\langle \mathbf{s}, \mathbf{a} \rangle'$
  Compute median action $\mathbf{a}_{vm}$ of l neighbors using Eq. **??**
  if ($\|\mathbf{a}_{vm} - \mathbf{a}'\| > \eta$)
    Mark $\langle \mathbf{s}, \mathbf{a} \rangle'$
Delete all marked pairs

Figure 2: Conflict elimination algorithm.

3.2 Autonomous Behavior

Once an adequate set of state-action pairs has been collected, we must construct the continuous policy, $\mu : \mathbf{s} \rightarrow \mathbf{a}$. We do this through one of two popular machine learning techniques: ε-regression SVM [**?**] or k-nearest neighbor [**?**] (or through a classification scheme if the actions are symbolic). We have chosen these two techniques because they are powerful and well-established, yet have contrasting strengths and weaknesses in our application.

The primary strength of continuous k-nearest neighbor (k-nn) is that there are strong guarantees about its accuracy, as it merely interpolates local cases (e.g., it can robustly handle non-smooth mappings, and the outputs will be within the convex hull of the k local cases). Therefore we use k-nn whenever the demonstrated policy is rough. However, k-nn does have a large memory footprint ($\sim$1 MB per policy), and more state-action cases are required than with SVM due to weaker generalization.

The support vector machine (SVM) is an interesting alternative to k-nn. It is a compact and global technique. As a result, it performs powerful generalization, but can struggle with highly non-smooth mappings. We have found SVM to be useful when μ is somewhat smooth, especially when it is C^0 or C^1 continuous. We use the popular ε-regression scheme with a Gaussian kernel to perform function approximation.

4 Experimental Results

These experiments were designed to cover, in a general fashion, most of the major distinguishing aspects of popular uses of embodied virtual agents. Between our favorable experimental results, and the results from the agents/robotics literature on programming-by-demonstration (see the related work section), we have some empirical evidence that our scheme is a viable tool for creating some popular types of EVA policies.

The results we achieved in our experiments are summarized in Table **??**. Comparisons against reinforcement learning, A*, and PEGASUS are summarized in Table **??** and discussed in more detail later (while we do not take an RL approach in our technique, we believe it is valuable to compare against RL since it such a popular approach). The results of an informal user case study are reported in Table **??**. Videos of the animation results are available from:

`http://rivit.cs.byu.edu/a3dg/publications.php`

4.1 Spaceship Pilot

In our first experiment, the virtual agent is a spaceship pilot (see Figure **??**). The pilot's task is to maneuver the spaceship through random asteroid fields, flying from one end of

Pre-	Number of state-action pairs	6,000
	Time to demonstrate all pairs	8 min.
	Time to eliminate conflicts	16 sec.
***k*-nn**	Compute target function values	13 μsec.
	Storage requirements	$\sim$1 MB
	Total decision-making time to cross field	7.04 msec.
	Total animation time to cross field	36.1 sec.
SVM	Time to train ε-regression SVM	2.5 min.
	Compute new target function values	2 μsec.
	Storage requirements	$\sim$10 KB
	Total decision-making time to cross field	1.13 msec.
	Total animation time to cross field	37.7 sec.

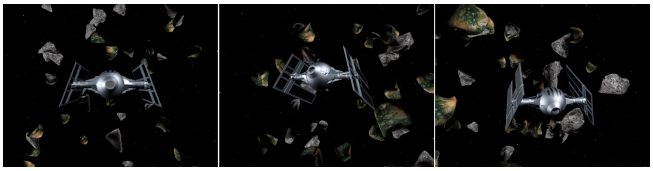
Table 1: Summary of average usage and performance results in our spaceship pilot case study (with a 1.7 GHz processor, 512 MB RAM, $k \in [2,7]$). Similar results were achieved in other experiments.

the field to the other as quickly as possible with no collisions. The animation runs at 15 frames per second, with an action computed for each frame. The virtual pilot controls the orientation of the spaceship.

μ is formulated as follows. The inputs are the spaceship's current orientation (θ, ϕ), and the separation vector between the spaceship and the two nearest asteroids ($\mathbf{p}_s - \mathbf{p}_{a1}$ and $\mathbf{p}_s - \mathbf{p}_{a2}\rangle$). Thus the complete state vector is $\mathbf{s} = \langle \theta, \phi, \Delta x_1, \Delta y_1, \Delta z_1, \Delta x_2, \Delta y_2, \Delta z_2 \rangle$. There are two outputs, which determined the change in the spaceship's orientation: $\mathbf{a} = \langle \Delta\theta, \Delta\phi \rangle$.

We achieved good results in this experiment, as shown in Table **??** and in the accompanying video. The SVM and *k*-nn policies worked well for all random asteroid fields we tried, and the animation was natural-looking, intelligent, and aesthetically pleasing (verified in an informal user case study).

To gain a point of reference and to illuminate the interesting computational properties of our technique, we analyzed learning the policy through reinforcement learning (Table **??**). This analysis helps validate why a non-RL approach is important in fulfilling our goals. It is straightforward to formulate our case study as an MDP. The state and action spaces have already been defined. We have a complete model of the deterministic environment (needed for animation), the initial state is a position on one side of the asteroid field, and the reward structure is positive for safely crossing the field and negative for crashing. However, even using state-of-the-art table-based techniques for solving large MDP's [**?**], it is currently implausible to perform RL with a continuous state space of dimensionality greater than 6 (this size even requires a supercomputer). Our MDP state space dimensionality is 9. Even gross discretization is intractable: e.g., for 10 divisions per axis, 10^9 states. The best result reported in [**?**] is a 4-dimensional state space discretized into approximately $10^{7.85}$ states. We also experimented with learning a policy through function-approximation-based RL using a multi-layer neural net, but this failed (the learned policies were very poor). This type of learning is typically considered most stable when employing a linear function approximator, but this can limit the learning [**?**; **?**]. Thus, for our application, demonstration-based learning is appealing since it is effective



in high-dimensional, continuous state and action spaces.

We also tested a non-standard form of RL designed specifically for continuous state spaces: PEGASUS [**?**; **?**]. In PEGASUS, a neural network learns a policy through hill climbing. Specifically, a small set of finite-horizon scenarios is used to evaluate a policy and determine the fitness landscape. Unlike traditional RL, PEGASUS succeeded in learning an effective policy in our case study (see Table **??**). Interestingly, the performance of the policies learned through our scheme and PEGASUS are similar. However, PEGASUS learning was far slower than our technique (hours versus minutes). Moreover, we found it extremely difficult and technical to tune the PEGASUS reward structure for aesthetics. Thus, while PEGASUS is a powerful and useful technique, it does not fulfill the requirement given in the introduction of providing non-technical users with an intuitive and robust interface for programming EVA AI. As a result, RL does not appear to be a good fit for our application.

We also analyzed explicit action selection through A* (Table **??**). A* has empirically proven successful, but requires significantly more CPU than our policies to make decisions, sometimes produces behavior that does not appear natural, and requires the programmer to develop an admissible heuristic.

4.2 Crowd of Human Characters

In our next experiment we created policies to control groups of human characters (see Figure **??**). The characters milled about, then a boulder fell from the sky and they ran away. The policies controlled the decision making of the characters (i.e., "turn left," "walk forward," etc), while the nuts-and-bolts animation was carried out by a traditional skeletal motion system. Thus the policy only specified a small set of discrete actions (the real-valued policy output was quantized to be discrete). We created several crowd animations. In each animation, all characters used the same policy, showing the variety of behavior that can be achieved. Note that each policy we constructed only required a few minutes to capture all necessary state-action examples, and only a few thousand examples were required for use online.

To train agents to interact in a crowd we performed several brief demonstration iterations, randomly placing some static characters in a scene. The human trainer then guided the learning agent through the scene. Thus the agent learned how to behave in response to other agents around it in arbitrary positions.

We created two policies. The first was used before the boulder impact, and the second afterwards. In the first policy (pre-boulder), the environment was discretized into a 2D grid. $\mathbf{s}$ was formulated as the current orientation of the character, and the separation between it and the next three closest characters. Decision making was computed at a rate Δt

RL	Storage requirements	∼8 G
	Time to learn policy	∼1 week
PEG.	Storage requirements	∼5 KB
	Time to learn policy	∼2.3 hrs.
	Time to select actions	2 msec.
	Total decision-making time to cross field	6.95 sec.
	Total animation time to cross field	36.6 sec.
A*	Average storage requirement	∼8 M
	Max storage requirement	∼370 M
	Time to select actions	0.2 sec.
	Total decision-making time to cross field	104.1 sec.
	Total animation time to cross field	34.7 sec.

Table 2: Summary of average results using reinforcement learning, PEGASUS, and A* in the spaceship pilot case study. Compare to Table **??**. The purpose of this comparison to help validate why we have selected demonstration-based learning for our problem domain. Reinforcement learning takes a prohibitive amount of time with static or adaptive table-based learning, and function-approximation-based RL did not learn effective policies in our experiments. PEGASUS successfully learns a useful policy in our case study, but requires significantly more learning time than our approach, and it is difficult to tune the learning for aesthetics. A* takes far more CPU time to make decisions than the policy learned through our technique, and requires an admissible heuristic. Thus, from a computational standpoint, our technique is an interesting alternative to these traditional schemes for agent action selection.

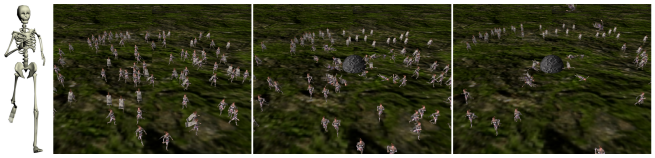


Figure 3: Virtual human crowd test bed.

of about twice per second. After the boulder hit the ground, we switched to another policy (with continuous state/action spaces) which simply directed the characters to run straight away from the boulder. This modularity made the policy learning easier.

5 Discussion

Our technique has a number of strong points. As has been shown empirically, policies can be learned rapidly. The aesthetic value of these policies was verified through informal user case studies (Table **??**). Also, its use is natural and straightforward, it is applicable to non-technical users, and there is empirical evidence that it is a viable tool for current uses of EVAs (e.g., computer games and animation). In addition, our approach has a nearly fixed online execution time, which is a useful feature for interactive agents.

Improvements to our approach that are the subject of ongoing research include the following. First, note that a demonstrator's choice of actions may not make the character traverse all regions of the state space, leaving gaps in μ. This problem

Test bed	*User*	*Instruct. time*	*Dem. time*
Space ship	Author	N.A.	7 min.
	Artist	3 min.	9.5 min.
	Game player	1 min.	8.5 min.
Crowd	Author	N.A.	12 min.
	Designer	4 min.	13 min.
	Student	3 min.	13.5 min.

Table 3: Summary of an informal user case study. *Instruct. time* is how long a new user needed to be tutored on the use of our system. Our technique is intuitive (little instruction required) and user-directed policies can be quickly constructed. The users in this study prefered the aesthetics of their demonstrated policies over traditional policies.

can be automatically solved, during training, by periodically forcing the character into these unvisited regions of the state space. Another important issue is perceptual aliasing, which can result in unanticipated character behavior. However, perceptual aliasing can be minimized through effective design of the compact state space.

We have opted for learning standard policies (no task structure or rules, e.g. [**?**]) because it allows for the most work to be done through programming-by-demonstration versus explicit programming. However, the scalability of policies is limited. Our technique could be augmented with a task learning method from the literature and thereby gain additional scalability and planning behavior, but at the cost of greater programmer involvement. A good example is our crowd case study. Pre/post conditions (e.g. [**?**]) for the discrete contexts could be learned, determining when to switch between policies.

Because our current technique learns deterministic policies, there are some interesting behaviors that it cannot learn. For example, complex goal-directed behaviors that must be broken down into sub-tasks. Moreover, our technique is limited to policies that are smooth state-action mappings.

Conflict elimination is important because demonstrator behavior may be non-deterministic, and conflicts may arise due to varying delays in demonstrator response time. These conflicts can result in unintended behavior or temporal aliasing (dithering) in the behavior. Our conflict elimination technique safely removes high frequencies in the state-action pairs, helping make μ a smooth and representative mapping. Also, unlike elimination through clustering [**?**], our method does not quantize or eliminate examples. As an alternative to eliminating conflicts, one may consider allowing disjunctive action rules or incorporating probabilistic actions, perhaps weighted by how often each conflicting action is seen during the demonstration period.

Policies can be incrementally constructed or adapted by adding new state-action pairs. If desired, older examples can be deleted, or conflict elimination can be run to cull less applicable examples.

References

[Angros *et al.*, 2002] R Angros, L Johnson, J Rickel, and A Scholer. Learning domain knowledge for teaching pro-

cedural skills. In *Proceedings of AAMAS'02*, 2002.

[Badler *et al.*, 1999] N Badler, M Palmer, and R Bindiganavale. Animation control for real-time virtual humans. *Communications of the ACM*, 42(8):65–73, 1999.

[Cole *et al.*, 2003] R Cole, S van Vuuren, B Pellom, K Hacioglu, J Ma, and J Movellan. Perceptive animated interfaces: first step toward a new paradigm for human-computer interaction. *Proceedings of the IEEE*, 91(9), 2003.

[Dinerstein and Egbert, 2005] J Dinerstein and P K Egbert. Fast multi-level adaptation for interactive autonomous characters. *ACM Transactions on Graphics*, 24(2):262–288, 2005.

[Dinerstein *et al.*, 2005] J Dinerstein, D Ventura, and P K Egbert. Fast and robust incremental action prediction for interactive agents. *Computational Intelligence*, 21(1):90–110, 2005.

[Dinerstein *et al.*, 2006] J Dinerstein, P K Egbert, and David Cline. Enhancing computer graphics through machine learning: A survey. *The Visual Computer*, 22(12), 2006.

[Duncan, 2002] J Duncan. Ring masters. *Cinefex*, 89:64–131, 2002.

[Friedrich *et al.*, 1996] H. Friedrich, S. Munch, R. Dillmann, S. Bocionek, and M. Sassin. Robot programming by demonstration (RPD): supporting the induction by human interaction. *Machine Learning*, 23(2-3):163–189, 1996.

[Gratch and Marsella, 2005] J Gratch and S Marsella. Evaluating a computational model of emotion. *Autonomous Agents and Multi-Agent Systems*, 11(1):23–43, 2005.

[Gratch *et al.*, 2002] J Gratch, J Rickel, E Andre, J Cassell, E Petajan, and N Badler. Creating interactive virtual humans: some assembly required. *IEEE Intelligent Systems*, pages 54–61, 2002.

[Haykin, 1999] S Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ., 2nd edition, 1999.

[Isbell *et al.*, 2001] C Isbell, C Shelton, M Kearns, S Singh, and P Stone. A social reinforcement learning agent. In *Proceedings of Fifth International Conference on Autonomous Agents*, pages 377–384, 2001.

[Kasper *et al.*, 2001] M Kasper, G Fricke, K Steuernagel, and E von Puttkamer. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems*, 34:153–164, 2001.

[Koschan and Abidi, 2001] A Koschan and M Abidi. A comparison of median filter techniques for noise removal in color images. In *7th Workshop on Color Image Processing*, pages 69–79, 2001.

[Laird, 2001] J Laird. It knows what you're going to do: Adding anticipation to the quakebot. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, 2001.

[Mataric, 2000] M Mataric. Getting humanoids to move and imitate. *IEEE Intelligent Systems*, pages 18–24, July 2000.

[Mitchell, 1997] T Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[Monzani *et al.*, 2001] J Monzani, A Caicedo, and D Thalmann. Integrating behavioural animation techniques. *Computer Graphics Forum*, 20(3), 2001.

[Ng and Jordan, 2000] A Y Ng and M Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.

[Ng *et al.*, 2004] A Y Ng, A Coates, M Diel, V Ganapathi, J Schulte, B Tse, E Berger, and E Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[Nicolescu, 2003] M Nicolescu. *A Framework for Learning From Demonstration, Generalization and Practice in Human-Robot Domains*. PhD thesis, University of Southern California, 2003.

[Pomerleau, 1996] D Pomerleau. Neural network vision for robot driving. In S Nayar and T Poggio, editors, *Early visual learning*, pages 161–181. Oxford University Press, New York, NY, 1996.

[Reynolds, 1987] C Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH 1987*, pages 25–34. ACM Press / ACM SIGGRAPH, 1987.

[Sammut *et al.*, 1992] C Sammut, S Hurst, D Kedzier, and D Michie. Learning to fly. In *Proceedings of Machine Learning*, pages 385–393, 1992.

[Sutton and Barto, 1998] R Sutton and A Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[Tesauro, 1995] G Tesauro. Temporal difference learning in TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[van Lent and Laird, 2001] M van Lent and J Laird. Learning procedural knowledge through observation. In *Proceedings of International Conference On Knowledge Capture*, pages 179–186. ACM Press, 2001.

[Wingate and Seppi, 2005] D Wingate and K D Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.