In *IEEE International Conference on Robotics and Automation* (ICRA 2010), Anchorage, Alaska, May 2010.

Generalized Model Learning for Reinforcement Learning on a Humanoid Robot

Todd Hester, Michael Quinlan, and Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
{todd,mquinlan,pstone}@cs.utexas.edu

***Abstract*— Reinforcement learning (RL) algorithms have long been promising methods for enabling an autonomous robot to improve its behavior on sequential decision-making tasks. The obvious enticement is that the robot should be able to improve its own behavior without the need for detailed step-by-step programming. However, for RL to reach its full potential, the algorithms must be *sample efficient*: they must learn competent behavior from very few real-world trials. From this perspective, model-based methods, which use experiential data more efficiently than model-free approaches, are appealing. But they often require exhaustive exploration to learn an accurate model of the domain. In this paper, we present an algorithm, Reinforcement Learning with Decision Trees (RL-DT), that uses decision trees to learn the model by generalizing the relative effect of actions across states. The agent explores the environment until it believes it has a reasonable policy. The combination of the learning approach with the targeted exploration policy enables fast learning of the model. We compare RL-DT against standard model-free and model-based learning methods, and demonstrate its effectiveness on an Aldebaran Nao humanoid robot scoring goals in a penalty kick scenario.**

I. Introduction

As the tasks that we desire robots to perform become more complex, and as robots become capable of operating autonomously for longer periods of time, we will need to move from hand-coded solutions to having robots learn solutions to problems on their own. Recently, various machine learning techniques have been used to learn control policies and optimize parameters on robots such as helicopters [1], [2] and Sony Aibos [3], [4].

While there has been a great deal of progress in the machine learning community on value-function-based reinforcement learning (RL) methods [5], most successful learning on robots, including all of the above references, has used policy search methods. In value-function-based reinforcement learning, rather than learning a direct mapping from states to actions, the agent learns an intermediate data structure known as a value function that maps states (or state-action pairs) to the expected long term reward. Value-function-based learning methods are appealing because the value-function has well-defined semantics that enable a straightforward representation of the optimal policy, and because of theoretical results guaranteeing the convergence of certain methods to an optimal policy [6], [7].

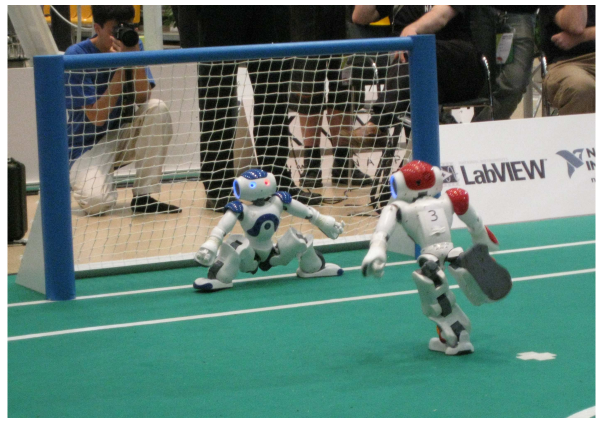

Fig. 1. One of the penalty kicks during the semi-finals of RoboCup 2009.

Value-function methods can themselves be divided into model-free algorithms, such as Q-LEARNING [6], that are computationally cheap, but ignore the dynamics of the world, thus requiring lots of experience; and model-based algorithms, such as R-MAX [7], that learn an explicit domain model and then use it to find the optimal actions via simulation in the model. Model-based reinforcement learning, though computationally more intensive, gives the agent the ability to perform directed exploration of the world so it can learn an accurate model and learn an optimal policy from limited experience.

Model-based methods are thus more appropriate for robot learning, where trials are expensive due to the time involved and/or the wear and tear on the robot. However, existing model-based methods require detailed or exhaustive exploration in order to build the model. In a complex domain, such exploration can itself render the algorithm infeasible from the perspective of sample complexity.

In this paper we describe a model-based RL algorithm, Reinforcement Learning with Decision Trees (RL-DT) that generalizes aggressively during model-learning so as to further limit the number of trials needed for learning. We apply RL-DT on a humanoid robot, the Aldebaran Nao, as it learns to perform a challenging problem from the RoboCup Standard Platform League (SPL): scoring on penalty kicks. We compare RL-DT empirically to both Q-LEARNING and R-MAX, demonstrating that it can successfully be used to learn this task, both in simulation and on a physical humanoid robot. To the best of our knowledge, this first application of RL-DT in a robotic setting is also the first example of successful RL of any kind on the Nao.

II. The Problem

We used reinforcement learning algorithms to train humanoid robots to score penalty kick goals. This scenario takes place in the domain of the RoboCup SPL. RoboCup is an annual robot soccer competition with the goal of developing an autonomous humanoid robot soccer team that can defeat the world champion human team by 2050. Games in the SPL are played between two teams of three Aldebaran Nao [1] robots on a 6 by 4 meter field [8].

Games in the SPL use the V3 RoboCup version of the Aldebaran Nao humanoid robot. The robot is 58 centimeters tall and has 21 degrees of freedom. The robot has two cameras in its head (only one can be used at a time). Computation is performed on the robot using its AMD Geode processor.

When an elimination game in the SPL ends in a tie score, the winner is determined by best of five penalty kicks. In the penalty kick, the defending robot starts in the middle of the goal on the goal line, while the offensive robot starts at mid field. The ball is placed on a white cross located 1.8 meters from the goal. The robot has one minute to walk up to the ball and score a goal.

Penalty kicks are critical to success in the SPL as many of the teams are evenly matched and many games end in a tie and are decided by penalty kicks. At RoboCup 2009 in Graz, Austria, 28 of the 64 games (43.75%) ended in a tie, including all 8 games in the intermediate round and one of the semi-final games (shown in Figure 1) [2]. In addition, our team's US Open Championship win came in penalty kicks over the UPennalizers after a 1-1 tie in regulation. The frequency of games ending in ties makes penalty kicks an important aspect of the games.

Even though most teams employ a stationary goal keeper for the kicks (as in Figure 1), teams rarely scored on penalty kicks as lining up and aiming the ball past the keeper has proved to be particularly difficult. Out of the 9 games that were decided by penalty kicks (the rest were left as a draw), only 3 had goals scored during the best of five penalty kicks. In total, there were only 7 goals scored in 90 penalty kick attempts, resulting in a low scoring percentage of 7.8%.

III. Reinforcement Learning

We adopted the standard Markov Decision Process (MDP) formalism for this work [5]. An MDP consists of a set of states S, a set of actions A, a reward function $R(s,a)$, and a transition function $P(s'|s,a)$. In many domains, the discrete state s is represented by a vector of n discrete state variables $s = \langle x_1, x_2, ..., x_n \rangle$. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s'. The new state s' is determined from the probability distribution $P(s'|s,a)$.

[1] http://www.aldebaran-robotics.com

[2] http://www.tzi.de/spl/bin/view/Website/Results2009

Algorithm 1 RL-DT($RMax, s$)

$A \leftarrow$ Set of Actions
$S \leftarrow$ Set of States
$\forall a \in A : visits(s,a) \leftarrow 0$
loop
  $a \leftarrow \text{argmax}_{a' \in A} Q(s,a')$
  Execute a, obtain reward r, observe state s'
  Increment $visits(s,a)$
  $(P_M, R_M, \text{CH}) \leftarrow$ UPDATE-MODEL(s, a, r, s', S, A)
  $exp \leftarrow$ CHECK-POLICY(P_M, R_M)
  if CH **then**
    COMPUTE-VALUES$(RMax, P_M, R_M, S_M, A, exp)$
  end if
  $s \leftarrow s'$
end loop

The value $Q^*(s,a)$ of a given state-action pair (s,a) is determined by solving the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \qquad (1)$$

where $0 < \gamma < 1$ is the discount factor. The optimal value function Q^* can be found through value iteration by iterating over the Bellman equations until convergence [5]. The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime. The optimal policy π is then as follows:

$$\pi(s) = \text{argmax}_a Q^*(s,a) \qquad (2)$$

Model-free methods update the value of an action towards its true value when taking each action. In contrast, model-based reinforcement learning methods learn a model of the domain by approximating its transition and reward functions and then simulate actions inside their models. In this paper, we compare a model-free method (Q-LEARNING [6]), and two model-based methods (R-MAX [7] and RL-DT [9]) on the task of scoring on a penalty kick.

Q-LEARNING is a typical model-free reinforcement learning method. In it, values are stored for every state-action pair. Upon taking an action, the value of the action is updated closer to its true value through the Bellman equations. The algorithm was run using ϵ-greedy exploration, where the agent takes a random exploration action ϵ percent of the time and takes the optimal action the rest of the time. In our experiments, Q-LEARNING was run with the learning rate $\alpha = 0.3$, $\epsilon = 0.1$, and the action values initialized to 0.

R-MAX is a representative model-based method that explores unknown states to learn its model quickly. It records the number of visits to each state-action in the domain. State-actions with more than M visits are considered *known*, while ones with fewer visits are *unknown* and need to be explored. The agent is driven to explore the unknown state-actions by assuming they have the maximum reward in the domain. Once the model is completely learned, the algorithm can determine the optimal policy through value iteration.

IV. RL-DT

The RL-DT algorithm (shown in Algorithm 1) is a novel algorithm that incorporates generalization into its model learning. In many domains, it is too expensive or time consuming to exhaustively explore every state in the domain. RL-DT was developed with the idea that by generalizing the model to unseen states, the agent can *avoid* exploring some states. Its two main features are that it uses decision trees to generalize the effects of actions across states, and that it has explicit exploration and exploitation modes. The algorithm is described in detail below, and more pseudocode is available in [9].

In a state s, the algorithm takes the action a with the highest action-value, entering a new state s' and receiving a reward r. Then the algorithm updates its models with this new experience through the model learning approach described below. The algorithm decides to explore or exploit based on the value of its policy in the call to CHECK-POLICY. Next, the algorithm re-computes the action-values using value iteration if the model was changed. It then continues executing actions until the end of the experiment or episode.

The algorithm starts out with a poor model of the domain and refines it over time as it explores. It uses a heuristic to determine when it should explore. If the model predicts that the agent can only reach states with rewards $< 40\%$ of the maximum one-step reward in the domain (given to the algorithm ahead of time) from any state, then the agent goes into *exploration* mode, exploring the state-actions with the fewest visits. If it predicts that it can reach state-actions with better rewards, then it goes into *exploitation* mode, where it takes what it believes is the optimal action at each step.

RL-DT's two modes allow it to explore intelligently early, until it finds good rewards, and then exploit that reward. It does not continue exploring after finding this reward, meaning it explores less than R-MAX does. However, this also means that it may stop exploring before it has found the optimal policy. In many domains, it may be too expensive to explore exhaustively and we may prefer that the agent simply finds a reasonable policy quickly, rather than find an optimal policy slowly.

A distinguishing characteristic of RL-DT is the way in which it learns models of the transition and reward functions. RL-DT generalizes the transition and reward functions across states to learn a model of the underlying MDP in as few samples as possible. In many domains, the *relative* transition effects of actions are similar across many states, making it easier to generalize actions' relative effects than their absolute ones. For example, in many gridworld domains, there is an EAST action that usually increases the agent's x variable by 1. It is easier to generalize that this relative effect ($x \leftarrow x + 1$) occurs in many states than the absolute effects ($x \leftarrow 7$). RL-DT takes advantage of this idea by using supervised learning techniques to generalize the relative effects of actions across states when learning its model. This generalization allows it to make predictions about the effects of actions even in states that it has not visited.

We treat the model learning as a supervised learning problem, with the current state and action as the input, and the relative change in state and the reward as the outputs to be predicted. The agent learns models of the transition and reward functions using decision trees. Decision trees were used because they generalize well while still making accurate predictions. The decision trees are an implementation of Quinlan's C4.5 algorithm [10]. The C4.5 algorithm repeatedly splits the dataset on one of the input variables until the instances in each leaf of the tree cannot be split anymore. It chooses the optimal split at each node of the tree based on information gain. The state features used as inputs to the decision trees are treated both as numerical and categorical inputs, meaning both splits of the type $x = 3$ and $x > 3$ are allowed.

A separate decision tree is built to predict the reward and each of the n state variables. The first n trees each make a prediction of the probabilities of the change in the state feature $P(x_i^r|s, a)$, while the last tree predicts the average reward $R(s, a)$. The input to each tree is a vector containing the n state features and action a: $\langle a, s_1, s_2, ..., s_n\rangle$.

After all the trees are updated, they can be used to predict the model of the domain. For a queried state vector s and action a, each tree makes predictions based on the leaf of the tree that matches the input. The first n trees output probabilities for the relative change, x_i^r, of their particular state features. This output $P(x_i^r|s, a)$ is the number of occurrences of x_i^r in the matching leaf of the tree divided by the total number of experiences in that leaf. The predictions $P(x_i^r|s, a)$ for the n state features are combined to create a prediction of probabilities of the relative change of the state $s^r = \langle x_1^r, x_2^r, ..., x_n^r\rangle$. Assuming that each of the state variables transition independently, the probability of the change in state $P(s^r|s, a)$ is the product of the probabilities of each of its n state features:

$$P(s^r|s, a) = \Pi_{i=0}^{n} P(x_i^r|s, a) \tag{3}$$

The relative change in the state, s^r, is added to the current state s_m to get the next state s'. The last tree predicts reward $R(s, a)$ by outputting the average reward in the matching leaf of the tree. The combination of the model of the transition function and reward function make up a complete model of the underlying MDP.

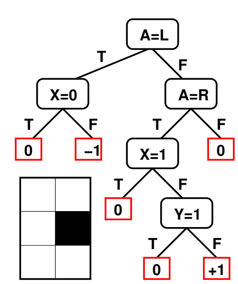

Fig. 2. Example tree predicting the change in the x variable in a gridworld.

Figure 2 shows an example of a decision tree classifying the relative change in the x variable in the gridworld shown in the figure. The tree first splits on the action (if the action was LEFT) and then splits on the x and y variables. In some cases, it can ignore large parts of the state space. For example, when the action is not left or right, the tree predicts a change of 0 in the x variable. In other cases, it makes a prediction that is specific to a single state, such as when it predicts

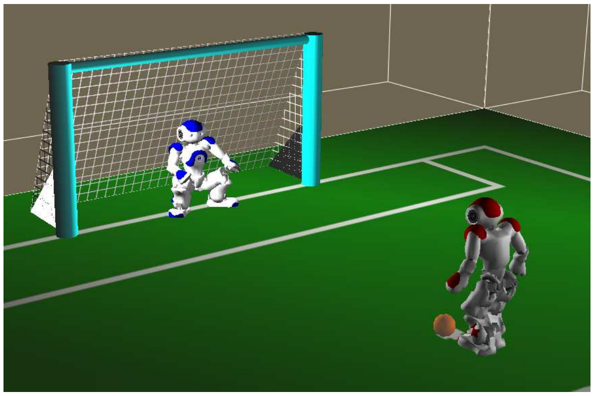

Fig. 3. The experimental setup in the Webots simulator, with the robot learning to aim its kick past the keeper to score on penalty kicks.

a change of 0 for the action right when x is not 1 and y equals 1.

Once the model has been updated, exact value iteration with a discount factor $\gamma = 0.99$ is performed on the approximate model to find a policy. If the agent is in exploration mode, the least visited states are given a reward bonus to drive the agent to explore them. Otherwise, the value iteration calculates the optimal policy according to its learned model.

V. EXPERIMENTAL SETUP

Our goal was to have the robot learn how to score penalty goals against a typical stationary keeper. We set up the experiments with the ball on the penalty mark 1.8 meters from the goal as specified in the SPL penalty kick rules [8]. The robot was placed facing the goal with the center of its feet 15 cm behind the penalty mark. In an actual penalty kick, the robot starts at midfield, but here we are strictly trying to learn to aim the kick and we assume the robot has walked up to ball. The non-stationary keeper was placed in a crouched position in the center of the goal. Every episode began with this exact setup, shown in Figure 3.

For each episode, the agent started our normal kick engine, standing on its right leg and looking down at the ball. The learning algorithm then controlled the free left leg with three available actions: MOVE-OUT, MOVE-IN, and KICK. The robot's state consisted of two state features: the x coordinate of the ball in the robot's camera image and the distance the free foot was shifted out from the robot's hip in millimeters, demonstrated in Figure 4. Each feature was discretized: the ball's image coordinate was discretized into bins of two pixels each, while the leg distance was discretized in 4 millimeter bins. The MOVE-OUT and MOVE-IN actions each moved the leg 4 mm in or out from the robot's body. The agent received a reward of -1 for each action moving the leg in or out and -20 if the action caused the robot to fall over (by shifting its leg too far in either direction). When kicking, the agent received a reward of $+20$ if it scored a goal, and -2 if it did not. The agent's goal was to learn exactly how far to shift its leg before kicking such that it would kick the ball at an angle past the keeper.

Due to the difficulties and time involved in performing learning experiments on the physical robot, we started by performing experiments in the Webots simulator from Cyberbotics [3]. Webots is a robotics simulator that uses the Open Dynamics Engine (ODE) for physics simulation. It simulates all the joints and sensors of the robot, including the camera. After running experiments in the simulator, we ran experiments on the physical robot to validate our results.

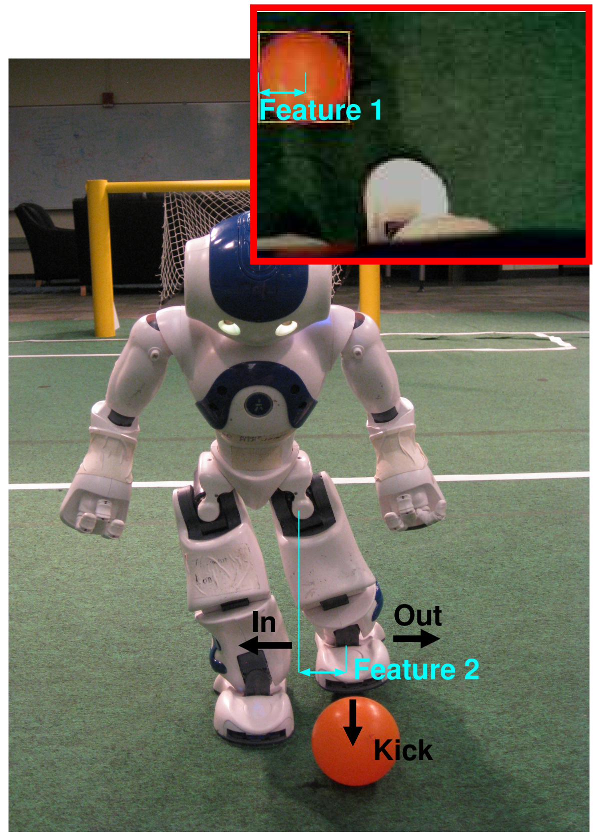

Fig. 4. This robot is eager to learn! For the learning problem, the robot's state consisted of the x coordinate of the ball in the robot's camera image and the distance the robot's foot was shifted out from its hip. The actions available to the robot were to move the leg in, out, or kick.

VI. RESULTS

For the first experiment, the ball was always placed at the same location relative to the robot. The ball was placed 30 mm left of the penalty mark, requiring the robot to shift its leg out 3 or 4 times before kicking the ball to aim it past the keeper. We ran each of the three learning algorithms for 30 trials over 100 episodes of the task. The cumulative reward of the learning algorithms over 100 episodes is shown in Figure 5. We show the cumulative reward plot to demonstrate how quickly the algorithms learn and we analyze the quality of their final policies below. RL-DT learned the task quickly, having significantly ($p < .0005$) more reward per episode than the other two algorithms from episode 5 to episode 34. At episode 35, R-MAX had also learned the task and its performance was no longer significantly different than RL-DT.

[3] http://www.cyberbotics.com

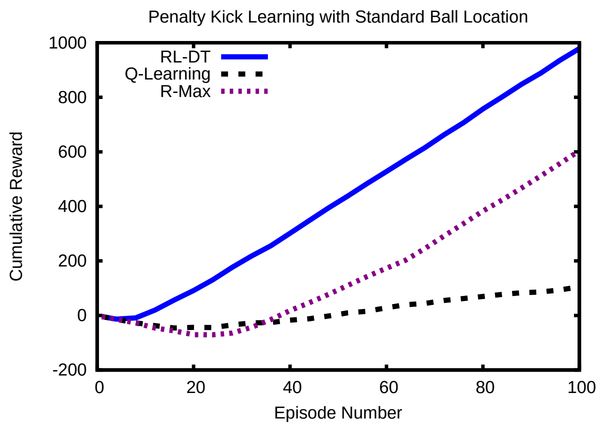

Fig. 5. Cumulative reward of the learning agents on the penalty kick task with a standard ball location. The final policies are analyzed in the text.

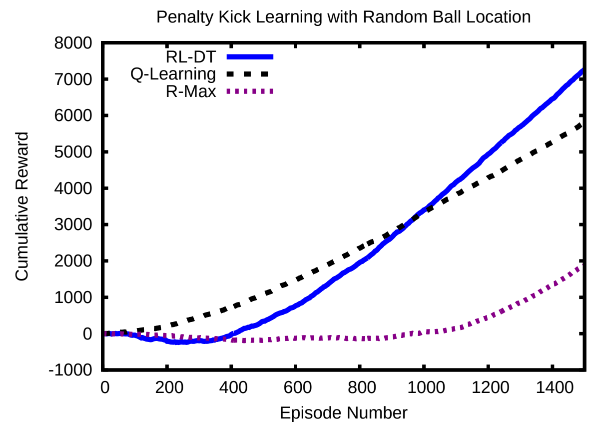

Fig. 6. Cumulative reward of the learning agents on the penalty kick task with a random ball location.

The optimal policy in this task was to shift the leg outward 4 times, so that the robot's foot was 112 mm out from its hip, and then kick. Kicking from this position would score on 85% of its attempts, resulting in an average reward per episode of 12.7. Kicking from 108 mm was a close second best, as it scored 75% of the time, but took one fewer action to get to the kicking position, resulting in an average reward per episode of 11.5. RL-DT converged to one of these two policies in 28 of the 30 trials, Q-LEARNING in 18 of the 30 trials, and R-MAX in all 30 trials.

Following the first experiment, we ran a second experiment with the ball in a random location relative to the robot to simulate the noisiness of the robot's approach to the ball. We first determined the range of ball locations where it was possible to score from and then randomly placed the ball in this region, which was between 0 and 34 mm or between 74 and 130 mm left of the penalty mark. In this experiment, the robot needed to use the state feature about the ball's location in its camera image to determine how far it needed to shift its leg to line up the ball properly to score a goal.

Plots of the average cumulative reward for the three algorithms over 30 trials are shown in Figure 6. Here, it was possible to score at many positions without shifting the leg and Q-LEARNING performed well by quickly learning to score consistently at these positions. While Q-LEARNING performed well early, its final policy was not as good as that of the other two algorithms. Figure 7 shows the percentage of tries that each algorithm scored at each ball position during the final 200 episodes. Q-LEARNING does very well on the positions where there was no leg shift required, but was unable to learn to score on more difficult positions, such as when the ball was offset between 74 and 84 mm. The two model-based methods performed more exploration and learned to score from these positions. RL-DT explored fast enough that it was able to accumulate enough reward from its better policy to surpass the cumulative reward of Q-LEARNING and it had a better policy than RMAX at the end of the 1500 episodes.

Following these experiments, we ran one trial of RL-DT on the physical robot (which is highlighted in the accompanying video [4]). In this case, we manually reset the robots and ball to the correct positions between each episode, resulting in more noise than in the simulator. The cumulative reward plot for the one physical robot trial is shown in Figure 8 along with the cumulative reward averaged over the 30 trials of RL-DT in the simulator for comparison. The results the algorithm achieves on the real robot are very similar to its performance in the simulator, validating that these experiments do cross over to the physical robot.

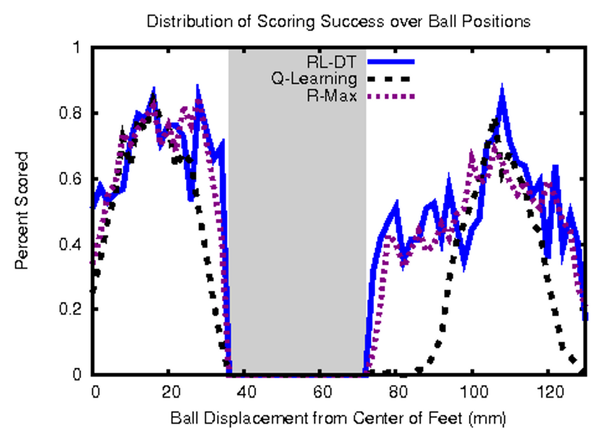

Fig. 7. This graph shows the percentage of tries that each algorithm scored at each ball position in the last 200 episodes. Note that the ball's initial position was never between 34 and 74 mm because these locations were impossible to score from.

Next we ran the final learned policy from the simulator for the standard ball location case on the real robot, to see how well the policy transferred over. The learned policy from the simulator was to shift the leg outward 4 times, so that the robot's foot was 112 mm out from its hip, and then kick. On the real robot, this policy scored on 7 of 10 attempts. In the simulator, this policy scored 85% of the time. However, in the simulator, if the ball was moved 4 mm in either direction, its success rate dropped to 65% or 75%. Noise in the placement of the real ball may have caused the lower scoring percentage, as well as the fact that the ball does not

[4] http://www.cs.utexas.edu/~AustinVilla/?p=research/rl_kick

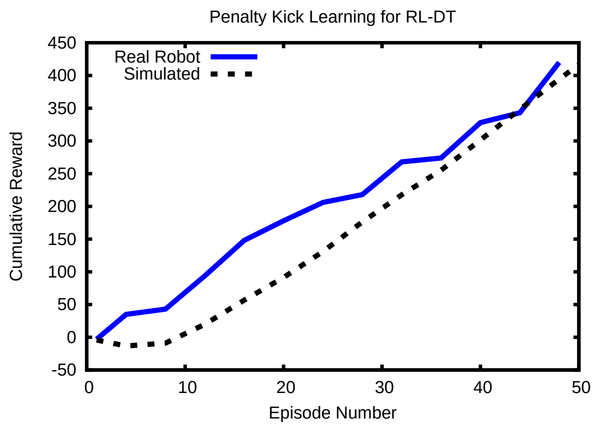

Fig. 8. Cumulative reward of RL-DT on one trial on the real robot and averaged over 30 trials in the simulator with a standard ball location.

roll as straight in reality as it does in the simulator.

VII. Related Work

There are a few learning algorithms that are based on a similar premise to ours. SLF-RMAX [11] learns which state features are relevant for predictions of the transition and reward functions. The algorithm enumerates all possible combinations of input features as elements and then keeps counts of visits and outcomes for all pairs of these elements. It then determines which elements are the relevant factors. The algorithm makes predictions when a relevant factor is found for a queried state; if none is found, the state is considered unknown and explored. Looking at subsets of features allows SLF-RMAX to perform less exploration than R-MAX, but keeping counters for all possible combinations of input features is very computationally expensive.

Degris et al. [12] use decision trees to learn a model of the MDP. A separate decision tree is built to predict each next state feature as well as the reward. Unlike our algorithm, theirs attempts to predict the absolute transition function, which may not generalize as well as the relative effects. Degris et al.'s algorithm calculates an ϵ-greedy policy through value iteration based on the model provided by the decision trees. Our approach differs from theirs by starting in an R-MAX-like exploration mode, which provides a more guided exploration approach than an ϵ-greedy policy.

There are also a few examples in the literature of reinforcement learning being applied to robots. In [13], the authors introduce an algorithm called RAM-RMAX, which is a model-based algorithm where some part of the model is provided to the algorithm ahead of time. They demonstrate the algorithm on a robot learning to navigate on two different surfaces.

In Robot Learning [14], the authors provide an overview of robot learning methods, including reinforcement learning. Specifically, they look at ways to scale up reinforcement learning and other methods to robots. One of the methods that they mention for scaling up learning methods is to learn action models, which is similar to the models of transition effects of actions that RL-DT learns with its decision trees.

VIII. Discussion

We demonstrated that reinforcement learning, and specifically model-based reinforcement learning, can be successfully used to learn a task on a humanoid robot. Our learning algorithms were able to learn to score penalty goals in all three experiments: with a standard ball location, a randomized ball location, and on the physical robot. Specifically, our algorithm, RL-DT, was very successful at solving each task quickly.

These experiments demonstrate that RL-DT is a good choice for learning tasks on a robot. Its generalized models and targeted exploration allow it to learn a reasonable policy quickly without taking too many actions, which can be very expensive and time consuming to get on a real robot. This ability allowed RL-DT to accrue the highest cumulative rewards on all the experiments.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-0615104 and IIS-0917122), ONR (N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the Federal Highway Administration (DTFH61-07-H-00030).

References

[1] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry, "Autonomous helicopter flight via reinforcement learning," in *Advances in Neural Information Processing Systems 17*. MIT Press, 2004.

[2] J. A. Bagnell and J. C. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *In International Conference on Robotics and Automation*. IEEE Press, 2001, pp. 1615–1620.

[3] R. Zhang and P. Vadakkepat, "An evolutionary algorithm for trajectory based gait generation of biped robot," in *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[4] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

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

[6] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[7] R. I. Brafman and M. Tennenholtz, "R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 953–958.

[8] R. T. Committee, "Robocup standard platform league rule book," May 2009.

[9] T. Hester and P. Stone, "Generalized model learning for reinforcement learning in factored domains," in *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2009.

[10] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[11] A. L. Strehl, C. Diuk, and M. L. Littman, "Efficient structure learning in factored-state mdps," in *AAAI*. AAAI Press, 2007, pp. 645–650.

[12] T. Degris, O. Sigaud, and P.-H. Wuillemin, "Learning the structure of factored markov decision processes in reinforcement learning problems," in *ICML '06: Proceedings of the 23rd international conference on Machine learning*. New York, NY, USA: ACM, 2006, pp. 257–264.

[13] B. R. Leffler, M. L. Littman, and T. Edmunds, "Efficient reinforcement learning with relocatable action models," in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 2007, pp. 572–577.

[14] J. Connell and S. Mahadevan, *Robot Learning*. MA: Kluwer Academic Publishers, 1993.