Instance-Based Action Models for Fast Action Planning

Mazda Ahmadi and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500, Austin, TX 78712-0233
Email:{mazda,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{mazda,pstone}

***Abstract*— Two main challenges of robot action planning in real domains are uncertain action effects and dynamic environments. In this paper, an *instance-based* action model is learned empirically by robots trying actions in the environment. Modeling the action planning problem as a Markov decision process, the action model is used to build the transition function. In static environments, standard value iteration techniques are used for computing the optimal policy. In dynamic environments, an algorithm is proposed for fast replanning, which updates a subset of state-action values computed for the static environment. As a test-bed, the goal scoring task in the RoboCup 4-legged scenario is used. The algorithms are validated in the problem of planning kicks for scoring goals in the presence of opponent robots. The experimental results both in simulation and on real robots show that the instance-based action model boosts performance over using parameterized models as done previously, and also replanning significantly improves over original off-line planning.**

I. Introduction

In many robotic applications, robots need to plan a series of actions to achieve goals. Some of the challenges that robots face in many real applications, are (1) noisy actions, (2) dynamic environments, and (3) actions not being well-defined. In this paper, the problem of action planning in dynamic environments with uncertain action effects is considered; an *instance-based* action model is introduced, and used for action planning. An on-line transition model modification is used for dealing with dynamic environments.

Learning action models has been studied in classical planning (e.g. see [1], [2]), and also in probabilistic planning (e.g. see prioritized sweeping [3]). But those methods use many trials to learn the model; instead we use domain heuristics to learn the model with few experiments prior to planning.

A common shortcoming of prior methods for learning probabilistic action models is the assumption that the noise is normal, which in many cases is not true. To overcome that shortcoming, we propose an instance-based approach for learning the action model. The action model is built empirically by trying actions in the environment. Each sample effect is stored and is considered individually for planning purposes.

The planning problem is modeled as an MDP. The transition function of the MDP is built with the help of the learned action model. Using value iteration [4] with state aggregation, a plan which maximizes the received reward is generated. When the environment is static, the value iteration algorithm is run offline.

In dynamic environments, the planning should be performed online. The online algorithm should be fast enough to be within computational bounds of the robots.

Fast replanning algorithms for robotic applications has been studied for classical planning problems (e.g. see [5], [6]). However, the probabilistic replanning algorithms that we know of (e.g. [7]), are computationally expensive.

Observation of each dynamic factor changes the transition function of the MDP. But it only changes the value of a small subset of state-action pairs. In the replanning algorithm, using domain-dependent heuristics, the state-action pairs that are affected by the dynamic factors are discovered, and only the values of those state-action pairs are updated.

To evaluate our methods, we use a goal scoring task from the 4-legged RoboCup competitions. Chernova and Veloso [8] learn models of kicks for the 4-legged robots, however they do not discuss how they use this model. Their model consists of the speed and direction of the ball in the first second. We extend their work by introducing an instance-based model instead of their parameterized model, and show the advantages of using such an instance-based model. Furthermore, we use the kick model for action planning in the presence of opponent robots.

In Section II the RoboCup test-bed is presented. In the next three sections, we go back to the abstract MDP case. In Section III the details of the instance-based action model, and how to learn in it, is provided; Section IV is about the planning algorithm; and Section V considers the planning problem in dynamic environments. The implementation details for the RoboCup domain are presented in Section VI. In Section VII, experimental results are provided, and finally the paper is concluded with Section VIII.

II. Problem Description

One goal of a robot in the RoboCup four legged league is goal scoring. In this work we explore single robot goal scoring possibly against multiple opponents. We assume the opponents only block the ball, and do not kick or grab it.

The robots in the RoboCup four legged league are Sony ERS-7 four-legged AIBO robots (Figure 1). The robot's vision device is a camera mounted on the head of the robot. It can capture 208 $\times$ 160 frames of pixels at roughly 30Hz.[1] It has 20 degrees of freedom and a 576Mhz on-board processor.

As baseline software, we use the UT Austin Villa code base [9], which provides robust color-based vision, fast locomotion, and reasonable localization within a 3.6m $\times$ 5.4m area[2] via a particle filtering approach. Even so, the robot is not, in general, perfectly localized, as a result of both noisy sensations and noisy actions. The robot also has limited processing power, which limits the algorithms that can be designed for it.

Fig. 1. ERS-7 Sony AIBO robot

The baseline software provides different types of kicks. The two that are considered in this work are called FALL-KICK (see Figure 2) and HEAD-KICK (see Figure 3). The effects of these kicks are probabilistic based on how accurately they are executed and what exact part of the robot hits the ball.

Fig. 2. Representation of FALL-KICK from left to right

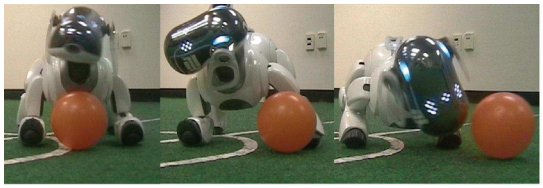


Fig. 3. Representation of HEAD-KICK from left to right

III. Instance-Based Action Model

The first step of planning any action is understanding its effects. We build the action model empirically by trying actions in the domain. In most of the real robot applications, actions have probabilistic effects. Thus the action model should be able to represent the uncertainty in action effects.

Previous methods (e.g. [8]) use parameterized models of actions. Most popular method for modeling the noise assume a Gaussian distribution for each of the parameters of the action model. Instead we take an *instance-based* approach, where each action effect from experiments is called a sample action effect, and is stored in the model. We claim and show in the experiments, that our instance-based action model is more efficient than a parameterized action model.

In addition to noisy action effects, robots are faced with noise from other sources (e.g. uncertain localization). Previous methods of building action models (e.g. [8]) try to minimize the effects of such *other* noise on the action model. If the action model is noise-free, the effects of the environment's noise should be considered in some other way for action planning. Instead, we aim at having noise from all sources to also be captured by the action model. In this way, if the action model is used with any planning algorithm, all the sources of noise are also considered. This requires the samples to be collected in a situation similar to the one that the robot faces in the real environment, not in some other controlled setting.

An example of an instance-based action model for the FALL-KICK in the RoboCup domain is shown in Figure 4.

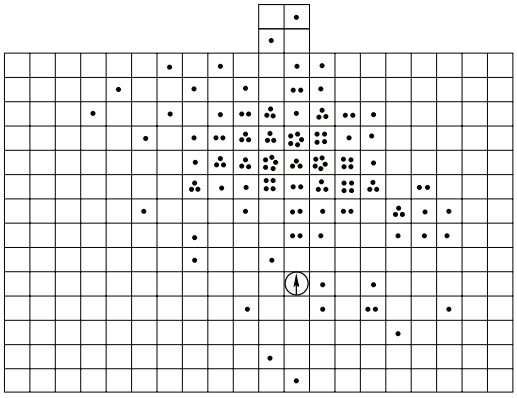


Fig. 4. Representation of the model of FALL-KICK. The position of the robot and kick direction is marked by an arrow.

IV. Planning

In this section, we show how the instance-based action model can be used for action planning. We model the problem as a Markov decision process (MDP) and use a value iteration method to solve it. In this section the environment is assumed to be static, dynamic environments are considered in the next section.

The planning problem is modeled as an MDP $(S, A, Pr(s^{'}|s,a), R)$, where:

- S is a discrete set of *states*. If the state space is continuous, it should be descritized. We show that descritizing the state space does not have much effect in the RoboCup goal shooting test-bed.
- A is the set of possible actions.
- $Pr(s^{'}|s,a)$ is the state transition probabilities. It gives the probability of getting to state $s^{'}$ from taking action

[1] Due to the computational intensity of image processing, our robots typically make decisions at roughly 25Hz.

[2] The field is as specified in the 2005 rules of the RoboCup Four-Legged Robot League: `http://www.tzi.de/4legged`

a in state s. Because of noise in the environment, and uncertainty of action effects, the transition function is stochastic

- $R(s,a)$ is the *reward function*.

The goal of the robot is to find a *policy* $\pi : S \mapsto A$ that maximizes the received reward. The policy determines which action is chosen by the robot from each state.

$Pr(s^{'}|s,a)$ is not known to the robot, however the robot can use the action model to approximate it. The approximation of $Pr(s^{'}|s,a)$ is called $\widetilde{Pr}(s^{'}|s,a)$. The approximation is based on domain-dependent heuristics. $R(s,a)$ is also computed with the help of action model. The details of computing $\widetilde{Pr}(s^{'}|s,a)$ and $R(s,a)$ for the RoboCup goal shooting test-bed is presented in Section VI.

For state s, the expected value $V_t^{\pi}(s)$ is defined as the sum of reward from s for t actions (stage t), while following policy π. $V_T^*(s)$ is the optimal T-horizon policy if $V_T^*(s) \geq V_T^{\pi}(s)$ for all policies π and all $s \in S$ in T horizon. The bellman principle of optimality [4] establishes the following relationship between the optimal value function of stage t and the optimal value function of the previous stage:

$$V_t(s) = \max_{a \in A}\{R(s,a) + \sum_{s^{'} \in S} Pr(s^{'}|a,s)V_{t-1}^*(s^{'})\} \qquad (1)$$

The above equation forms the basis for *value iteration* algorithm for finite-horizon algorithms[10]. $Q_t^{\pi}(s,a)$ is defined as the sum of the rewards received from state s, with t actions, while following policy π (see Equation 2), if the first action to execute is a. $Q_t^*(s,a)$ is for the case where the policy is optimal. Advantage of using $Q_t^{\pi}(s,a)$ over $V_t^{\pi}(s,a)$, is presented in the next section, where only a subset of Q values needs to be updated.

We use following two equations for value iteration, which are basically the same as Equation 1:

$$Q_t(s,a) = R(s,a) + \sum_{s^{'} \in S} \widetilde{Pr}(s^{'}|a,s)V_{t-1}^*(s^{'}) \qquad (2)$$

$$V_t(s,a) = \max_{a \in A} Q(s,a) \qquad (3)$$

Value iteration starts with $Q_0^*(s,a) = R(s,a)$ and with above equations the Q and V values are updated. Using value iteration algorithm with finite horizon, and approximating Pr with $\widetilde{Pr}$, the expected values for the states of the system are computed offline, and the V_T^* values are ready to be used for the robot.

Up to this point in this section, we provided the MDP formalism and the value iteration algorithm to solve it. None is a contribution of this paper, but they are the basis of the replanning algorithm presented in the next section.

When the system is in state s, a common practice for action selection is to choose action a such that it maximizes $Q(s,a)$. This approach coupled with using a discrete state representation results in a sub-optimal policy. In order to alleviate this effect, for action selection, instead of the state, the robot uses its true state estimate that maximizes the following value:

$$R(TS,a) + \sum_{s^{'} \in S} \widetilde{Pr}(s^{'}|a,TS)V_T^*(s^{'}) \qquad (4)$$

where TS is the true state estimate of the robot.

This way, the effects of discretizing the state are deferred to the next step, and it results in a better policy. In the experiments section (Section VII), we empirically show that in the RoboCup test-bed, discretizing the environment with the true state for action selection is very close in performance to using the true state in planning.

Note that using the true state to evaluate all possible actions is a computational burden on the robot, but it can be done in manageable time. In the experiments section the advantage of this action selection method is showed empirically in simulation.

V. Replanning

In the previous section, the environment was assumed to be static, and the value function could be computed offline. In this section, the possibility of the presence of dynamic factors (e.g. the presence of opponent robots in the RoboCup test-bed) is considered. The dynamic factors change the transition function of the MDP, and the value function for the new MDP needs to be computed online. The robot's weak processing power does not allow for full value iteration for consideration of dynamic factors. In this section a fast replanning algorithm is presented, which uses the Q-values that are computed for the static environment.

If the dynamic factors are considered in computing $\widetilde{Pr}(s^{'}|a,s)$ in Equation 5, the value iteration and action selection algorithms described in Section IV can also be applied to the dynamic environment. As mentioned, the problem is that we can no longer compute the V values offline, and that algorithm is too slow to run online on robots.

As with $Q_t^{\pi}(s,a)$, $Q_t^{\pi}(s,a|F)$ is defined as sum of the received reward in the presence of dynamic factors F, from state s, with t actions, while following policy π (see Equation 4), if the first action to execute is a. $Q_t^*(s,a|F)$ is for the optimal policy.

Difference between $Q^*(s,a)$ and $Q^*(s,a|F)$ is only substantial when F has a direct effect on the Q-value of (s,a) state-action pair. For the rest of the states, $Q^*(s,a|F) \approx Q^*(s,a)$. This fact, which is typical of many dynamic systems, where the dynamic factors have an effect on only a subset of the Q-values, is the basis of the proposed replanning algorithm.

Assuming $Q(s,a)$ is the current Q-function, the algorithm for updating the Q-values in the event of observing dynamic factor f is as follows:

1) Flag the (s,a) pairs that are potentially affected by f.
2) For flagged pair (s,a), there is a good chance that $Q(s,a|f)$ is very different from $Q(s,a)$. Thus, if (s,a) is flagged, $Q(s,a|f)$ is initialized to zero, and otherwise, it is initialized to $Q(s,a)$.

3) Only for flagged pairs (s, a), the $Q(s, a|f)$ are updated using Equation 2. Notice that only one round of updates is performed. After all the Q-updates, the V values are re-computed using Equation 3.

The action selection is the same as the one discussed in the previous section. The two main benefits of our replanning algorithm are that it is fast and the robot can distribute the value iteration steps over different decision periods (updates to Q can be done in several time cycles), so the robot processor does not get stuck in this process.

Recall that the robot does another level of inference on the effect of actions in the action selection step, so the effects of adding f is effectively backed up two steps.

VI. Implementation Details

In this section the implementation details of the instance-based action model (Section III), planning (Section IV), and replanning (Section V) algorithms for the goal scoring test-bed (Section II) are presented.

A. Learning Kick Model

The main action for scoring goals is kicking (assuming the robot walks to the new position of the ball after each kick action). Kicks (as shown in Figures 2 and 3) have uncertain (i.e. probabilistic) effects on the final position of the ball, which is based on the exact point of contact between the robot and the ball. In this section we present the implementation details of learning the instance-based kick model.

Chernova and Veloso [8] use average and standard deviation for the speed and angle of each kick to model the kick. They measure the angle and distance one second after the kick, thus measuring the speed. By just considering the kick in the first second, and also setting the initial position of the ball by hand, they try to minimize the noise in their kick model. They do not provide details of how they use this model. But a popular way of using average and standard deviation is modeling the parameters with Gaussian distributions.

For creating the kick model, the robot goes to the ball, grabs it, and kicks it to the center of the field. Right before kicking, it records its position (kick position), which includes the usual localization errors. Then, the robot follows the ball, and when the ball stops moving, a human observer sends a signal to the robot, and the robot records the new ball position (final ball position). The gathered sample kick effects (kick position, final ball position) are processed by an offline program, and for each kick sample, the difference between the final ball position and the kick position is computed. These changes in x and y coordinates get discretized and are stored in a grid.

The learned action model is a three dimensional array KT, where for kick type k, $KT[k][x][y]$ represents the number of kicks that changed the position of the ball for x grid cells in the x-axis and y grid cells in the y-axis. Figure 4 shows the KT[FALL-KICK] where the position of the robot (kick position) and kick direction is shown with the arrow, and each black dot represents one sample action effect resulting in the ball to end up in that grid cell. The main rectangular grid is in the size of the legged soccer field.

Two fundamental difference between our model and that of Chernova and Veloso's [8] and other usual action models are that ours is (1) instance-based, and (2) unlike usual action models, where the designers make an effort to reduce the noise (e.g. tracking for one second, and putting the ball in front of the robot in [8]), we try to capture the full environmental noise in the action model.

B. Planning

We begin by dividing the robot's environment into the disjoint grid G. Dotted lines in Figure 5 shows the actual grid used in the experiments. KT is the set of different kick types available to the robot, and D is a discrete set of possible kick directions.

In Section VII, we empirically show that discretizing the field does not have much effect on the efficiency of the algorithm.

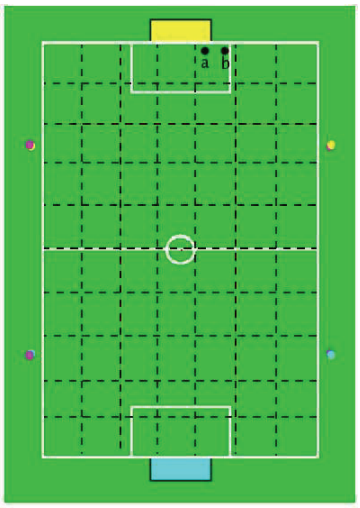


Fig. 5. Soccer field which is divided into a grid.

The MDP $(S, A, Pr(s'|s, a), R)$ for the test-bed problem is defined as:

- $S = G$ is a set of *states*, representing the grid cell that the ball is in. Whenever the grid cell is used as a position, the center of the cell is assumed. In the rest of the paper, the state is also used to point at the grid cell where the ball is located in that state.
- $A = KT \times D$ is the set of possible actions, which is kicking with a specific kick type (from KT) at a direction (from D).
- $Pr(s'|s, a)$ is the state transition probabilities.
- $R(s, a)$ is the *reward function* and is the probability of scoring a goal from state s using action a.

To approximate $\widetilde{Pr}(s'|s, a)$, kick model is projected to the starting point of the kick with the kick direction to get a

distribution over kick effects (i.e. the possible cells that the ball can land in). $\widetilde{Pr}(s^{'}|s,a)$ is computed by the following equation:

$$\widetilde{Pr}(s^{'}|s,a) = \frac{KT[k][s^{'}_{xd} - s_{xd}][s^{'}_{yd} - s_{yd}]}{N[k]} \quad (5)$$

where s_{xd} (s_{yd}) is the x-index (y-index)[3] of the grid cell containing the ball in s, after the transformation that transforms the kick direction to the x-axis. For each state s, the center of the cell is used for computing s_{xd} and s_{yd}. k is the type of action a, and $N[k]$ is the number of kick samples of type k.

$R(s,a)$ is also computed with the help of action model, that is, $R(s,a)$ is equal to the percentage of the sample kick effects from kick model that result in a goal from state s by taking action a.

The value iteration and action selection algorithms described in Section IV are used for computing Q-values and action selection.

C. Replanning

The presence of opponent robots in the goal scoring test-bed are considered as dynamic factors. We assume that opponent robots only block the ball and do not grab or kick it. The robot models the blocking as ball reflection from the opponent robots. That is, if the kick trajectory of a kick sample of the kick model hits any of the opponent robots, the reflection from the opponent robot is assumed by the robot as the final position of the ball.

The replanning algorithm presented in Section V is used to update the Q values. In the first step of the algorithm, a pair (s,a) in the presence of the opponent robot f is flagged if f is reachable by an average a kick from s (i.e. instead of the kick model of a, it uses the average distance and angle of kick a). [4]

One special case to consider is when the opponent robot o intersects with a grid cell g, and based on a sample action effect, the ball also ends up in grid cell g. The value of a point in cell g is highly dependent on which side of opponent o the point is located. If the final ball point is on the same side of o as the center of cell g, $V^*(g)$ is used, and if not, average V values of the cells adjacent to g and on the same side of o as the ball are used.

VII. Experimental Results

The algorithms are evaluated both on a custom-built AIBO simulator [9] and on real AIBO robots. The simulator, though abstract with respect to locomotion, provides a reasonable representation of the AIBO's visual and localization capabilities, and allows for a more thorough experimentation with significant results. The kick model used in the simulator is the same as the one used on the robot. Thus, the simulation results are based on the assumption of a correct kick model. There are methods of active localization (e.g. [11]) to reduce the localization errors, which are not considered here and can be used orthogonally with this work. Thus, robots should deal with large localization errors (in the simulation, this is reflected in the learned kick model), and that lowers the scoring percentages.

Five different algorithms are compared in this section. The considered algorithms include:

- **AtGoal:** In this algorithm, the robot directly shoots at the goal using fall-kick which is a more accurate kick. It is used as a baseline algorithm.
- **RePlan:** This is the planning algorithm presented in Section V, where the robot observes the position of the opponent robots online.
- **FullPlan:** This algorithm is used for comparison with RePlan. In FullPlan, it is assumed that the position of the opponent robots is known as a priori, and an offline algorithm performs the full value iteration, and passes the Q-values to the robot.
- **Plan:** This is the planning algorithm (Section IV) for the clear field, where no opponent robot is present. In clear fields, this is the algorithm of choice, but it is also used to compare to RePlan in the presence of opponent robots.
- **NormalPlan (NormalFullPlan):** This is similar to Plan (FullPlan for the case of NormalFullPlan) algorithm, but instead of full kick model, a normalized kick model is used. Average and standard deviation (similar to [8]) for distance and angle of each kick sample is computed. Two different Gaussians are assumed, one for the angle, and the other for the distance of the kick samples. Each time that the robot considers a kick, it draws n random samples from each of the Gaussians, where n is equal to the number of different kick samples that it would have considered for the instance-based kick model. For each of the n pairs of (angle, distance), it evaluates the kick. The final evaluation is based on the average of the n evaluations. This experminet is used to show the power of the instance-based kick model compared to the parameterized kick model with a normalized noise.

The grid used in the experiments is 7×10 and is shown in Figure 5. The horizon for Plan and FullPlan algorithms is set at 20, that is number of the value iteration rounds is 20.

A. Simulation Results

In the first experiment, the environment is assumed to be static. In later experiments, the algorithms are evaluated in dynamic environments. At the end of this section, the effects of considering the true position for action selection (see Section IV) is investigated. While doing so, we also argue that, the effects of assuming a grid for representing the position are minor.

[3] x-index (y-index) of grid cell g is the floor value of x-coordinate (y-coordinate) of the center of cell g divided by length of a cell in x-axis (y-axis).

[4] More elaborate techniques that consider all kick samples proved to need heavy computation, which is not available on the robots.

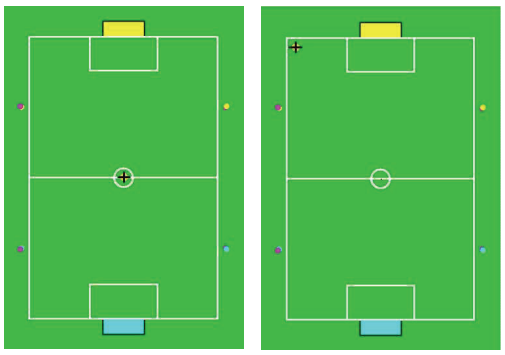
Each episode starts with the ball stationed in a starting point, and is finished when a goal is scored or the ball goes out of bounds. Each trial consists of 100 episodes (except for Section VII-A.5). The percentage of the episodes which resulted in goals and the average number of kicks per episode are computed for each trial. Except for Section VII-A.5, the reported data is averaged over 28 trials.

1) Clear Field Experiment: We start the experiments with no opponents (Figure 6). Two starting points for the ball are considered. The experiments for the center point (Figure 6(a)) and the up-left point (Figure 6(b)) are presented.

Recall that the ATGOAL algorithm only uses FALL-KICK. For a fair comparison between ATGOAL and PLAN, the result for the PLAN*(*FALL-KICK*)* algorithm, which is similar to PLAN, but only uses FALL-KICK, is presented. As the results in Table I suggest, using the HEAD-KICK does not make much of a difference for the PLAN algorithm. For that reason, in the next experiments PLAN(FALL-KICK) is not considered. However, one of the benefits of the algorithms presented in this paper is enabling the addition of newly designed kicks.

The scoring percentage and average number of kicks per episode for ATGOAL, PLAN and NORMALPLAN are presented in Table I. As it is shown in the table, when the starting ball point is at the center of the field, planning significantly increases performance over the ATGOAL algorithm by 30%, and increases the performance by 76% when the starting ball position is the up-left point. For the planning algorithm, the average number of kicks is also higher, which is a compromise for achieving better scoring percentage.

The effectiveness of the instance-based kick model is shown by the significant advantage of the PLAN algorithm compared to NORMALPLAN in Table I. Using the instance-based kick model for the center and up-left starting point increases the performance by 43% and 42% compared to NORMALPLAN, respectively.



(a) (b)

Fig. 6. Clear field. (a) The center of the field is the starting point of the ball. (b) The up-left point is the starting point for the ball.

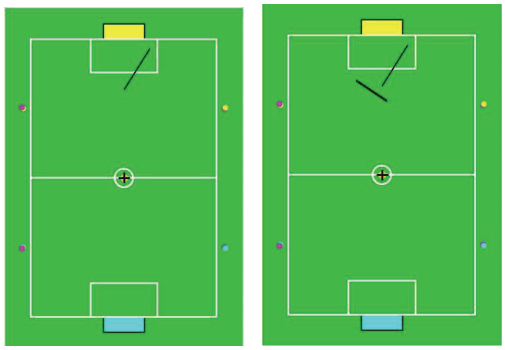
2) One Opponent Experiment: In this experiment, a stationary opponent robot is placed in the field. The field with the opponent robot is shown in Figure 7(a). The ball's starting point is at the center of the field. The algorithms ATGOAL, REPLAN, FULLPLAN, PLAN, and NORMALFULLPLAN are compared. Success percentage and average number of kicks is presented for the above-mentioned algorithms in Table II.

Start. point	Algorithm	Scoring %	# Kicks/episode
Center	ATGOAL	46.28 ± 4.85	3.41 ± 0.12
Center	PLAN	60.53 ± 4.87	9.71 ± 0.91
Center	PLAN(FALL-KICK)	58.78 ± 5.03	9.61 ± 0.92
Center	NORMALPLAN	42.11 ± 5.75	10.12 ± 0.78
Up-Left	ATGOAL	29.11 ± 5.03	1.74 ± 0.11
Up-Left	PLAN	51.39 ± 5.40	6.35 ± 0.65
Up-Left	PLAN(FALL-KICK)	53.14 ± 5.31	6.11 ± 0.80
Up-Left	NORMALPLAN	39.00 ± 5.21	7.12 ± 0.77

TABLE I

COMPARING DIFFERENT ALGORITHMS FOR THE TWO STARTING BALL POINTS IN THE CLEAR FIELD EXPERIMENT (SEE FIGURE 6).

REPLAN algorithm significantly improves scoring percentage compared to ATGOAL, PLAN and NORMALFULLPLAN by 104%, 13%, and 45% respectively. It is also very close in performance to FULLPLAN, where the transition function is assumed to be known as a priori and the Q-values are computed offline without computational limitations. The difference is 1.5%, which is not significant. The average number of kicks per episode is the most for the REPLAN algorithm, which is considered not important compared to scoring efficiency.



(a) (b)

Fig. 7. (a) The field with one stationary opponent robot. (b) The field with two stationary opponent robots.

Algorithm	Scoring %	# Kicks/episode
ATGOAL	32.00 ± 5.83	1.77 ± 0.11
REPLAN	65.92 ± 4.20	11.74 ± 1.00
FULLPLAN	67.10 ± 3.92	8.22 ± 0.67
PLAN	57.42 ± 3.76	8.42 ± 0.63
NORMALFULLPLAN	45.17 ± 4.59	9.59 ± 0.69

TABLE II

SCORING PERCENTAGE AND AVERAGE NUMBER OF KICKS FOR DIFFERENT ALGORITHMS, IN THE ONE OPPONENT ROBOT SCENARIO (SEE FIGURE 7(A)).

3) Two Opponent Experiment: In this experiment, an additional robot is positioned in the field. The field is shown in

Figure 7(b). The scoring rate and average number of kicks for different algorithms is presented in Table III. The trend in the result is consistent with the observation in the one opponent robot scenario in the previous experiment (Section VII-A.2).

Algorithm	Scoring %	# Kicks/episode
ATGOAL	21.85 ± 4.64	5.09 ± 0.32
REPLAN	54.25 ± 5.58	19.45 ± 1.97
FULLPLAN	54.64 ± 5.59	11.43 ± 0.90
PLAN	46.46 ± 5.10	13.04 ± 1.14
NORMALFULLPLAN	38.07 ± 4.39	11.84 ± 0.96

TABLE III
COMPARING DIFFERENT ALGORITHMS FOR PLAYING AGAINST THE TWO OPPONENT CASE (SEE FIGURE 6(A)).

4) Moving Opponents Experiment: In this experiment two *moving* opponent robots are present in the field. In each episode, the opponent robots start from the position of the opponent robots in the previous experiment (Figure 7(b)), and after each kick, they move $150cm$[5] randomly in one of the four main directions (i.e. left, right, up or down). In an effort to reduce the noise in the result, the seed of the pseudo-random generator, which determines what direction opponent robots move, is fixed for all trials (not episodes).

The scoring percentage and average number of kicks for ATGOAL, REPLAN, and PLAN algorithms is provided in Table IV. The performance of the REPLAN is 178% better than ATGOAL and 6% better than the PLAN algorithm. Since the robot movement is random, the position of the opponent robots can not be known as a priori, and no offline algorithm like FULLPLAN can be developed.

Algorithm	Scoring %	# Kicks/episode
ATGOAL	20.60 ± 5.14	4.52 ± 0.37
REPLAN	57.32 ± 4.82	9.51 ± 0.65
PLAN	54.14 ± 3.98	9.51 ± 0.67

TABLE IV
COMPARING ATGOAL, PLAN AND REPLAN ALGORITHMS IN THE PRESENCE OF TWO MOVING OPPONENT ROBOTS.

5) Investigating the Use of the True Position in Action Selection and the Effects of Using a Grid: As mentioned in Section IV, for action selection, the robot uses its true position instead of the state it is in. In this section the advantage of using the true position is shown. We also investigate how much of the advantage is due to the precision needed for scoring a goal with a single kick. It is also argued that the use of the grid does not significantly affect the performance.

The scenario of Section VII-A.1, where the starting point of the ball is the up-left point (Figure 6(b)) is assumed, and for added precision in the results, each trial consists of 1000 episodes, and the presented data is averaged over 13 trials. The effects of using three different action selection methods for PLAN algorithm is compared with the one presented in Section IV (which is called ALWAYSFROMPOS here). Note that the Q-values are the same for all algorithms, and only the action selection method is different. The three considered action selection algorithms are as follows:

- ALWAYSFROMSTATE**:** Robot chooses the action that maximizes $Q(s, a)$. That is, the robot effectively uses the center of the cell that it is in instead of its true position. As discussed in Section IV, this is the common practice in solving MDPs.
- FROMPOSIFGOAL**:** If from the true robot position *or* center of the cell that the robot is in, one of the sample kick effects results in a goal, the robot uses the true position for that particular sample kick effect, otherwise it uses the center of the cell.
- FROMSTATEIFGOAL**:** It is the opposite of the FROMPOSIFGOAL algorithm, that is, if from the true position *or* center of the cell, one of the sample kick effects results in a goal, the robot uses the center of the cell for that particular sample kick effect, otherwise it uses the true position.

Action Selection Alg.	Scoring %
ALWAYSFROMPOS	54.73 ± 1.00
ALWAYSFROMSTATE	46.47 ± 1.58
FROMPOSIFGOAL	51.50 ± 0.95
FROMSTATEIFGOAL	48.86 ± 1.38

TABLE V
COMPARING ACTION SELECTION ALGORITHMS, FOR THE SCENARIO OF FIGURE 6(B).

As evident from the results in Figure V, using the true position significantly increases the performance by 18% compared to using the states.

Notice that FROMPOSIFGOAL is 11% better than ALWAYSFROMSTATE, and just 6% worse than ALWAYSFROMPOS. Also FROMSTATEIFGOAL is just 5% better than ALWAYSFROMPOS which is not significant. This suggests that most of the added efficiency because of using the true position corresponds to the added precision needed for the last step of scoring goals.

We now claim that the advantage gained from using the true position for planning (though computationally almost impossible), as opposed to using a grid, is not significant.

Since the PLAN algorithm uses the true position for the current location of the ball, further use of the true position does not have any effect on the precision needed for the last step of scoring goals. Thus the effects of further use of true position is just in changing the position of the ball before goal scoring. Using the true position of the current location of the ball (as PLAN does) is far more important than the true position of the location of the ball after a kick, because the action model is probabilistic and approximate, and there is a distribution of the locations that the ball can land in after a kick. However, the difference of performance even for the first step is not signifi-

[5]Recall that the size of the field is $5400cm \times 3600cm$.

cant (difference of performance between FROMSTATEIFGOAL and ALWAYSFROMSTATE). This suggests that the effects of further consideration of the true position is not significant.

B. Real Robots

In this section, experiments on a real robot are reported. Robot experiments are time consuming and it is not possible to do as much trials as in simulation. First, experiments in the clear case (Figure 6(b)) and then against two opponent robots (Figure 7(b)) is provided.

1) Real Robot in Clear Field: The configuration is the same as the one shown in Figure 6. Each trial consists of 50 episodes, and the result for one trial is reported in Table VI. The trend is similar to the simulation experiment of the same configuration, and PLAN increases performance over ATGOAL and NORMALPLAN by 80% and 20% respectively.

Algorithm	Scoring %
ATGOAL	20
PLAN	36
NORMALPLAN	30

TABLE VI

COMPARING DIFFERENT ALGORITHMS FOR UP-LEFT STARTING BALL POINTS FOR THE CLEAR FIELD EXPERIMENT (SEE FIGURE 6(B)) WITH A REAL ROBOT.

2) Real Robot Against Two Opponent Robots: In this experiment the configuration of the field in Figure 7(b) is used for a real robot. Each trial consists of 25 episodes, and the result is reported in Table VII.[6] The result shows the same trend as the simulation in this field, and REPLAN is better than PLAN and ATGOAL.

Algorithm	Scoring %
ATGOAL	4
PLAN	16
REPLAN	24

TABLE VII

COMPARING DIFFERENT ALGORITHMS FOR REAL ROBOT IN THE EXPERIMENT AGAINST TWO OPPONENT ROBOTS (SEE FIGURE 7(B)).

VIII. CONCLUSION

Action planning problem in noisy environments, is considered and modeled as an MDP. An instance-based action model is introduced to model noisy actions. The action model is then used to build the transition function of the MDP. Furthermore a fast algorithm for action planning in dynamic environments, where dynamic factors have effect on only a subset of state-action pairs is introduced. To test these approaches, goal scoring in RoboCup 4-legged league is used as a test-bed. The experimental results show the advantage of using the instance-based approach compared to parameterized action models. Also it shows that the fast replanning algorithm outperforms the original off-line planning and approaches the best possible performance assuming the full transition model is known as a priori.

Instead of using heuristics to model the effects of dynamic factors on the transition model, one can learn the effects with experiment and build a more robust system. Also the extension of this approach to multi-robot systems is among the future works.

[6]Since in the used base code, the robots do not walk around opponent robots, in this experiment whenever the robot wanted to move across the opponent, the human removed the opponent momentarily.

ACKNOWLEDGMENT

The authors would like to thank the members of the UT Austin Villa team for their efforts in developing the software used as a basis for the work reported in this paper. Special thanks to Gregory Kuhlmann for developing the simulator. This research was supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

REFERENCES

[1] Q. Yang, K. Wu, and Y. Jiang, "Learning action models from plan examples with incomplete knowledge," in *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, June 2005.

[2] X. Wang, "Learning planning operators by observation and practice," in *Artificial Intelligence Planning Systems*, 1994, pp. 335–340.

[3] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[4] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[5] A. T. Stentz, "The focussed d* algorithm for real-time replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

[6] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, May 2002.

[7] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley, "Planning and reacting in uncertain and dynamic environments," *Journal of Experimental and Theoretical AI*, vol. 7, no. 1, pp. 197–227, 1995.

[8] S. Chernova and M. Veloso, "Learning and using models of kicking motions for legged robots," in *Proceedings of International Conference on Robotics and Automation (ICRA'04)*, May 2004.

[9] P. Stone, K. Dresner, P. Fidelman, N. Kohl, G. Kuhlmann, M. Sridharan, and D. Stronger, "The UT Austin Villa 2005 RoboCup four-legged team," The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, Tech. Rep. UT-AI-TR-05-325, November 2005.

[10] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[11] C. Kwok and D. Fox, "Reinforcement learning for sensing strategies," in *The IEEE International Conference on Intelligent Robots and Systems*, 2004.