Instance-Based Action Models for Fast Action Planning

Mazda Ahmadi and Peter Stone

Department of Computer Sciences,
The University of Texas at Austin
{mazda,pstone}@cs.utexas.edu

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 parametric models as done previously, and also incremental replanning significantly improves over original off-line planning.

1 Introduction

In many robotic applications, robots need to plan a series of actions to achieve their goals. In comparison to classical planning, planning on-board robotic agents introduces several new challenges, including (1) exceedingly noisy actions effects, often with irregular noise distributions; (2) dynamically changing environments; and (3) a need for real-time decision-making despite limited processing power. In this paper, the problem of robot action planning in dynamic environments with uncertain action effects is considered. We introduce an *instance-based* action model that captures arbitrary distributions of action effects and use it for action planning. To cope with dynamically changing environments, we introduce an efficient on-line incremental replanning method that modifies the transition model to account for the effects of other agents and then replans only for the affected states.

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 normally distributed, 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 a Markov Decision Process (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 can be run offline. In dynamic environments, the planning must be performed online. The online algorithm must be fast enough to be within computational bounds of the robots. Though fast replanning algorithms for robotic applications have been studied for classical planning problems (e.g. see [5,6]), the probabilistic replanning algorithms that we know of (e.g. [7]), are computationally expensive.

When using an instance-based approach, the observation of each dynamic factor changes the modeled transition function of the MDP. But it only changes the values 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 parametric 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.

The two main contributions of this paper are:

- An approach to dynamic replanning of action sequences based on an instance-based representation of action effects that is fully-implemented and tested on a physical robot.
- An empirical comparison of an instance-based action model and the more popular parametric action models on a physical robot.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. In Section 3 the RoboCup test-bed is presented. In the next three sections, we consider the problem as an abstract MDP. In Section 4 the details of the instance-based action model, and how to learn in it, are provided; Section 5 introduces the planning algorithm for static environments; and Section 6 extends the planning problem to dynamic environments. The implementation details of the RoboCup domain are presented in Section 7. In Section 8, experimental results are provided, and Section 9 concludes.

2 Related Work

In recent years there has been significant progress in building autonomous mobile robots. For example Burgard et al. have developed an autonomous tour guide robot [9]. Also, there have been many advances in robot soccer playing agents for the RoboCup competitions. However the focus of the research in the RoboCup 4-legged community is mainly on the lower level components such as vision, localization or locomotion. The action selection has generally been reactive or the result of shallow look-ahead. The main reason is that the action effects are highly uncertain and creating an accurate action model is challenging. In this paper using the lower level components, we tackle the problem of building an accurate action model and use that to incorpate a full-blown planning approach based on an MDP representation.

Researchers have used planning methods for robotic applications. For example Farritor and Dubowsky build a climbing robot which uses planning to find a sequence of actions which achieves its goal [10]. In earlier work Frommherz and Werling propose using heuristic search for planning and show experiments in a simple assembly task [11]. Compared to the problem considered in this paper, in the mentioned papers the action effects are not nearly as uncertain. Thus a parametric action model performs sufficiently well. Also, the environment is not dynamic, so there is no need for replanning.

Instance-based methods have been used for various fields such as health care, assessment and design [12]. Also Planning tasks have been solved with instance-based methods [12]. Atkeson [13] investigates the use of an instance-based method (locally weighted regression) to learn task models for control. Gabel and Riedmiller use an instance based method for value function approximation of a reinforcement learning problem in the soccer simulation domain [14]. Atkeson and Santamaria compare model based (using instance-based method) and model-free reinforcement learning in a simple robotic task (pendulum swing up) and conclude that model based methods learn faster [15]. Note that in the mentioned robotic applications the uncertainty of the action effects is low and also the environment is static. We take the next step of evaluating instance-based action models in a dynamic robotic environment with highly uncertain action effects.

3 Problem Description

As a test-bed domain for our research we use a subtask of the RoboCup four legged league. In this work we consider single robot goal scoring possibly against multiple opponents. We assume the opponents only block the ball, and do not kick or grab it.

As baseline software, we use the UT Austin Villa code base [16], which provides robust color-based vision, fast locomotion, and reasonable localization within a 3.6m $\times$ 5.4m area[1] via a particle filtering approach. Even so, the robot

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

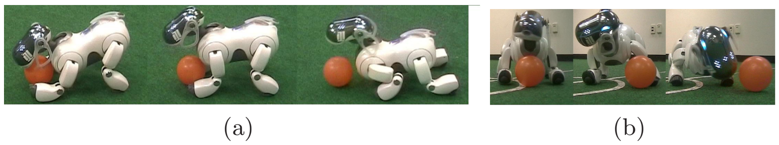

Fig. 1. (a) Representation of FALL-KICK from left to right (b) Representation of HEAD-KICK from left to right

is not, in general, perfectly localized, as a result of both noisy sensations and noisy actions effects. The robot also has limited processing power, which limits the algorithms that can be designed for it.

The baseline software provides different types of kicks. The two that are considered in this work are called FALL-KICK (see Figure 1(a)) and HEAD-KICK (see Figure 1(b)). The effects of these kicks are probabilistic based on how accurately they are executed and what exact part of the robot hits the ball.

4 Instance-Based Action Model

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

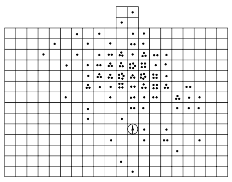

Fig. 2. Representation of the model of FALL-KICK. The position of the robot and kick direction is marked by an arrow. Each dot represents a sample action effect.

Previous methods (e.g. [8]) use parametric models of actions. Most popular methods for modeling the noise assume a Gaussian distribution of noise 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 effective than a parametric 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* noises on the action model. If the action model does not represent the noise from all sources, the effects of the environment's noise must be accounted for in some other way for action planning (e.g. an expensive way of accounting for localization errors is by planning from a group of possible positions). Instead, we aim at having noise from all sources 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 2.

5 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 Section 6.

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 discretized. We show that discretizing the state space does not have much of a detrimental effect in the RoboCup goal shooting test-bed.
- A is the set of possible actions.
- $Pr(s^{'}|s, a)$ is the true continuous state transition probability function. It gives the probability of getting to state $s^{'}$ after taking action a in state s. Because of noise in the environment and uncertainty of action effects, the transition function is stochastic
- $R(s, a) \in \mathbb{R}$ 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 the action model. For computing $\widetilde{Pr}(s^{'}|s, a)$, where s is a discrete state, a representative of s is used for computing $\widetilde{Pr}(s^{'}|s, a)$ (center of the cell in grid discretization); when s is a continuous state, the true state (TS) is used to computed $\widetilde{Pr}(s^{'}|TS, a)$. $R(s, a)$ is also computed with the help of the action model. The details of computing $\widetilde{Pr}(s^{'}|s, a)$ and $R(s, a)$ for the RoboCup goal shooting test-bed are presented in Section 7.

For state s, the value $V^{\pi}(s)$ is defined as the expected sum of rewards from s until the end of the episode, while following policy π. $V^{*}(s)$ is the optimal policy if $V^{*}(s) \geq V^{\pi}(s)$ for all policies π and all $s \in S$. $Q^{\pi}(s, a)$ is defined as the sum of the rewards received from state s until the end of the episode, while following policy π (see Equation 1), if the first action executed is a. $Q^{*}(s, a)$ is the optimal such policy. The advantage of using $Q^{\pi}(s, a)$ over $V^{\pi}(s, a)$ is presented in the next section, where only a subset of Q values needs to be updated.

We use following two equations to perform standard value iteration [17]:

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

$$V(s,a) = \max_{a \in A} Q(s,a) \tag{2}$$

When the system is in state s, a common practice for action selection is to discretize the state space (e.g. using a grid instead of the exact position) and to choose action a such that it maximizes $Q(s,a)$. Discretization makes solving the MDP tractable. But enforcing that each position in a discrete grid-cell (state) takes the same action can lead to dramatic sub-optimality. In order to alleviate this effect, we take the middle ground: for *action selection*, instead of the grid state, the robot uses its true state estimate (e.g. true position) that maximizes the following value:

$$R(TS,a) + \sum_{s' \in S} \widetilde{Pr}(s^{'}|TS,a)V^{*}(s^{'}) \tag{3}$$

where TS is the true state estimate of the robot. Note that TS is only used for action selection, not for the learning process.

This way, the effects of discretizing the state are deferred to the next step, and it results in a better policy. In [16], 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 with discretized state (grid position) the policy can be directly derived from the V-values. However with the true state, all possible actions must be evaluated, requiring significant, but manageable (on the Aibo) computational resources. In the experiments section the advantage of this action selection method is showed empirically in simulation.

6 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. Existence of dynamic factors changes the transition function of the MDP, and the value function for the new MDP needs to be computed online. Because of the robot's limited processing power and need for real-time decision making, performing full online value iteration for consideration of the dynamic factors is not possible. In this section a fast replanning algorithm is presented, which leverages the Q-values that are computed for the static environment.

If the dynamic factors are considered in computing $\widetilde{Pr}(s^{'}|s,a)$ in Equation 4, the value iteration and action selection algorithms described in Section 5 can also be applied to the dynamic environment. However dynamic factors, by their nature, are not known in advance.

Similar to $Q^{\pi}(s,a)$, $Q^{\pi}(s,a|F)$ is defined as the sum of the received reward in the presence of dynamic factors F, from state s until the end of the episode, while following policy π (see Equation 3), if the first action to execute is a. $Q^{*}(s,a|F)$ is the optimal such policy.

In the systems that we are considering, the difference between $Q^*(s,a)$ and $Q^*(s,a|F)$ is only substantial for states where F has a direct effect on $Q(s,a)$. 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 (using domain-dependent heuristics).
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 1. Notice that only one round of updates is performed. After all the Q-updates, the V values are re-computed using Equation 2.

Action selection is the same as 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 several decision cycles, so the robot does not miss action opportunities.

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

7 Implementation Details

In this section the implementation details of the instance-based action model (Section 4), planning (Section 5), and replanning (Section 6) algorithms for the goal scoring test-bed (Section 3) are presented.

7.1 Learning a 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 1(a) and 1(b)) 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 the average and standard deviation for the speed and angle of each kick to model the kick. They measure the angle and distance that the ball travels in one second after the kick. 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 (angle and distance) with Gaussian distributions.

In contrast, for creating our kick model, the robot approaches the ball, grabs it, and kicks it to the center of the field. Right before kicking, it records its position (kick position), which includes possible localization errors. Afterwards, the robot follows the ball, and when the ball stops moving, a human observer sends a signal to the robot and the robot evaluates the ball position and stores it (final ball position) [2]. 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 2 shows 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 ending up in that grid cell. The main rectangular grid represents the size of the legged soccer field.

Two fundamental differences between our model and Chernova and Veloso's [8] as well as other usual action models (e.g. [1,2]) 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 aim at designing an action model which captures the full environmental noise.

7.2 Planning

We begin by dividing the robot's environment into the disjoint grid G. Dotted lines in Figure 3 show 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 8, we empirically show that discretizing the field does not have much of a determental effect on the effectiveness of the algorithm.

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

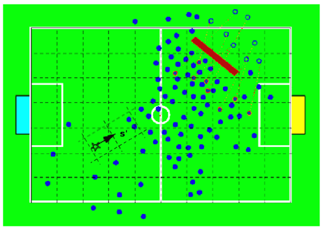

Fig. 3. Soccer field which is divided into a grid. A sample kick with the possible effects is also shown in presence of an opponent robot.

$S = G$ is a set of *states*, representing the grid cell that the ball is in. The center of a cell is assumed as the position of the grid cell. 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 (with just one kick) from state s using action a.

[2] In principle it is possible for the robot to recognize when the ball has stopped moving, but our robots do not have that capability.

The kick model (KT) is used to approximate $\widetilde{Pr}(s^{'}|s,a)$. KT 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). This new distribution is $\widetilde{Pr}(s^{'}|s,a)$. More precisely $\widetilde{Pr}(s^{'}|s,a)$ is computed using the following equation:

$$\widetilde{Pr}(s^{'}|s,a) = \frac{KT[k][s^{'}_{xd} - s_{xd}][s^{'}_{yd} - s_{yd}]}{N[k]} \tag{4}$$

where s_{xd} (s_{yd}) is the x-index[3] (y-index) of the grid cell containing the ball in s, after the transformation that transforms the kick direction to the x-axis. For example if the a is the kick action shown in Figure 3, and $s^{'}$ is the grid cell shown on the field (Figure 3), then $s^{'}_{xd} - s_{xd} = 0, s^{'}_{yd} - s_{yd} = +1$. Thus $\widetilde{Pr}(s^{'}|s,a) = \frac{KT[k][0][1]}{N[k]} = \frac{2}{118}$ 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 the action model, that is, $R(s,a)$ is equal to the percentage of the sample kick effects from the kick model that result in a goal from state s by taking action a. A line is computed from the robot position to the final kick effect's position, if that lines intersects the goal line, it is considered a goal.

The value iteration and action selection algorithms described in Section 5 are used for computing Q-values and action selection. Note that for static systems the value iterations are performed offline. Each round of value iteration on the Aibo robot's processor roughly takes 2 seconds, and each decision cycle is around 33 milliseconds. This shows why performing multiple rounds of value iteration is not feasible for dynamic systems, and therefore there is a need for a fast replanning method.

7.3 Replanning

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 in the kick model would hit any of the opponent robots, the reflection from the opponent robot is assumed by the robot as the final position of the ball. Sample kick effects of a FALL-KICK (see Figure 1(a)) in presence of an opponent robot are shown in Figure 3. The unfilled circles are the possible kick effects when the opponent does not exist.

The replanning algorithm presented in Section 6 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 for recomputation of its value if f is reachable by an average kick a from s (i.e. instead of the kick model of a, it uses the average distance and angle of kick a).[4]

[3] x-index of the grid-cells in the ith row of the grid is i.

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

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.

One round of full updates with the provided incremental planning roughly takes 1 second, and we distribute that process over 25 decision cycles. Given the fact that most of the 33 milliseconds of the decision cycle is already taken by the vision processing, the robot on average loses every other cycle while performing the incremental update (effectively the decision cycle while performing the update is closer to 66 milliseconds).

8 Experimental Results

The algorithms are evaluated both on a custom-built AIBO simulator [16] and also on the 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. [18]) 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 results in lower scoring percentages.

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

- **AtGoal:** In this algorithm, the robot always shoots directly at the goal using fall-kick which is the more accurate kick. It is used as a baseline algorithm.
- **Plan:** This is the planning algorithm (Section 5) 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.
- **RePlan:** This is the planning algorithm presented in Section 6, 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.
- **ParamPlan (ParamFullPlan):** This is similar to the Plan (FullPlan for the case of ParamFullPlan) algorithm, but instead of the full instance-based kick model, a parametric 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 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 experiment is used to show the power of the instance-based kick model compared to the parametric kick model with normalized noise.

Comparing RePlan with AtGoal shows the general effectiveness of our approach. The advantage of RePlan compared to Plan shows the benefit of the replanning algorithm. Experiments also show that performance of RePlan is close to FullPlan which highlights the effectiveness of the proposed fast replanning algorithm. Comparing RePlan with ParamPlan demonstrates the benefit of using instance-based action models.

The grid used in the experiments is 7×10 and is shown in Figure 3. The number of rounds of value iteration is set at 20.

8.1 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 5) are investigated. While doing so, we also argue that, the effects of assuming a grid for representing the position are minor.

Each episode starts with the ball positioned 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. The percentage of the episodes which resulted in goals and the average number of kicks per episode are computed for each trial. The reported data is averaged over 28 trials.

Clear Field Experiment. We start the experiments with no opponents (Figure 4). Two starting points for the ball are considered: center point (Figure 4(a)) and the upper-left point (Figure 4(b)).

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 also presented. As the results in Table 1 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 no longer considered. However, one of the benefits of the algorithms presented in this paper is that they enable the addition of newly designed kicks. All that is needed is their instance-based models.

The scoring percentage and average number of kicks per episode for AtGoal, Plan and ParamPlan are presented in Table 1. As 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 upper-left point. For the planning algorithm, the average number of kicks is also higher, which is a compromise for achieving a better scoring percentage.

The effectiveness of the instance-based kick model is shown by the significant advantage of the Plan algorithm compared to ParamPlan in Table 1. Using the instance-based kick model for the center and upper-left starting points increases the performance by 43% and 42% compared to ParamPlan, respectively.

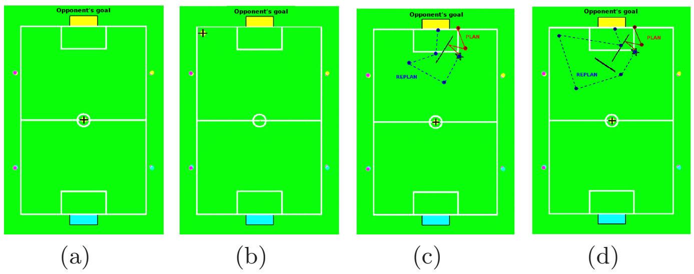

Fig. 4. Clear field. (a) The center of the field is the starting point of the ball. (b) The upper-left point is the starting point for the ball. (c) The field with one stationary opponent robot. Sample sequences for RePlan and Plan are shown for the $\star$ starting point. (d) The field with two stationary opponent robots.

Table 1. Comparing different algorithms for the two starting ball points in the clear field experiment (see Figure 4)

Starting Point: Center (Fig. 4(a))

Algorithm	AtGoal	Plan	Plan(fall-kck)	ParamPlan
Scoring%	46.2 ± 4.8	**60.5** ± 4.8	**58.7** ± 5.0	42.1 ± 5.7
Kicks/episode	3.4 ± 0.1	9.7 ± 0.9	9.6 ± 0.9	10.1 ± 0.8

Starting Point: Upper-Left (Fig. 4(b))

Algorithm	AtGoal	Plan	Plan(fall-kck)	ParamPlan
Scoring%	29.1 ± 5.0	**51.3** ± 5.4	**53.1** ± 5.3	39.0 ± 5.2
Kicks/episode	1.7 ± 0.1	6.3 ± 0.6	6.1 ± 0.8	7.1 ± 0.7

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 4(c). The ball's starting point is at the center of the field. The algorithms AtGoal, RePlan, FullPlan, Plan, and ParamFullPlan are compared. Success percentage and average number of kicks is presented for the above-mentioned algorithms in Table 2.

The RePlan algorithm significantly improves scoring percentage compared to AtGoal, Plan and ParamFullPlan by 104%, 13%, and 45% respectively. RePlan is also very close in performance to FullPlan (non-significant difference of 1.5%), where the transition function is assumed to be known a priori and the Q-values are computed offline without computational limitations. The

Table 2. Scoring percentage and average number of kicks for different algorithms, in the one opponent robot scenario (see Figure 4(c))

Algorithm	AtGoal	RePlan	FullPlan	Plan	ParamFullPlan
Scoring %	32.00 ± 5.83	**65.92** ± 4.20	**67.10** ± 3.92	57.42 ± 3.76	45.17 ± 4.59
# Kicks/episode	1.77 ± 0.11	11.74 ± 1.00	8.22 ± 0.67	8.42 ± 0.63	9.59 ± 0.69

average number of kicks per episode is the most for the RePlan algorithm (see Figure 4(c) for a sample kick sequence for RePlan), but in this domain, scoring efficiency is of greater importance.

Two Opponents Experiment. In this experiment, an additional robot is positioned on the field. The field is shown in Figure 4(d). The scoring rate and average number of kicks for different algorithms is presented in Table 3. The trend in the result is consistent with the observation in the one opponent robot scenario in the previous experiment (Section 8.1).

Table 3. Comparing different algorithms for playing against the two opponents case (Fig. 4(a))

Algorithm	AtGoal	RePlan	FullPlan	Plan	ParamFullPlan
Scoring %	21.85 ± 4.64	**54.25** ± 5.58	**54.64** ± 5.59	46.46 ± 5.10	38.07 ± 4.39
# Kicks/episode	5.09 ± 0.32	19.45 ± 1.97	11.43 ± 0.90	13.04 ± 1.14	11.84 ± 0.96

Moving Opponents Experiment. In this experiment two *moving* opponent robots are present on the field. In each episode, the opponent robots start from the position of the opponent robots in the previous experiment (Figure 4(d)), 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 4. The performance of 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.

8.2 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 many trials as in simulation. First, experiments in the clear field case (Figure 4(b)) and then against two opponent robots (Figure 4(d)) are described.

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

Table 4. Comparing AtGoal, Plan and RePlan algorithms in the presence of two moving opponent robots

Algorithm	AtGoal	RePlan	Plan
Scoring %	20.60 ± 5.14	**57.32** ± 4.82	54.14 ± 3.98
# Kicks/episode	4.52 ± 0.37	9.51 ± 0.65	9.51 ± 0.67

Real Robot on a Clear Field. The configuration is the same as the one shown in Figure 4(b). Each trial consists of 50 episodes, and the result for one trial is reported in Table 5. The trend is similar to the simulation experiment of the same configuration, and Plan increases performance over AtGoal and ParamPlan by 80% and 20% respectively.

Table 5. Comparing different algorithms for upper-left starting ball points for the clear field experiment (see Figure 4(b)) with a real robot

Algorithm	AtGoal	Plan	ParamPlan
Scoring %	20	36	30

Real Robot Against Two Opponent Robots. In this experiment the configuration of the field in Figure 4(d) is used for a real robot. Each trial consists of 25 episodes, and the results are reported in Table 6.[6] The results show the same trend as the simulation on this field: RePlan is better than Plan and AtGoal.

Table 6. Comparing different algorithms for real robot in the experiment against two opponent robots (see Figure 4(d))

Algorithm	AtGoal	Plan	RePlan
Scoring %	4	16	24

9 Conclusion

This paper considers the action planning problem in noisy environments, modeling it as an MDP. An instance-based action model is introduced to model noisy action effects. The action model is then used to build the transition function of a 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 evaluate these approaches, goal scoring in the RoboCup 4-legged league is used as a test-bed. The experiments show the advantage of

[6] Since in base code used, the robots do not walk around opponent robots, in this experiment whenever the robot attempts to walk through the opponent, the human temporarily removes the opponent robot. Were the robot equipped with obstacle avoidance, it would score in roughly the same trials — it would just take a bit longer.

using an instance-based approach compared to parametric action models. They also show that the fast replanning algorithm outperforms off-line planning and approaches the best possible performance assuming the full transition model is known a priori.

In future work, the performance of the incremental replanning algorithm may be improved by learning the effects of dynamic factors on the transition model. Also, extending this approach to team behaviors, where the value of each state also depends on the positions of teammates, is an interesting direction for future consideration.

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. Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples with incomplete knowledge. In: Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (June 2005)
2. Wang, X.: Learning planning operators by observation and practice. In: Artificial Intelligence Planning Systems, pp. 335–340 (1994)
3. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning 13, 103–130 (1993)
4. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
5. Stentz, A.T.: The focussed d* algorithm for real-time replanning. In: Proceedings of the International Joint Conference on Artificial Intelligence (August 1995)
6. Koenig, S., Likhachev, M.: Improved fast replanning for robot navigation in unknown terrain. In: Proceedings of the 2002 IEEE International Conference on Robotics and Automation (May 2002)
7. Wilkins, D.E., Myers, K.L., Lowrance, J.D., Wesley, L.P.: Planning and reacting in uncertain and dynamic environments. Journal of Experimental and Theoretical AI 7(1), 197–227 (1995)
8. Chernova, S., Veloso, M.: Learning and using models of kicking motions for legged robots. In: Proceedings of International Conference on Robotics and Automation (ICRA 2004) (May 2004)
9. Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S.: Experiences with an interactive museum tour-guide robot. Artificial Intelligence 114(1-2), 3–55 (1999)
10. Bevly, D., Farritor, S., Dubowsky, S.: Action module planning and its application to an experimental climbing robot. In: International Conference on Robotics and Automation (2000)

11. Frommherz, B., Werling, G.: Generating robot action plans by means of an heuristic search. In: International Conference on Robotics and Automation (1990)
12. Leake, D.B. (ed.): Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press, Menlo Park (1996)
13. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. Artificial Intelligence Review 11(1-5), 75–113 (1997)
14. Gabel, T., Riedmiller, M.: Cbr for state value function approximation in reinforcement learning. In: 6th International Conference on Case-Based Reasoning (2005)
15. Atkeson, C., Santamaria, J.: A comparison of direct and model-based reinforcement learning. In: International Conference on Robotics and Automation (1997)
16. Stone, P., Dresner, K., Fidelman, P., Kohl, N., Kuhlmann, G., Sridharan, M., Stronger, D.: The UT Austin Villa 2005 RoboCup four-legged team. Technical Report UT-AI-TR-05-325, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory (November 2005)
17. Puterman, M.L.: Markov Decision Processes. Wiley, NY (1994)
18. Kwok, C., Fox, D.: Reinforcement learning for sensing strategies. In: The IEEE International Conference on Intelligent Robots and Systems (2004)