Reinforcement Learning for Sensing Strategies

Cody Kwok and Dieter Fox

University of Washington, Computer Science & Engineering, Seattle, WA

***Abstract*— Since sensors have limited range and coverage, mobile robots often have to make decisions on where to point their sensors. A good sensing strategy allows a robot to collect information that is useful for its tasks. Most existing solutions to this *active sensing* problem choose the direction that maximally reduces the uncertainty in a single state variable. In more complex problem domains, however, uncertainties exist in multiple state variables, and they affect the performance of the robot in different ways. The robot thus needs to have more sophisticated sensing strategies in order to decide which uncertainties to reduce, and to make the correct trade-offs. In this work, we apply a least squares reinforcement learning method to solve this problem. We implemented and tested the learning approach in the RoboCup domain, where the robot attempts to reach a ball and accurately kick it into the goal. We present experimental results that suggest our approach is able to learn highly effective sensing strategies.**

I. Introduction

To successfully operate in its environment, a robot needs to know the states of the objects that are relevant for its tasks. For example, an office delivery robot needs to know its own location in the environment, the location of the letter it is supposed to deliver, and whether the door to the office containing the letter is open or closed. These states have to be estimated from noisy sensor information. Bayes filters such as Kalman filters or particle filters integrate sensor information over time in order to estimate such states [1], [6].

Since sensors only have limited range and coverage, a robot has to decide where to point its sensors so as to collect information that is most valuable for its task. Bayesian techniques typically address this problem by choosing the sensing action that minimizes the expected uncertainty of the state, where uncertainty is measured by the entropy of the posterior probability distribution over the state space. Typically, these active sensing techniques consider uncertainty in only one state variable (*e.g.*, robot location [5], [11] or object position [4]). In many situations, however, multiple relevant state variables carry uncertainty and the sensor cannot collect information about all of them at the same time.

To see, consider the problem of scoring a goal in the RoboCup domain. In order to kick the ball into the goal, a robot has to know where the ball is and in which direction the goal is located. Since the robot cannot observe the ball and the goal at the same time, it has to decide whether to look at the ball in order to minimize uncertainty in the ball location, or to point the camera in a direction that helps it to determine the relative location of the goal. Obviously, there is a trade-off, since looking at the goal increases uncertainty in the ball location and vice versa. Furthermore, it is not clear which uncertainty should be reduced at any point in time, since there is a highly non-linear relationship between the uncertainties and the success of the goal kick. While in principle, techniques such as POMDPs [10] and augmented MDPs [11] are well suited to address such problems, they do not scale to highly complex state spaces and they assume that a good model of the robot and its environment is available.

Fig. 1: The RoboCup domain is highly dynamic. We introduce reinforcement learning to develop sensing strategies that decide where a robot should point its camera so as to best score a goal.

In this paper we show how to solve such multi-state active sensing problems using reinforcement learning. More specifically, we use least squares policy iteration (LSPI) to learn a robot's sensing strategy. LSPI has been introduced recently for highly efficient model-free reinforcement learning [9]. To take uncertainties in the state estimation into account, we augment the state space such that it explicitly contains the uncertainties in the different state variables. The approach is implemented and tested in the context of RoboCup. The legged AIBO robot is able to learn a sensing strategy that results in superior performance in a goal scoring task.

This paper is organized as follows. After discussing related work, we review the basics of reinforcement learning and least squares policy iteration in Section III. Then, in Section IV, we show how to apply LSPI to our active sensing problem. Section V presents experimental results, followed by our conclusions.

II. Related Work

Active sensing has been applied in various areas ranging from robot localization [5] to vision [4] to sensor management for target tracking [7]. Most existing approaches use information theoretic measures of uncertainty and choose the sensing action that minimizes expected uncertainty (*e.g.*, in robot or target location). While such greedy approaches are very efficient, they do not consider the long term impact of sensor information. Furthermore, these techniques typically consider uncertainty in one state variable only and are not directly applicable to the problems we consider in this work.

In principle, partially observable Markov decision processes (POMDPs) are well suited to overcome these limitations. POMDPs generate sensing and action policies that

take uncertainty into account. However, even recent solutions to POMDPs are intractable for realistic problems [10]. Augmented MDPs provide an alternative approximation to POMDPs [11]. The augmented state space of such Markov decision processes contains uncertainty variables in addition to the original state variables. The uncertainty variables represent the uncertainty (measured in entropy) in the original state variables. Such MDPs have been applied successfully to the problem of robot path planning under uncertainty in a robot's position [11]. However, their work assumes the availability of an accurate model of a robot's actions and observations. Furthermore, they rely on a discretization of the augmented state space, a representation that does not scale to higher dimensional problems such as ours. Our technique overcomes these limitations by using reinforcement learning on the augmented state space in combination with efficient linear function approximation.

The technique most closely related to our work is the one proposed by Busquets and colleagues [3]. They apply reinforcement learning to minimize the travel distance and camera usage as a robot moves to a target. A very coarse grid is used to represent the state space and associated uncertainties. Even though their technique worked well in simulated scenarios, the discretization requires careful hand tuning and it is not clear how the technique can be applied to more noisy and dynamic environments such as ours.

III. Least Squares Reinforcement Learning

In this section, we shall review the basics of reinforcement learning using least squares policy iteration; see [12], [9] for details. Reinforcement learning provides a framework by which an agent can learn control policies based on experience and rewards. The concept underlying reinforcement learning is that of a Markov decision process (MDP).

An MDP models an agent acting in an environment as a tuple $\langle S, A, P, R\rangle$, where S is the set of states, and A is a finite set of actions. $P(s'|s,a)$ is the transition model that describes the probability of ending up in state s' after executing action a in state s. $R(s,a,s')$ is the reward obtained when the agent executes action a in state s and transitions to s'. The goal of solving an MDP is to find a policy, $\pi : S \mapsto A$, that maps states to actions such that the agent's cumulative future reward is maximized. If the parameters of P and R are known, then the optimal control policy for the agent can be determined efficiently using techniques such as value iteration.

If the exact parameters of an MDP are not known, then reinforcement learning can be used to determine the optimal control policy. There are two different families of reinforcement learning techniques, model-based and model-free. The first class of techniques aims at learning the parameters of the transition and reward functions, P and R, and then uses standard MDP solution techniques to determine the optimal policy. P and R are learned by observing the agent acting in the environment. Model-free approaches, on the other hand, do not attempt to learn the model parameters P and R, but rather aim at learning a policy directly.

Here, we apply least squares policy iteration (LSPI), a recently introduced approach to efficient, model-free reinforcement learning (see [9] for details). Like SARSA(λ), LSPI aims at learning a policy π that maximizes the corresponding Q-function, where $Q^\pi(s,a)$ is the expected cumulative reward of executing action a in state s, assuming that the agent will follow the policy π in the future. For a given policy, the values of the Q-function are given by solving the following Bellman equations:

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

The first term on the right hand side of (1), $r(s,a)$, is the expected reward of executing action a in s. It is given by averaging over the possible next states: $r(s,a) = \sum_{s'} P(s'|s,a) R(s,a,s')$. The second term describes the expected future reward when following policy π, where $\pi(s') = \operatorname{argmax}_a Q^\pi(s',a)$. The discount factor γ models the fact that future reward is less valuable than immediate reward.

Policy iteration is an iterative procedure for learning the optimal policy, π^*. Denote by π^t the policy at the beginning of the t-th iteration of the learning procedure. In the value determination step, the agent interacts with the environment using the policy $\pi^t(s)$ to determine its actions. These interactions are used to estimate the Q-function of this policy, denoted Q^t. In the following policy improvement step, Q^t is used to generate a new, improved policy π^{t+1}, where $\pi^{t+1}(s) = \operatorname{argmax}_a Q^t(s,a)$. It can be shown that by iterating through these two steps until convergence, the algorithm learns the optimal policy, that is, the policy that maximizes the cumulative, expected reward.

A. Estimation of the Linear Q-function

A key step in policy iteration is the value determination step, which estimates the Q-function (1) for a fixed policy π. As a model-free approach, LSPI does not explicitly estimate P and R, it rather estimates the Q-function directly from interactions with the environment. In order to avoid the complexity of grid-based approximations, LSPI uses a linear function, $\widehat{Q}^\pi(s,a)$, to approximate the Q-function:

$$Q^\pi(s,a) \;\approx\; \widehat{Q}^\pi(s,a) \;=\; \sum_{i=1}^{k} \phi_i(s,a)\, w_i \;=\; \phi(s,a)^\mathsf{T} w \qquad (2)$$

The k features, $\phi_i(s,a)$, represent information extracted from the state-action pairs; they are designed manually. The weights w_i are the parameters of the linear function.

LSPI learns the weights of the linear function approximation using temporal difference learning. More specifically, each interaction between the agent and the environment provides a sample $\langle s,a,r,s'\rangle$, which describes the reward r received upon executing action a in state s and ending in state s'. The sequence of tuples is then used to update a $k \times k$ matrix, A, and a k-dimensional vector, b, as follows:

$$A \;\leftarrow\; A + \phi(s,a)(\phi(s,a)^\mathsf{T} - \gamma \phi(s', \pi(s'))^\mathsf{T}) \qquad (3)$$

$$b \;\leftarrow\; b + \phi(s,a)\, r \qquad (4)$$

Initially, both A and b are set to zero. In essence, the algorithm "remembers" the experience the agent has seen so far

using the A matrix, and records the rewards in b. At any time, an estimate of the weights of the linear function $\widehat{Q}^{\pi}(s,a)$ can be extracted from A and b by solving the system $w = A^{-1}b$. Lagoudakis and Parr show that these estimates converge to the optimal weights of the linear function approximation as the number of samples increases [9]. An advantage of this technique over traditional approaches, such as gradient-descent TD(λ) or SARSA(λ), is that it converges faster with less samples, since the samples are used very efficiently. This data efficiency comes from two properties of LSPI. First, least squares learning evaluates the policy with a single pass over the set of samples, whereas gradient-descent algorithms require multiple passes or many more samples. Moreover, LSPI is an off-policy algorithm, which separates the sample-generating policy from the policy under evaluation. This makes it possible to use the same set of samples to evaluate and improve the policy on each iteration, until the algorithm converges to a fixed point policy [9]. In practice, more than one set of samples may be necessary; however, LSPI retains data efficiency by improving the policies as much as possible given the available samples. Also, LSPI does not require a carefully chosen step size parameter, as in the traditional approaches. Computationally, LSPI converges quickly when k is not extremely large, as pointed out in [2].

B. Least Square Policy Iteration Algorithm

The estimation of the Q-function discussed above plays the role of the policy evaluation step in policy iteration. Whenever the weights of the approximation are updated, policy improvement is done easily by selecting for each state the action with the highest Q-value:

$$\pi(s|w) = \underset{a}{\operatorname{argmax}}\, \phi(s,a)^{\mathsf{T}}\, w \tag{5}$$

To summarize, LSPI iterates between policy estimation and policy improvement as follows. The learner collects samples of the form $\langle s, a, r, s' \rangle$ by interacting with the environment. These samples are used to update the matrix A and the vector b using the update equations (3) and (4), respectively. After all the samples are processed, inversion of A and multiplication with b yields the weights w for the new linear Q-function (policy evaluation step). This Q-function estimate is then used to update the policy according to (5) (policy improvement step). This process then repeats with the improved policy, until a fixed point policy is reached, *i.e.*, the weights of policies between successive iterations do not differ significantly.

IV. Reinforcement Learning for Active Sensing

A key limitation of MDPs is that they assume full observability, which means the agent can always observe the true state of the world. The only uncertainty is in the outcome of actions. Unfortunately, the assumption of full observability is violated in most applications, since a robot can observe the environment only through its noisy sensors. While POMDPs extend MDPs to partially observable environments, they are intractable for all but rather small problems [10]. Augmented MDPs [11] extend MDPs by adding information-theoretic measures of uncertainty to the state space. Augmented MDPs have the key advantage that they are tractable and consider uncertainty in the state space. In this section we will show how to apply reinforcement learning to augmented MDPs in the context of active sensing in the RoboCup domain.

A. Task Description

We illustrate and evaluate our approach to active sensing using the task of kicking a ball into the opponent's goal. Our legged AIBO robots use detections of field markers, goals, the ball, and other robots to estimate their own location on the field, the location of the ball, and the location of other robots. Individual detections are integrated over time using a Rao-Blackwellised particle filter, which represents the robot location by particles and the ball location by multi-model Kalman filters [8]. State estimation on legged robots is complicated by very noisy odometry and observations. Motion is highly imprecise, and motion errors are often biased in some way due to unbalanced wear and tear on the leg joints. Thus a robot must frequently observe landmarks if it needs reliable location estimates. Furthermore, observations are very noisy due to low resolution images that are often blurred by the bumpy locomotion of the robot.

To kick the ball into the goal, a robot needs to know the relative position of the ball and its own position relative to the goal. Unfortunately, the robot's camera has a limited field of view, so that it can only observe a subset of important features at a time. For example, to estimate the location of the ball, the robot has to point its camera at the ball. However, the robot cannot observe the goal at the same time, hence it becomes more uncertain in the relative location of the goal as it moves toward the ball. On the other hand, to estimate the location of the goal, the robot has to point the camera at the goal or a field marker, which increases the risk of losing track of the ball. Obviously, there is a trade-off between reducing uncertainty in the different state variables. Furthermore, the system is highly non-linear and it is not clear how the different uncertainties affect the outcome of the actions, for example, of grabbing the ball or kicking it toward the goal.

B. Augmented State Space

In order to apply reinforcement learning to the active sensing problem, we add uncertainty variables to the state space of the robot. Our state space is thus $S = \langle S_X, S_H \rangle$, where S_x is the set of variables associated with the world states, and S_H is the set of uncertainty variables. For the goal kicking task, $S_X = \langle d_b, \theta_b \rangle$ and $S_H = \langle H_r, H_b, H_{\theta_g} \rangle$. The three components in S_X are the distance to the ball, the relative angle of the ball, and the relative angle of the goal, respectively (see Fig. 2(a)). The three components in S_H describe the uncertainties in the robot, ball, and goal location. The uncertainties, measured by entropy, are extracted from the Rao-Blackwellised particle filter estimating the robot and ball location [8].

To reduce the complexity of the learning problem, we decouple the active sensing problem from the motion control of the robot. Our control strategy uses the mean estimates of the robot, ball, and goal position to guide the robot to the

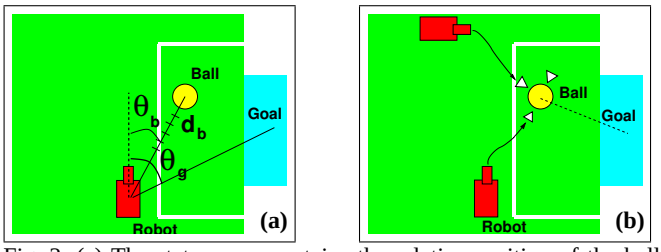

Fig. 2: (a) The state space contains the relative position of the ball and the relative orientation of the goal. The distance to the ball is discretized using tile coding. (b) The motion controller uses the estimated ball and goal location to guide the robot to one of three possible ball docking locations. Depending on the docking location, the robot chooses a straight kick or a left/right head kick.

ball so that it can kick the ball toward the goal (this control strategy has been developed beforehand, see Fig. 2(b)). The learning task is to generate a sensing policy that determines at which object the robot should look. A head controller will then move the head to the most likely position where the robot will see the object. There are nine sensing actions, pointing either at the ball, the six field markers, or the two goals. Note that executing these actions can often have unpredictable results. The robot may fail to see the object because it has an incorrect location estimate in the case of landmarks, or when the ball has moved. Multiple objects can also be seen, further complicating the learning task.

As described in Section III, least squares policy iteration approximates the Q-function by a linear function of features extracted from the state space. In our case, we use a mixed discrete / continuous representation of the state space. The Q-function is learned for each of the nine actions independently. Furthermore, the distance to the ball, d_b, is discretized into six overlapping intervals. The first interval covers distances below 20cm and the last interval covers distances above 60cm. The other four intervals evenly split the distances between 20cm and 60cm [1]. For each of the resulting 54 combinations (9 actions $\times$ 6 distances) we use the following feature vectors

$$\phi(s,a) = \langle |\theta_b|, H_r, H_b, H_{\theta_g}, |\theta_n|, 1 \rangle. \quad (6)$$

Thus the total number of variables is 324. The different components of $\phi(s,a)$ are as follows:

- $|\theta_b|$: the absolute-valued orientation of the ball relative to the robot. We assume symmetry w.r.t. this angle.
- H_r: the entropy of the robot's location estimate is extracted from the particle filter [8]. To do so, the samples of the particle filter are put in a 3d grid, and if p_j is the sum of the weights in cell j, then the entropy H_r is computed by $-\sum_j p_j \log p_j$.
- H_b: the entropy of the ball estimate is extracted in closed-form from the ball Kalman filters [8], [1].
- H_{θ_g}: the entropy of the robot's orientation towards the goal. Extracted from the particle filter by generating a grid over the robot's orientation to the goal.
- $|\theta_n|$: the angular difference between the pan of the current neck position and the target neck position of a. This value encodes the cost of the sensing action.
- 1: a constant that represents the feature of selecting action a for the current distance interval.

Like d_b, $|\theta_b|$ captures information about the position of the robot relative to the ball. The entropies represent the uncertainties to be traded off. They are balanced against the cost of an action, which is the angular distance the neck has to travel for the action. One key advantage of representing the active sensing problem with linear approximation, besides lower complexity, is that the importance of each factor involved in the trade off can be represented in a simple and intuitive manner by a combination of linear weights. With a grid/tabular approach, it is not clear how the variables should be discretized, and some finer distinctions may require very fine grids, which increases the size of the state space.

The reward structure and learning procedure are described in more detail in the next section.

V. Experiments

We performed experiments both in a simulated environment and on real robots.

A. Simulated Experiments

Since it is expensive to obtain data from a real robot, we learn a policy using our AIBO robot simulator. The simulator models noise in sensing and robot motion at the level of individual observations and fine-grained motion commands. Furthermore, it takes constraints such as maximum velocities and limited camera view into account. The simulator uses a model of the robot body to determine when the robot touches or can grab/kick a ball. On top of the simulator runs the complete control software, including the Rao-Blackwellised particle filter [8] and the motion control described above.

At the beginning of a training episode, we place the robot and the ball at random locations on the field. The robot then attempts to move to ball, grab it, and kick it into the goal. The robot makes a sensing decision and executes it every 0.5 second. The episode completes when one of the following four cases happen:

1) The ball is kicked into the goal: **reward** +4.
2) The robot kicks the ball but misses the goal. The reward is given by a linear function of the distance from the goal (measured where the ball hits the border on the goal side). The function gives a maximum **reward** of 1.5 immediately next to the goal and reaches a minimum **reward** of 0.1 50cm away from the goal.
3) The robot fails to grab the ball by accidentally touching it with its legs: **reward** -5.
4) The robot loses track of the ball: **reward** -5.

Additionally, for each time period $\triangle t$ elapsed, a reward of -0.05 is given as we want the sensing strategy to support a fast approach to the ball.

While it is possible to generate a single set of samples for LSPI to train with, we find it more data efficient to generate samples with improved intermediate policies. We train LSPI in batches of 50 episodes. After each batch, we run LSPI

[1] We use tile coding (CMACs) [12] with two overlapping tilings offset by 5cm. The reason for the discretization is that d_b greatly affects the trade-offs between state variables, and the non-linear influence is best captured by learning a different linear function for each group.

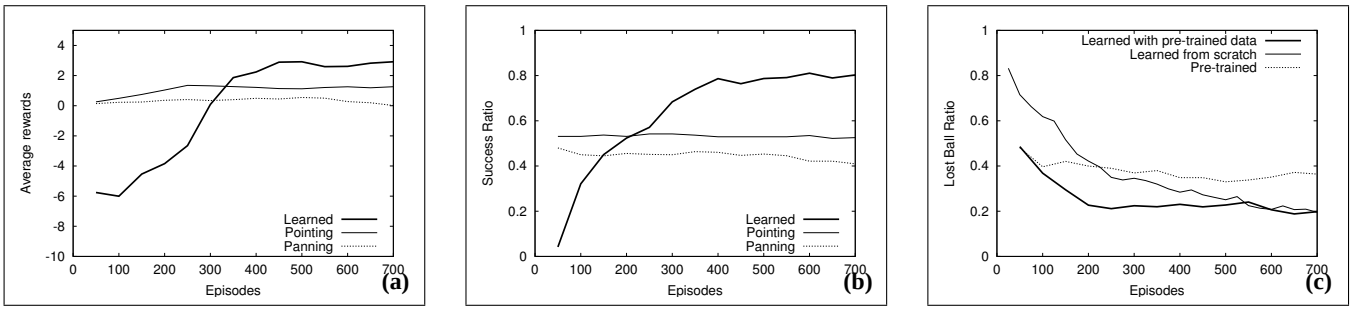

Fig. 3: (a) Average rewards during learning. The thick line indicates the rewards for each each batch of 50 episodes. The thin lines provide results for hand tuned policies. (b) Success ratio: Fraction of goals scored per episode. (c) Fraction of failed ball tracking during episodes with opponent robot. The thick line gives results when the robot starts with the policy learned without considering other robots.

until it reaches a fixed point policy, using *all* the samples generated so far. LSPI is stopped when the difference between the weights of two consecutive policy evaluations falls below 0.01, which usually takes less than 10 iterations. Then we generate new samples with the improved policy and repeat. The first 50 episodes are generated using a random policy, but later episodes use ϵ-greedy policies. The discount factor γ is set to 0.9.

The evolution of the average reward during a learning run is given by the thick line in Fig. 3(a). A different evaluation of this experiment is given in Fig. 3(b). Shown there is the success ratio measured by fraction of goals scored. During the first 200–300 episodes, we observed that the robot learns to focus on the ball in order to avoid penalties associated with losing track of the ball and improper docking. After that, it learns to look at different landmarks to improve the goal scoring accuracy. Eventually, it converges at the best trade-off between the different sensing actions, resulting in a success rate of 80% (Fig. 3(b)).

To gauge the quality of the learned policy, we compared it to two hand-tuned sensing strategies. The *panning* policy focuses the camera on the ball, and performs a 180° head sweep to look at all markers in its visible range whenever the goal orientation uncertainty is above a certain threshold. The *pointing* policy also focuses the camera on the ball, but looks up to a single marker when the goal orientation uncertainty threshold is exceeded. The choice of marker is made according to the relative angle of the marker from the robot. The robot switches between looking at the closest and the second closest marker, which increases the information gained from each individual detection. The choice of uncertainty thresholds are tuned to give optimal performance for the respective policies. To improve docking, both policies constantly look at the ball once the robot is less than 30cm away from it. The rewards and success ratios of the panning and pointing policies are indicated by the thin dashed and solid lines in Fig. 3(a) and (b), respectively. Obviously, our approach is able to learn a superior sensing strategy.

An analysis of the learned policy showed that it captures the resilient features of the hand tuned policies, and improves upon them. Most of the time the robot chooses to look at the ball, because this action gives it the best estimate of where the ball is and allows it to walk the shortest path to the ball. It also looks at various landmarks from time to time. The choice of landmarks appears to reflect the shifting importance of H_r to H_{θ_g}. Initially, H_r is important because if the robot is lost, it cannot predict the location of landmarks. Later, when the robot is close to the ball, it only needs to be sure where the goal is in order to score, so it frequently looks at the goal instead. The robot often looks at a landmark one last time at 25-30cm from the ball to enable a final adjustment, and focuses solely on the ball afterward for successful docking.

Goal scoring with opponent robot: In this set of experiments we increase the difficulty of the task by adding an opponent robot. A randomly placed opponent moves straight to the ball and kicks it as soon as it reaches it. Even though the robot is not able to reach the ball if the opponent is closer, it is clear that it should avoid looking away from the ball when the opponent reaches the ball. This way, it is much less likely to lose track of the kicked ball. For this task, we add the following features to the feature vector $\phi(s, a)$:

- v_b: velocity of the ball.
- o_d: distance of closest **d**etected opponent.
- o_u: minimal distance an **u**ndetected opponent must be away from the ball.

The additional features o_d and o_u represent information about how close to the ball an opponent can be. o_d represents *positive* detections; if the robot detects the opponent when it looks at the ball, then it can estimate how close the opponent is to the ball. This distance is stored in o_d. If the robot looks at the ball and does not detect any opponent in the same camera frame, then it knows that all opponents must be at least a certain distance away from the ball (estimated from the field of view). This *negative* information is stored in o_u. If the robot looks at a landmark, then both distances decrease according to the maximum velocity an opponent can move. Thus, if any of these two distances becomes small, the robot should look at the ball in order to avoid losing track of it.

The ratio of episodes the robot loses track of the ball when using the policy learned without considering the opponent is indicated by the dashed, thin line in Fig. 3(c). The robot loses track of the ball in approximately 40% of the episodes. We can use this policy at the beginning of learning a new, adapted policy (in essence, we initialize the A and b values). As is shown by the thick line in Fig. 3(c), the robot learns to use the additional information so as to improve its performance under these new circumstances. The resulting policy loses track of the ball in only half as many episodes. The solid, thin line in

	Goals	Misses	Average Miss Distance	Kick Failures
Learned	31	10	$6 \pm; 0.3$ cm	4
Pointing	22	19	$9 \pm; 2.2$ cm	4
Panning	15	21	$22 \pm; 9.4$ cm	9

Table 1: Results of the various policies on the ball scoring task, using the real robots. 45 episodes were performed for each policy.

the figure gives the learning curve when starting from scratch. Obviously, the learning process is sped up significantly when starting with prior experience (thick line). For clarity, we omit the hand-tuned policies, since their performance is much inferior to all learned policies. An analysis of the policy showed that the robot learned to always look at the ball when it knows that an opponent is near (o_d small). If it only knows that an opponent *might* be near, then it occasionally looks at landmarks and the goal (o_u small).

B. Real Robot Experiments

After learning the sensing policy on the simulator, we tested it on the real robots. The task is again to move to the ball and kick it into the goal. For each policy (learned and two hand-tuned), we performed 3 batches of 15 episodes, each run based on one of 3 selected robot starting positions and one of 5 ball positions on the field (all positions were within 2m of the goal). The results are summarized in Tab. 1. These results are very similar to the simulated experiments, except the success rate is slightly lower, with the 69% success rate for the learned policy here versus the 80% achieved in simulation. This difference is not surprising since the simulation does not model uneven ground, which makes the real ball move on non-straight trajectories. However, the learned policy is still significantly better than the hand-tuned policies, even though these policies were additionally tuned for the real robot.

The third column shows that even when the ball misses the goal, on average it ends up closer to the goal with the learned policy. While the robot never loses the ball in these trials, the kick failures for both the learned and the pointing head strategy are caused by errors in executing the kicks, which happen probabilistically.

VI. Conclusions and Future Work

We showed how to learn active sensing strategies for a mobile robot using least square policy iteration. LSPI is an efficient, model-free reinforcement learning technique for continuous state spaces. To apply LSPI to the active sensing problem, we used an augmented state representation. In addition to the state variables a robot needs to consider for its control, the augmented state space contains the uncertainties in these variables. As a result, the robot can learn which uncertainties it should reduce to achieve best performance. This technique is not limited to our domain, but also generalizes to other problems. The augmented state space model offers an effective approximation to partially observable problems. The main design effort is to identify the relevant uncertainties in the problem and encode them in the state space. Once the augmented representation is available, LSPI can be applied to learn a policy efficiently.

We implemented and tested our approach in the context of learning a sensing strategy that enables a legged robot to score a goal in the RoboCup domain. Both simulated and real robot experiments show that our learned sensing strategy significantly outperforms two hand tuned strategies. Our technique learned the best trade-off between minimizing the different uncertainties involved in the task. Furthermore, the technique was able to adapt to situations in which an opponent kicks the ball before the robot can reach it. The learned strategy reflects the intuition that the robot should look at the ball as the opponent approaches it, thereby avoiding to lose track of it when it is kicked by the opponent.

These results are extremely encouraging and we are convinced that our technique is applicable to a wide range of active sensing problems. There are various directions for future research. Currently, we assume that there already exists a fixed motion policy. Our next step will be to add robot motion to the action space of the robot, thereby enabling the approach to learn joint motion and sensing strategies. So far, reinforcement learning was done in simulation only. While this approach worked well due to the high quality of our simulator, it is certainly advantageous to further improve a policy using a real robot. An interesting question in this context is how the simulation experience should be weighed relative to the robot's real world experience. Finally, we will try to scale reinforcement learning for active sensing to more complex scenarios involving multiple robots.

Acknowledgments

This work has partly been supported by the NSF under grant number IIS-0093406.

References

[1] Y. Bar-Shalom, X.-R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley, 2001.

[2] Justin A. Boyan. Least-squares temporal difference learning. In *Proc. of the International Conference on Machine Learning, 1999*.

[3] D. Busquets, R.L. de Màntaras, C. Sierra, and T.G. Dietterich. Reinforcement learning for landmark-based robot navigation. In *Proc. of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002.

[4] J. Denzler and C.M. Brown. Information theoretic sensor data selection for active object recognition and state estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(2), 2002.

[5] D. Fox, W. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

[6] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3), 2003. Special Issue on Dealing with Uncertainty.

[7] C. Kreuchner, K. Kastella, and A.O. Hero. Sensor managment using an active sensing approach. *IEEE Transactions on Signal Processing*, 2004. Special Issue on Machine Learning, to appear.

[8] C.T. Kwok and D. Fox. Map-based multiple model tracking of a moving object. In *Proc. of RoboCup International Symposium*, 2004.

[9] M.G. Lagoudakis and R. Parr. Model-free least squares policy iteration. In *Advances in Neural Information Processing Systems 14*, 2001.

[10] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems 15*, 2002.

[11] N. Roy and S. Thrun. Coastal navigation for mobile robots. In *Advances in Neural Information Processing Systems 12*, 1999.

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