Closing the Learning-Planning Loop with Predictive State Representations

Byron Boots
Machine Learning Dept.
Carnegie Mellon University
Pittsburgh, PA, USA
Email: beb@cs.cmu.edu

Sajid M. Siddiqi
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA
Email: siddiqi@google.com

Geoffrey J. Gordon
Machine Learning Dept.
Carnegie Mellon University
Pittsburgh, PA, USA
Email: ggordon@cs.cmu.edu

***Abstract*— A central problem in artificial intelligence is to choose actions to maximize reward in a partially observable, uncertain environment. To do so, we must *learn* an accurate model of our environment, and then *plan* to maximize reward. Unfortunately, learning algorithms often recover a model which is too inaccurate to support planning or too large and complex for planning to be feasible; or, they require large amounts of prior domain knowledge or fail to provide important guarantees such as statistical consistency. To begin to fill this gap, we propose a novel algorithm which provably learns a compact, accurate model directly from sequences of action-observation pairs. To evaluate the learner, we then *close the loop* from observations to actions: we plan in the learned model and recover a policy which is near-optimal in the *original* environment (not the model). In more detail, we present a spectral algorithm for learning a Predictive State Representation (PSR). We demonstrate the algorithm by learning a model of a simulated high-dimensional, vision-based mobile robot planning task, and then performing approximate point-based planning in the learned model. This experiment shows that the learned PSR captures the essential features of the environment, allows accurate prediction with a small number of parameters, and enables successful and efficient planning. Our algorithm has several benefits which have not appeared together in any previous PSR learner: it is computationally efficient and statistically consistent; it handles high-dimensional observations and long time horizons by working from real-valued features of observation sequences; and finally, our close-the-loop experiments provide an end-to-end practical test.**

I. Introduction

Planning a sequence of actions or a policy to maximize reward has long been considered a fundamental problem for autonomous agents. In the hardest version of the problem, an agent must form a plan based solely on its own experience, without the aid of a human engineer who can design problem-specific features or heuristics; it is this version of the problem which we must solve to build a truly autonomous agent.

Partially Observable Markov Decision Processes (POMDPs) [23, 3] are a general framework for single-agent planning. POMDPs model the state of the world as a latent variable and explicitly reason about uncertainty in both action effects and state observability. Plans in POMDPs are expressed as *policies*, which specify the action to take given any possible probability distribution over states. Unfortunately, exact planning algorithms such as *value iteration* [23] are computationally intractable for most realistic POMDP planning problems. Furthermore, researchers have had only limited success learning POMDP models from data.

Predictive State Representations (PSRs) [11] and the closely related *Observable Operator Models (OOMs)* [8] are generalizations of POMDPs that have attracted interest because they both have greater representational capacity than POMDPs and yield representations that are *at least* as compact [20, 4]. In contrast to the latent-variable representations of POMDPs, PSRs and OOMs represent the state of a dynamical system by tracking occurrence probabilities of a set of future events (called *tests* or *characteristic events*) conditioned on past events (called *histories* or *indicative events*). Because tests and histories are observable quantities, it has been suggested that learning PSRs and OOMs should be easier than learning POMDPs. And, many successful approximate planning techniques for POMDPs can be used to plan in PSRs and OOMs with minimal adjustment.

The quality of an optimized policy for a POMDP, PSR, or OOM depends strongly on the accuracy of the model: inaccurate models typically lead to useless plans. We can specify a model manually or learn one from data, but due to the difficulty of learning, it is far more common to see planning algorithms applied to hand-specified models, and therefore to small systems where there is extensive and goal-relevant domain knowledge. For example, recent extensions of approximate planning techniques for PSRs have only been applied to hand-constructed models [9, 7].

Learning models for planning in partially observable environments has met with only limited success. As a result, there have been few successful attempts at closing the loop by learning a model from an environment, planning in that model, and testing the plan in the environment. For example, Expectation-Maximization (EM) [1] does not avoid local minima or scale to large state spaces; and, although many learning algorithms have been proposed for PSRs [21, 30, 13, 26, 2] and OOMs [8, 5, 31], none have been shown to learn models that are accurate enough for planning.

Several researchers have, however, made progress in the problem of planning using a learned model. In one instance [17], researchers obtained a POMDP heuristically from the output of a model-free algorithm [12] and demonstrated

planning on a small toy maze. In another instance [16], researchers used Markov Chain Monte Carlo (MCMC) inference both to learn a factored Dynamic Bayesian Network (DBN) representation of a POMDP in a small synthetic network administration domain, as well as to perform online planning. Due to the cost of the MCMC sampler used, this approach is still impractical for larger models. In a final example, researchers learned Linear-Linear Exponential Family PSRs from an agent traversing a simulated environment, and found a policy using a policy gradient technique with a parameterized function of the learned PSR state as input [29, 27]. In this case both the learning and the planning algorithm were subject to local optima. In addition, the authors determined that the learned model was too inaccurate to support value-function-based planning methods [27]. Perhaps the closest prior work is that of Rosencrantz et al. [15]: like our method, their algorithm uses a straightforward sequence of algebraic operations to derive parameter estimates from matrices of observable statistics. However, Rosencrantz et al. do not attempt to provide a detailed proof of consistency such as the one in Equations 4(a–c) and Section III below. And, their method does not easily generalize to allow real-valued observations, "indicative features," and "characteristic features," as we derive in Sections III-A and III-B below; in our experience, using these features (instead of discrete, mutually-exclusive events) greatly reduces the variance of our estimated model parameters. Finally, the experiments of Rosencrantz et al. focus on the observation-only case, rather than on estimating the effects of actions and using the learned model for planning. (We describe several more-minor differences between the algorithms below in Section III.)

The current paper differs from these and other previous examples of planning in learned models: it both uses a principled and provably statistically consistent model-learning algorithm, and demonstrates positive results on a challenging high-dimensional problem with continuous observations. In particular, we propose a novel, consistent spectral algorithm for learning a variant of PSRs called *Transformed PSRs* [15] directly from execution traces. The algorithm is closely related to subspace identification for learning Linear Dynamical Systems (LDSs) [22, 25] and spectral algorithms for learning Hidden Markov Models (HMMs) [6] and Reduced-Rank HMMs [18]. We then demonstrate that this algorithm is able to learn compact models of a difficult, realistic dynamical system without any prior domain knowledge built into the model or algorithm. Finally, we perform approximate point-based value iteration (PBVI) in the learned compact models, and demonstrate that the greedy policy for the resulting value function works well in the original (not the learned) system. To our knowledge this is the first research that combines all of these achievements, closing the loop from observations to actions in an unknown nonlinear, non-Gaussian system with no human intervention beyond collecting the raw transition data and specifying features.

II. Predictive State Representations

A predictive state representation (PSR) [11] is a compact and complete description of a dynamical system. PSRs represent state as a set of predictions of observable experiments or *tests* that one could perform in the system. Specifically, a test of length k is an ordered sequence of action-observation pairs $\tau = a_1^\tau o_1^\tau \dots a_k^\tau o_k^\tau$ that can be executed and observed at a given time. Likewise, a *history* is an ordered sequence of action-observation pairs $h = a_1^h o_1^h \dots a_t^h o_t^h$ that has been executed and observed prior to a given time. The *prediction* for a test τ is the probability of the sequence of observations $\tau^O = o_1^\tau, \dots, o_k^\tau$ being generated, given that we intervene to take the sequence of actions $\tau^A = a_1^\tau, \dots, a_k^\tau$. If the observations produced by the dynamical system match those specified by the test, the test is said to have *succeeded*. The key idea behind a PSR is that, if we know the expected outcomes of executing all possible tests, then we also know everything there is to know about the state of a dynamical system.

In PSRs, actions in tests are *interventions*, not observations. We use a single vertical bar to indicate conditioning and a double vertical bar to indicate intervening: e.g., $\Pr[\tau^O \,|\, h \,||\, \tau^A]$ is the probability of the observations in test τ, given an observed history h, and given that we intervene to execute the actions in τ. We write $Q(h)$ for the *prediction vector* of success probabilities for a set of tests $Q = \{q_i\}$:

$$Q(h) = [\Pr[q_1^O \,|\, h \,||\, q_1^A], ..., \Pr[q_{|Q|}^O \,|\, h \,||\, q_{|Q|}^A]]^\mathsf{T}$$

Knowing the probabilities of some tests may allow us to compute the probabilities of other tests. That is, given a test τ and a set of tests Q, there may exist a *prediction function* f_τ such that $\Pr[\tau^O \,|\, h \,||\, \tau^A] = f_\tau(Q(h))$ for all histories h. In this case, we say $Q(h)$ is a *sufficient statistic* for τ.

Formally, a PSR is a tuple $\langle A, O, Q, F, m_1 \rangle$. A is the set of possible actions, and O is the set of possible observations. Q is a *core* set of tests, i.e., a set whose prediction vector $Q(h)$ is a sufficient statistic for *all* tests. F is the set of prediction functions f_τ for all tests τ (which must exist since Q is a core set), and $m_1 = Q(\epsilon)$ is the initial prediction vector after seeing the empty history ϵ. In this work we will restrict ourselves to *linear* PSRs, in which all prediction functions are linear: $f_\tau(Q(h)) = r_\tau^\mathsf{T} Q(h)$ for some vector $r_\tau \in \mathbb{R}^{|Q|}$.

Since $Q(h)$ is a sufficient statistic for all tests, it is a *state* for our PSR: i.e., we can remember just $Q(h)$ instead of h itself. After taking action a and seeing observation o, we can update $Q(h)$ recursively: if we write M_{ao} for the matrix with rows $r_{ao\tau}^\mathsf{T}$ for $\tau \in Q$, then we can use Bayes' Rule to show:

$$Q(hao) = \frac{M_{ao} Q(h)}{\Pr[o \,|\, h \,||\, a]} = \frac{M_{ao} Q(h)}{m_\infty^\mathsf{T} M_{ao} Q(h)} \tag{1}$$

where m_∞ is a normalizer, defined by $m_\infty^\mathsf{T} Q(h) = 1 \, (\forall h)$.

Specifying a PSR involves first finding a core set of tests Q, called the *discovery problem*, and then finding the parameters M_{ao}, m_∞, and m_1 for these tests, called the *learning problem*. A core set Q for a linear PSR is said to be *minimal* if the tests in Q are linearly independent [8, 20], i.e., no one test's

prediction is a linear function of the other tests' predictions. The discovery problem is usually solved by searching for linearly independent tests by repeatedly performing Singular Value Decompositions (SVDs) on collections of tests [30]. The learning problem is then solved by regression.

Transformed PSRs (TPSRs) [15] are a generalization of PSRs: TPSRs maintain a small number of sufficient statistics which are *linear combinations* of a (potentially very large and non-minimal) set of test probabilities. Accordingly, TPSRs can be thought of as *linear transformations* of regular PSRs. Therefore, TPSRs include PSRs as a special case, since this transformation can be the identity. The main benefit of TPSRs is that, given a core set of tests, the parameter learning problem can be solved and a large step toward solving the discovery problem can be achieved in closed form, as we will see below. In this respect, TPSRs are closely related to the transformed representations of LDSs and HMMs found by *subspace identification* [25, 22, 6].

A. Observable Representations

Our learning algorithm is based on an *observable representation* of a PSR, that is, one where each parameter corresponds directly to observable quantities. This representation depends on a core set of tests $\mathcal{T}$ (not necessarily minimal). It also depends on a set $\mathcal{H}$ of *indicative events*, that is, a mutually exclusive and exhaustive partition of the set of all possible histories. We will assume $\mathcal{H}$ is *sufficient* (defined below).

For purposes of gathering data, we assume that we can sample from some distribution ω over histories; our observable representation depends on ω as well. E.g., ω might be the steady-state distribution of some exploration policy. Note that this assumption means that we *cannot* estimate m_1, since we don't have samples of trajectories starting from m_1. So, instead, we will estimate m_*, an arbitrary feasible state, which is enough information to enable prediction. If we make the stronger assumption that we can repeatedly reset our PSR to its starting distribution, a straightforward modification of our algorithm will allow us to estimate m_1 as well.

We define several observable matrices in terms of $\mathcal{T}$, $\mathcal{H}$, and ω. After each definition we show how these matrices relate to the parameters of the underlying PSR. These relationships will allow us to define an equivalent TPSR, and will also be key tools in designing our learning algorithm and showing its consistency. The first observable matrix is $P_{\mathcal{H}} \in \mathbb{R}^{|\mathcal{H}|}$, containing the probabilities of every event $H_i \in \mathcal{H}$ when we sample a history h according to ω:

$$[P_{\mathcal{H}}]_i \equiv \Pr[H_i] \implies P_{\mathcal{H}} = \Pr[\mathcal{H}] \tag{2a}$$

Here we have defined $\Pr[H_i]$ to mean $\Pr[h \in H_i]$, and $\Pr[\mathcal{H}]$ to mean the vector whose elements are $\Pr[H_i]$ for $H_i \in \mathcal{H}$.

The next observable matrix is $P_{\mathcal{T},\mathcal{H}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{H}|}$, whose entries are *joint* probabilities of tests $\tau_i \in \mathcal{T}$ and indicative events $H_j \in \mathcal{H}$ when we sample $h \sim \omega$ and take actions τ_i^A:

$$\begin{aligned}[P_{\mathcal{T},\mathcal{H}}]_{i,j} &\equiv \Pr[\tau_i^O, H_j \,||\, \tau_i^A] \equiv \mathbb{E}[r_{\tau_i}^\mathsf{T} Q(h) \mid H_j] \Pr[H_j] \\ &\equiv r_{\tau_i}^\mathsf{T} s_{H_j} \Pr[H_j] \\ \implies P_{\mathcal{T},\mathcal{H}} &= RSD \end{aligned} \tag{2b}$$

Here we define $s_{H_j} = \mathbb{E}[Q(h) \mid H_j]$ to be the expected state given indicative event H_j; and as above, the vector r_{τ_i} lets us compute the probability of test τ_i given the state. Finally, we let $R \in \mathbb{R}^{|\mathcal{T}|\times|Q|}$ be the matrix with rows $r_{\tau_i}^\mathsf{T}$, $S \in \mathbb{R}^{|Q|\times|\mathcal{H}|}$ be the matrix with columns s_{H_j}, and $D = \operatorname{diag}(\Pr[\mathcal{H}])$.

Eq. (2b) tells us that the rank of $P_{\mathcal{T},\mathcal{H}}$ is no more than $|Q|$, since its factors R and S each have rank at most $|Q|$. At this point we can define a *sufficient* set of indicative events as promised: it is a set of indicative events for which the rank of $P_{\mathcal{T},\mathcal{H}}$ is *equal* to $|Q|$.

The final observable matrices are $P_{\mathcal{T},ao,\mathcal{H}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{H}|}$, one matrix for each action-observation pair. Entries of $P_{\mathcal{T},ao,\mathcal{H}}$ are probabilities of *triples* of an indicative event, the next action-observation pair, and a subsequent test, if we intervene to execute a and τ_i^A:

$$\begin{aligned}[P_{\mathcal{T},ao,\mathcal{H}}]_{i,j} &\equiv \Pr[\tau_i^O, o, H_j \,||\, a, \tau_i^A] \\ &= \mathbb{E}[\Pr[\tau_i^O, o \,|\, h \,||\, a, \tau_i^A] \mid H_j] \Pr[H_j] \\ &= \mathbb{E}[\Pr[\tau_i^O \mid h, o \,||\, a, \tau_i^A] \Pr[o \,|\, h \,||\, a] \mid H_j] \Pr[H_j] \\ &= \mathbb{E}[r_{\tau_i}^\mathsf{T} Q(hao) \Pr[o \,|\, h \,||\, a] \mid H_j] \Pr[H_j] \\ &= \mathbb{E}[r_{\tau_i}^\mathsf{T} M_{ao} Q(h) \mid H_j] \Pr[H_j] \quad \text{by Eq. (1)} \\ &= r_{\tau_i}^\mathsf{T} M_{ao} s_{H_j} \Pr[H_j] \\ \implies P_{\mathcal{T},ao,\mathcal{H}} &= R M_{ao} S D \end{aligned} \tag{2c}$$

Just like $P_{\mathcal{T},\mathcal{H}}$, the matrices $P_{\mathcal{T},ao,\mathcal{H}}$ have rank at most $|Q|$ due to their factors R and S.

We can also relate the factor S to the parameters m_∞ and m_*: for m_∞, we start from the fact $m_\infty^\mathsf{T} S = 1^\mathsf{T}$ (since each column of S is a vector of core-test predictions), repeatedly multiply by S and $S^\dagger$, and use the fact $SS^\dagger = I$.

$$m_\infty^\mathsf{T} S = 1_k^\mathsf{T} \implies m_\infty^\mathsf{T} S S^\dagger = 1_k^\mathsf{T} S^\dagger \implies m_\infty^\mathsf{T} = 1_k^\mathsf{T} S^\dagger \tag{3a}$$

$$\implies m_\infty^\mathsf{T} S = 1_k^\mathsf{T} S^\dagger S \implies 1_k^\mathsf{T} = 1_k^\mathsf{T} S^\dagger S \tag{3b}$$

Here, $k = |\mathcal{H}|$ and 1_k denotes the ones-vector of length k. For m_*, we note that each column of S is a feasible state, and so we can take m_* to be any convex combination of the columns of S. In particular, we will take $m_* = S \Pr[\mathcal{H}]$.

We now define a TPSR in terms of the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, $P_{\mathcal{T},ao,\mathcal{H}}$ and an additional matrix $U \in \mathbb{R}^{|\mathcal{T}|\times|Q|}$ such that $U^\mathsf{T} R$ is invertible. A natural choice for U is the left singular vectors of $P_{\mathcal{T},\mathcal{H}}$, although a randomly-generated U will work with probability 1. We can think of the columns of U as specifying a core set Q' of *linear combinations* of tests which define a state vector for our TPSR. Finally we simplify the expressions using Equations 2(a–c) to show that our parameters are only a similarity transform away from the original PSR parameters:

$$b_* \equiv U^\mathsf{T} P_{\mathcal{T},\mathcal{H}} 1_k = U^\mathsf{T} R S D 1_k = (U^\mathsf{T} R) m_* \tag{4a}$$

$$\begin{aligned} b_\infty^\mathsf{T} &\equiv P_{\mathcal{H}}^\mathsf{T} (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger = 1_n^\mathsf{T} S^\dagger S D (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= 1_n^\mathsf{T} S^\dagger (U^\mathsf{T} R)^{-1} (U^\mathsf{T} R) S D (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger \\ &= 1_n^\mathsf{T} S^\dagger (U^\mathsf{T} R)^{-1} = m_\infty^\mathsf{T} (U^\mathsf{T} R)^{-1} \end{aligned} \tag{4b}$$

$$
\begin{aligned}
B_{ao} &\equiv U^\mathsf{T} P_{\mathcal{T},ao,\mathcal{H}} (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= U^\mathsf{T} R M_{ao} S D (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= U^\mathsf{T} R M_{ao} (U^\mathsf{T} R)^{-1} (U^\mathsf{T} R) S D (U^\mathsf{T} P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= (U^\mathsf{T} R) M_{ao} (U^\mathsf{T} R)^{-1} \qquad (4c)
\end{aligned}
$$

The derivation of Eq. 4b makes use of Eqs. 3a and 3b. To get $b_1 = (U^\mathsf{T} R) m_1$ instead of b_* in Eq. 4a, replace $P_{\mathcal{T},\mathcal{H}} 1_k$, the vector of expected probabilities of tests for $h \sim \omega$, with the vector of probabilities of tests for $h = \epsilon$.

Given these parameters, we can calculate the probability of observations $o_{1:t}$, given that we start from state m_1 and intervene with actions $a_{1:t}$. Here we write the product of a sequence of transitions as $M_{ao_{1:t}} = M_{a_1 o_1} M_{a_2 o_2} \dots M_{a_t o_t}$.

$$
\begin{aligned}
\Pr[o_{1:t} || a_{1:t}] &= m_\infty^\mathsf{T} (U^\mathsf{T} R)^{-1} (U^\mathsf{T} R) M_{ao_{1:t}} (U^\mathsf{T} R)^{-1} (U^\mathsf{T} R) m_1 \\
&= b_\infty^\mathsf{T} B_{ao_{1:t}} b_1 \qquad (5)
\end{aligned}
$$

In addition to the initial TPSR state b_1, we define normalized conditional *internal states* b_t by a recursive update:

$$
b_{t+1} = \frac{B_{ao_t} b_t}{b_\infty^\mathsf{T} B_{ao_t} b_t} \qquad (6)
$$

It is straightforward to show recursively that b_t can be used to compute conditional observation probabilities. (If we only have b_* instead of b_1, then initial probability estimates will be approximate, but the difference in predictions will disappear over time as our process mixes.) The linear transform from a TPSR internal state b_t to a PSR internal state $q_t = Q(h_t)$ is given by $q_t = (U^\mathsf{T} R)^{-1} b_t$. If we chose U by SVD, then the prediction of tests $\Pr[\mathcal{T}^O \mid h \,||\, \mathcal{T}^A]$ at time t is given by $U b_t = U U^\mathsf{T} R q_t = R q_t$ (since by the definition of SVD, U is orthonormal and the row spaces of U and R are the same).

III. Learning TPSRs

Our learning algorithm works by building empirical estimates $\widehat{P}_\mathcal{H}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ of the matrices $P_\mathcal{H}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$ defined above. To build these estimates, we repeatedly sample a history h from the distribution ω, execute a sequence of actions, and record the resulting observations.

Once we have computed $\widehat{P}_\mathcal{H}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$, we can generate $\widehat{U}$ by singular value decomposition of $\widehat{P}_{\mathcal{T},\mathcal{H}}$. We can then learn the TPSR parameters by plugging $\widehat{U}$, $\widehat{P}_\mathcal{H}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ into Equation 4[1]:

$$
\begin{aligned}
\widehat{b}_* &= \widehat{U}^\mathsf{T} \widehat{P}_\mathcal{H} 1_k \\
\widehat{b}_\infty &= (\widehat{P}_{\mathcal{T},\mathcal{H}}^\mathsf{T} \widehat{U})^\dagger \widehat{P}_\mathcal{H} \\
\widehat{B}_{ao} &= \widehat{U}^\mathsf{T} \widehat{P}_{\mathcal{T},ao,\mathcal{H}} (\widehat{U}^\mathsf{T} \widehat{P}_{\mathcal{T},\mathcal{H}})^\dagger
\end{aligned}
$$

As we include more data in our averages, the law of large numbers guarantees that our estimates $\widehat{P}_\mathcal{H}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ converge to the true matrices $P_\mathcal{H}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$ (defined in Equation 2). So by continuity of the formulas above, if our system is truly a PSR of finite rank, our estimates $\widehat{b}_*$, $\widehat{b}_\infty$, and $\widehat{B}_{ao}$ converge to the true parameters up to a linear transform—that is, our learning algorithm is *consistent*. Although parameters estimated with finite data can sometimes lead to negative probability estimates when filtering or predicting, this problem can be avoided in practice by thresholding the predicted probabilities by some small positive probability.

Note that the learning algorithm presented here is distinct from the TPSR learning algorithm presented in Rosencrantz et al. [15]. In addition to the differences mentioned above, a key difference between the two algorithms is that here we estimate the *joint* probability of a past event, a current observation, and a future event in the matrix $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$, whereas in [15], the authors instead estimate the probability of a future event, *conditioned* on a past event and a current observation. To compensate, Rosencrantz et al. later multiply this estimate by an approximation of the probability of the current observation, conditioned on the past event. Rosencrantz et al. also derive the approximate probability of the current observation differently: as the result of a regression instead of directly from empirical counts. Finally, Rosencrantz et al. do not make any attempt to multiply by the marginal probability of the past event, although this term cancels in the current work. In the absence of estimation errors, both algorithms would arrive at the same answer, but taking errors into account, they will typically make different predictions. The difficulty of extending the Rosencrantz et al. algorithm to handle real-valued features stems from the difference between joint and conditional probabilities: the observable matrices in Rosencrantz et al. are conditional expectations, so their algorithm depends on being able to condition on discrete indicative events or observations. In contrast, the next section shows how to extend our algorithm to use real-valued features.

A. Learning TPSRs with Features

In data gathered from complex real-world dynamical systems, it may not be possible to find a reasonably-sized core set of discrete tests $\mathcal{T}$ or sufficient set of indicative events $\mathcal{H}$. When this is the case, we can generalize the TPSR learning algorithm and work with *features* of tests and histories, which we call *characteristic features* and *indicative features* respectively. In particular let $\mathcal{T}$ and $\mathcal{H}$ be large sets of tests and indicative events (possibly too large to work with directly) and let $\phi^\mathcal{T}$ and $\phi^\mathcal{H}$ be shorter vectors of characteristic and indicative features. The matrices $P_\mathcal{H}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$ will no longer contain probabilities but rather *expected values* of features or products of features. For the special case of features that are *indicator functions* of tests and histories, we recover the probability matrices from Section II-A.

Here we prove the *consistency* of our estimation algorithm using these more general matrices as inputs. In the following equations $\Phi^\mathcal{T}$ and $\Phi^\mathcal{H}$ are matrices of characteristic and indicative features respectively, with first dimension equal to the number of characteristic or indicative features and second dimension equal to $|\mathcal{T}|$ and $|\mathcal{H}|$ respectively. An entry of $\Phi^\mathcal{H}$ is the expectation of one of the indicative features given the occurrence of one of the indicative events. An entry of $\Phi^\mathcal{T}$

[1]The learning strategy employed here may be seen as a generalization of Hsu et al.'s spectral algorithm for learning HMMs [6] to PSRs. Note that since HMMs and POMDPs are a proper subset of PSRs, we can use the algorithm in this paper to learn back both HMMs and POMDPs in PSR form.

is the weight of one of our tests in calculating one of our characteristic features. With these features we generalize the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$:

$$
\begin{aligned}
&[P_{\mathcal{H}}]_i \equiv \mathbb{E}(\phi_i^{\mathcal{H}}(h)) = \textstyle\sum_{H\in\mathcal{H}} \Pr[H]\Phi^{\mathcal{H}}_{iH} \\
&\implies P_{\mathcal{H}} = \Phi^{\mathcal{H}} \Pr[\mathcal{H}] \qquad (7a)\\
&[P_{\mathcal{T},\mathcal{H}}]_{i,j} \equiv \mathbb{E}(\phi_i^{\mathcal{T}}(\tau^O)\cdot\phi_j^{\mathcal{H}}(h) \,||\, \tau^A) \\
&= \textstyle\sum_{\tau\in\mathcal{T}}\sum_{H\in\mathcal{H}} \Pr[\tau^O, H \,||\, \tau^A]\Phi^{\mathcal{T}}_{i\tau}\Phi^{\mathcal{H}}_{jH} \\
&= \textstyle\sum_{\tau\in\mathcal{T}} r_\tau^{\mathsf{T}}\Phi^{\mathcal{T}}_{i\tau}\sum_{H\in\mathcal{H}} s_H \Pr[H]\Phi^{\mathcal{H}}_{jH} \qquad \text{by Eq. (2b)} \\
&\implies P_{\mathcal{T},\mathcal{H}} = \Phi^{\mathcal{T}} R S D \Phi^{\mathcal{H}\mathsf{T}} \qquad (7b)\\
&[P_{\mathcal{T},ao,\mathcal{H}}]_{i,j} \equiv \mathbb{E}(\phi_i^{\mathcal{T}}(\tau^O)\cdot\phi_j^{\mathcal{H}}(h)\cdot\delta(o) \,||\, a, \tau^A) \\
&= \textstyle\sum_{\tau\in\mathcal{T}}\sum_{H\in\mathcal{H}} \Pr[\tau^O, o, H \,||\, a, \tau^A]\Phi^{\mathcal{T}}_{i\tau}\Phi^{\mathcal{H}}_{jH} \\
&= \left(\textstyle\sum_{\tau\in\mathcal{T}} r_\tau^{\mathsf{T}}\Phi^{\mathcal{T}}_{i\tau}\right) M_{ao} \left(\textstyle\sum_{H\in\mathcal{H}} s_H \Pr[H]\Phi^{\mathcal{H}}_{jH}\right) \overset{\text{by Eq. (2c)}}{} \\
&\implies P_{\mathcal{T},ao,\mathcal{H}} = \Phi^{\mathcal{T}} R M_{ao} S D \Phi^{\mathcal{H}\mathsf{T}} \qquad (7c)
\end{aligned}
$$

where $\delta(o)$ is an indicator for observation o. The parameters of the TPSR (b_*, b_∞, and B_{ao}) are now defined in terms of a matrix U such that $U^{\mathsf{T}}\Phi^{\mathcal{T}}R$ is invertible (we can take U to be the left singular values of $P_{\mathcal{T},\mathcal{H}}$), and in terms of the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$. We also define a new vector e s.t. $\Phi^{\mathcal{H}\mathsf{T}}e = 1_k$; this means that the ones vector 1_k^{T} must be in the row space of $\Phi^{\mathcal{H}}$. Since $\Phi^{\mathcal{H}}$ is a matrix of features, we can always ensure that this is the case by requiring one of our features to be a constant. Finally we simplify the expressions using Equations 7(a–c) to show that our parameters are only a similarity transform away from the original PSR parameters:

$$
\begin{aligned}
b_* &\equiv U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}}e = U^{\mathsf{T}}\Phi^{\mathcal{T}}RSD\Phi^{\mathcal{H}\mathsf{T}}e \\
&= U^{\mathsf{T}}\Phi^{\mathcal{T}}RSD1_k = (U^{\mathsf{T}}\Phi^{\mathcal{T}}R)m_* \qquad (8a)\\
b_\infty^{\mathsf{T}} &\equiv P_{\mathcal{H}}^{\mathsf{T}}(U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}})^\dagger = 1_n^{\mathsf{T}}S^\dagger SD\Phi^{\mathcal{H}\mathsf{T}}(U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= 1_n^{\mathsf{T}}S^\dagger(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)^{-1}(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)SD\Phi^{\mathcal{H}\mathsf{T}}(U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= 1_n^{\mathsf{T}}S^\dagger(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)^{-1} = m_\infty^{\mathsf{T}}(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)^{-1} \qquad (8b)\\
B_{ao} &\equiv U^{\mathsf{T}}P_{\mathcal{T},ao,\mathcal{H}}(U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= U^{\mathsf{T}}\Phi^{\mathcal{T}}RM_{ao}(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)^{-1}(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)SD\Phi^{\mathcal{H}\mathsf{T}}(U^{\mathsf{T}}P_{\mathcal{T},\mathcal{H}})^\dagger \\
&= (U^{\mathsf{T}}\Phi^{\mathcal{T}}R)M_{ao}(U^{\mathsf{T}}\Phi^{\mathcal{T}}R)^{-1} \qquad (8c)
\end{aligned}
$$

Just as in the beginning of Section III, we can estimate $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$, and then plug the matrices into Equations 8(a–c). Thus we see that if we work with characteristic and indicative features, and if our system is truly a TPSR of finite rank, our estimates $\widehat{b}_*$, $\widehat{b}_\infty$, and $\widehat{B}_{ao}$ again converge to the true PSR parameters up to a linear transform.

B. Kernel Density Estimation for Continuous Observations

For continuous observations, we use Kernel Density Estimation (KDE) [19] to model the observation probability density function (PDF). Although use of kernel density estimation for continuous observations in PSRs is not new [28], extending our algorithm to use KDE results, for the first time, in a statistically consistent learning algorithm for PSRs with continuous observations. We use a fraction of the training data points as kernel centers, placing one multivariate Gaussian kernel at each point.[2] The KDE of the observation PDF is a convex combination of these kernels; since each kernel integrates to 1, this estimator also integrates to 1. KDE theory [19] tells us that, with the correct kernel weights, as the number of kernel centers and the number of samples go to infinity and the kernel bandwidth goes to zero (at appropriate rates), the KDE estimator converges to the observation PDF in L_1 norm. The kernel density estimator is completely determined by the normalized vector of kernel weights; therefore, if we can estimate this vector accurately, our estimate of the observation PDF will converge to the observation PDF as well. Hence our goal is to predict the correct expected value of this normalized kernel vector given all past observations. In the continuous-observation case, we can still write our latent-state update in the same form, using a matrix B_{ao}; however, rather than learning each of the uncountably-many B_{ao} matrices separately, we learn one base operator per kernel center, and use convex combinations of these base operators to compute observable operators as needed. For details on practical aspects of learning with continuous observations, see Section IV-B.

C. Practical Computation of the TPSR Parameters

As defined above, each element of $\widehat{P}_{\mathcal{H}}$ is the empirical expectation (over histories sampled from ω) of the corresponding element of the indicative feature vector—that is, element i is $\frac{1}{w}\sum_{t=1}^{w}\phi_{it}^{\mathcal{H}}$, where $\phi_{it}^{\mathcal{H}}$ is the i^{th} indicative feature evaluated at the t^{th} sampled history. Similarly, each element of $\widehat{P}_{\mathcal{T},\mathcal{H}}$ is an empirical expectation of the *product* of one indicative feature and one characteristic feature, if we sample a history from ω and then follow an appropriate sequence of actions. We can compute all elements of $\widehat{P}_{\mathcal{T},\mathcal{H}}$ from a single sample of trajectories if we sample histories from ω, follow an appropriate exploratory policy, and then importance-weight each sample [2]: $[\widehat{P}_{\mathcal{T},\mathcal{H}}]_{ij}$ is $\sum_{t=1}^{w}\alpha_t\phi_{it}^{\mathcal{T}}\phi_{jt}^{\mathcal{H}}$, where α_t is an importance weight. (In our experiments below, the importance weights are constant by design, and therefore cancel out.)

Once we have constructed $\widehat{P}_{\mathcal{T},\mathcal{H}}$, we can compute $\widehat{U}$ as the matrix of left singular vectors of $\widehat{P}_{\mathcal{T},\mathcal{H}}$. One of the advantages of subspace identification is that the complexity of the model can be tuned by selecting the number of singular vectors in $\widehat{U}$, at the risk of losing prediction quality.

Finally, since there may be many large matrices $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$, rather than computing them directly, we instead compute $\widehat{U}^{\mathsf{T}}\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ for each pair a, o. The latter matrices are much smaller, and in our experiments, we saved substantially on both memory and runtime by avoiding construction of the larger matrices. To construct $\widehat{U}^{\mathsf{T}}\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$, we restrict to those training trajectories in which the action at the middle time step is a. Then, each element of $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ is a weighted empirical expectation (among the restricted set of trajectories) of the product of one indicative feature, one characteristic feature, and element o of the observation kernel vector. So,

[2]We use a general elliptical covariance matrix, chosen by PCA: that is, we use a spherical covariance after projecting onto the eigenvectors of the covariance matrix of the observations, and scaling by the square roots of the eigenvalues.

$$\widehat{U}^\intercal \widehat{P}_{\mathcal{T},ao,\mathcal{H}} = \sum_{t=1}^{w_a} \alpha_t^a (\widehat{U}^\intercal \phi_t^{\mathcal{T}})(\phi_t^{\mathcal{H}})^\intercal \frac{1}{Z_t} K(o_t - o) \quad (9)$$

where $K(\cdot)$ is the kernel function and Z_t is the kernel normalization constant computed by summing over the observation kernels for each o_t. As above, in our experiments, the importance weights α_t^a are uniform.

IV. Experimental Results

We have introduced a novel algorithm for learning TPSRs directly from data, as well as a kernel-based extension for modeling continuous observations. We judge the quality of our TPSR learning algorithm by first learning a model of a challenging non-linear, partially observable, controlled domain directly from sensor inputs and then "closing the loop" by *planning* in the learned model. Successful planning is a much stronger result than standard dynamical system evaluations such as one-step squared error or prediction log-likelihood. Unlike previous attempts to learn PSRs, which either lack planning results [15, 28], or which compare policies within the learned system [29], we compare our resulting policy to a bound on the best possible solution in the original system and demonstrate that the policy is close to optimal.

A. The Autonomous Robot Domain

Our simulated autonomous robot domain consisted of a simple 45×45 unit square arena with a central obstacle and brightly colored walls (Figure 1(A-B)), containing a robot of radius 2 units. The robot could move around the floor of the arena and rotate to face in any direction. The robot had a simulated 16×16 pixel color camera, with a focal plane one unit in front of the robot's center of rotation, and with a visual field of 45° in both azimuth and elevation, corresponding to an angular resolution of $\sim 2.8^\circ$ per pixel. The resulting 768-element pattern of unprocessed RGB values was the only input to the robot (images were *not* preprocessed to extract features), and each action produced a new set of pixel values. The robot was able to move forward 1 or 0 units, and simultaneously rotate 15°, -15°, or 0°, resulting in 6 unique actions. In the real world, friction, uneven surfaces, and other factors confound precisely predictable movements. To simulate this uncertainty, a small amount of Gaussian noise was added to the translation (mean 0, standard deviation .1 units) and rotation (mean 0, standard deviation 5°) components of the actions. The robot was allowed to occupy any real-valued (x, y, θ) pose that didn't intersect a wall; in case of an attempt to drive through a wall, we interrupted the commanded motion just before contact, simulating an inelastic collision.

The autonomous robot domain was designed to be a difficult domain comparable to the most complex domains that previous PSR algorithms have attempted to model. In particular, the domain in this paper was modeled after the autonomous robot domains found in recent PSR work [28, 29]. The proposed problem, learning a model of this domain and then planning in the learned model, is quite difficult. The autonomous robot has *no* knowledge of any of the underlying properties of the domain, e.g. the geometry of the environment or the robot motion model; it only has access to samples of the 256 pixel features, and how these features change as actions are executed. Writing a correct policy for a specific task in this domain by hand would be at best tedious—and in any case, as mentioned above, it is often impractical to hand-design a policy for an autonomous agent, since doing so requires guessing the particular planning problems that the agent may face in the future. Furthermore, the continuous and non-linear nature of this domain makes learning models difficult. For example, a POMDP model of this domain would require a prohibitively large number of hidden states, making learning and planning next to impossible. PSRs are able to overcome this problem by *compactly* representing state in a low-dimensional real-valued space, and the algorithm presented in this work allows us to efficiently learn the parameters of the PSR in closed form.

B. Learning a Model

We learn our model from a sample of 10000 short trajectories, each containing 7 action-observation pairs. We generate each trajectory by starting from a uniformly randomly sampled position in the environment and executing a uniform random sequence of actions. We used the first $l = 2000$ trajectories to generate kernel centers, and the remaining $w = 8000$ to estimate the matrices $P_{\mathcal{H}}$, $P_{\mathcal{T},\mathcal{H}}$, and $P_{\mathcal{T},ao,\mathcal{H}}$.

To define these matrices, we need to specify a set of indicative features, a set of observation kernel centers, and a set of characteristic features. We use Gaussian kernels to define our indicative and characteristic features, in a similar manner to the Gaussian kernels described above for observations; our analysis allows us to use arbitrary indicative and characteristic features, but we found Gaussian kernels to be convenient and effective. Note that the resulting features over tests and histories are just *features*; unlike the kernel centers defined over observations, there is no need to let the kernel width approach zero, since we are not attempting to learn accurate PDFs over the histories and tests in $\mathcal{H}$ and $\mathcal{T}$.

In more detail, we define a set of 2000 *indicative kernels*, each one centered at a sequence of 3 observations from the initial segment of one of our trajectories. We choose the kernel covariance using PCA on these sequences of observations, just as described for single observations in Section III-B. We then generate our indicative features for a new sequence of three observations by evaluating each indicative kernel at the new sequence, and normalizing so that the vector of features sums to one. (The initial distribution ω is, therefore, the distribution obtained by initializing uniformly and taking 3 random actions.) Similarly, we define 2000 *characteristic kernels*, each one centered at a sequence of 3 observations from the end of one of our sample trajectories, choose a kernel covariance, and define our characteristic feature vector by evaluating each kernel at a new observation sequence and normalizing. Finally, we define 500 *observation kernels*, each one centered at a single observation from the middle of one of our sample trajectories, and replace each observation by its corresponding vector of normalized kernel weights. Next, we construct the matrices $\widehat{P}_{\mathcal{H}}$, $\widehat{P}_{\mathcal{T},\mathcal{H}}$, and $\widehat{P}_{\mathcal{T},ao,\mathcal{H}}$ as the empirical

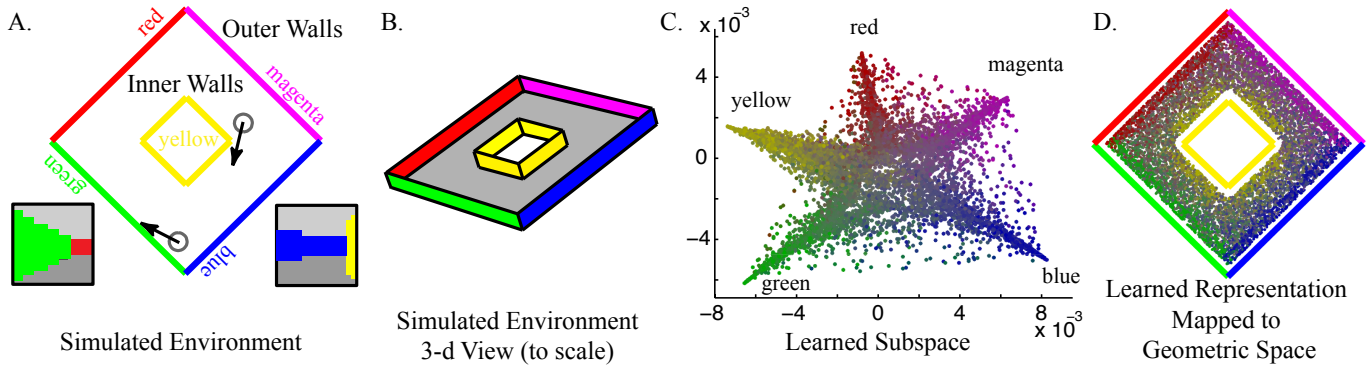

Fig. 1. Learning the Autonomous Robot Domain. Full-color figures are available in the online proceedings: http://www.roboticsproceedings.org/rss06/p36.html (A) The robot uses visual sensing to traverse a square domain with multi-colored walls and a central obstacle. Examples of images recorded by the robot occupying two different positions in the environment are shown at the bottom of the figure. (B) A to-scale 3-dimensional view of the environment. (C) The 2nd and 3rd dimension of the learned subspace (the first dimension primarily contained normalization information). Each point is the embedding of a single history, displayed with color equal to the average RGB color in the first image in the highest probability test. The star-shaped manifold captures the visual space of the robot with each "point" of the star containing concentrations of embeddings that predict images predominantly composed of a particular color. (D) The same points in (C) projected onto the environment's geometric space, demonstrating that the manifold sensibly captures features of geometric space.

expectations over our 8000 training trajectories according to the equations in Section III. Finally we chose $|Q'| = 5$ as the dimension of our TPSR, the smallest dimension that was able to produce high quality policies (see Section IV-D below).

C. Qualitative Evaluation

Having learned the parameters of the TPSR according to the algorithm in Section III, we can use the model for prediction, filtering, and planning in the autonomous robot domain. We first evaluated the model *qualitatively* by projecting the sets of histories in the training data onto the learned TPSR state space: $\widehat{U}^\intercal \widehat{P}_\mathcal{H}$. We colored each datapoint according to the average of the red, green, and blue components of the highest probability observation following the projected history. The features of the low dimensional embedding clearly capture the topology of the major features of the robot's visual environment (Figure 1(C-D)), and continuous paths in the environment translate into continuous paths in the latent space (Figure 2(B)).

D. Planning in the Learned Model

To test the quality of the learned model, we set up a navigation problem where the robot was required to plan to reach a goal image (looking directly at the blue wall). We specified a large reward (1000) for this observation, a reward of -1 for colliding with a wall, and 0 for every other observation. We learned a reward function by linear regression from the embedded histories $\widehat{U}^\intercal \widehat{P}_\mathcal{H}$ to the observed immediate rewards. We used the learned reward function to compute an approximate state-action value function via the PSR extension of the Perseus variant of PBVI [14, 24, 9, 7] with discount factor $\gamma = .8$, a prediction horizon of 10 steps, and with the 8000 embedded histories as the set of belief points. The learned value function is displayed in Figure 2(A). We then computed an initial belief by starting with b_* and tracking for 3 action-observation pairs, and executed the greedy policy for our learned value function. Examples of paths planned in the learned model are presented in Figure 2(B); the same paths are shown in geometric space in Figure 2(C). (Recall that the robot only has access to images, *never* its own position.)

The reward function encouraged the robot to navigate to a specific set of points in the environment, so the planning problem can be viewed as a shortest path problem. Even though we don't encode this intuition into our algorithm, we can use it to quantitatively evaluate the performance of the policy in the original system. First we randomly sampled 100 initial histories in the environment and asked the robot to plan paths for each based on its learned policy. We compared the number of actions taken both to a random policy and to the *optimistic path*, calculated by A* search in the robot's configuration space given the *true underlying position*. Note that this comparison is somewhat unfair: in order to achieve the same cost as the optimistic path, the robot would have to know its true underlying position, the dynamics would have to be deterministic, all rotations would have to be instantaneous, and the algorithm would need an unlimited amount of training data. Nonetheless, the results (Figure 2(D)) indicate that the performance of the TPSR policy is close to this optimistic bound. We think that this result is remarkable, especially given that previous approaches have encountered significant difficulty modeling continuous domains [10] and domains with similarly high levels of complexity [29].

V. Conclusions

We have presented a novel *consistent* subspace identification algorithm that simultaneously solves the *discovery* and *learning* problems for TPSRs. In addition, we provided two extensions to the learning algorithm that are useful in practice, while maintaining consistency: characteristic and indicative features only require one to know relevant features of tests and histories, rather than core sets of tests and sufficient sets of histories, while kernel density estimation can be used to find observable operators when observations are real-valued. We also showed how point-based approximate planning techniques can be used to solve the *planning* problem in the learned model. We demonstrated the representational capacity of our model and the effectiveness of our learning algorithm by learning a compact model from simulated autonomous robot vision data. Finally, we closed the loop by successfully planning with the learned models. To our knowledge this is the first instance

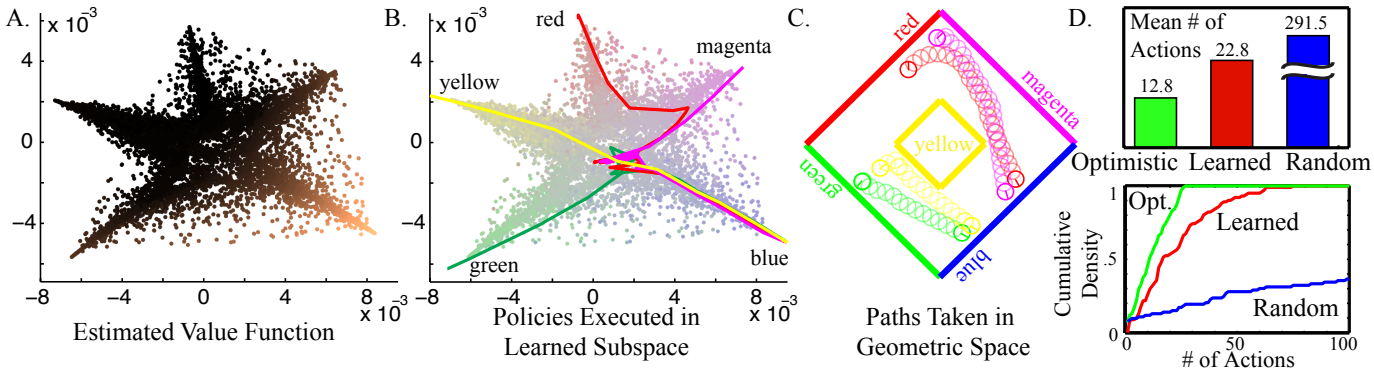

Fig. 2. Planning in the learned state space. (A) The value function computed for each embedded point; lighter indicates higher value. (B) Policies executed in the learned subspace. The red, green, magenta, and yellow paths correspond to the policy executed by a robot with starting positions facing the red, green, magenta, and yellow walls respectively. (C) The paths taken by the robot in geometric space while executing the policy. Each of the paths corresponds to the path of the same color in (B). The darker circles indicate the starting and ending positions, and the tick-mark indicates the robot's orientation. (D) Analysis of planning from 100 randomly sampled start positions to the target image (facing blue wall). In the bar graph: the mean number of actions taken by the optimistic solution found by A* search in configuration space (left); taken by executing the policy found by Perseus in the learned model (center); and taken by executing a random policy (right). Line graph illustrates the cumulative density of the number of actions given the optimal, learned, and random policies.

of learning a model for a simulated robot in a nonlinear, non-Gaussian, partially observable environment of this complexity using a consistent algorithm and successfully planning in the learned model. We compare the policy generated by our model to a bound on the best possible value, and determine that our policy is close to optimal.

We believe the spectral PSR learning algorithm presented here, and subspace identification procedures for learning PSRs in general, can increase the scope of planning under uncertainty for autonomous agents in previously intractable scenarios. We believe that this improvement is partly due to the greater representational power of PSRs as compared to POMDPs, and partly due to the efficient and statistically consistent nature of the learning method.

Acknowledgements

SMS (now at Google) was supported by the NSF under grant number 0000164, by the USAF under grant number FA8650-05-C-7264, by the USDA under grant number 4400161514, and by MobileFusion/TTC. BB was supported by the NSF under grant number EEEC-0540865. BB and GJG were supported by ONR MURI grant number N00014-09-1-1052.

References

[1] J. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, 1997.

[2] M. Bowling, P. McCracken, M. James, J. Neufeld, and D. Wilkinson. Learning predictive state representations using non-blind policies. In *Proc. ICML*, 2006.

[3] A. R. Cassandra, L. P. Kaelbling, and M. R. Littman. Acting optimally in partially observable stochastic domains. In *Proc. AAAI*, 1994.

[4] E. Even-Dar, S. M. Kakade, and Y. Mansour. Planning in POMDPs using multiplicity automata. In *UAI*, 2005.

[5] A. K. H. Jaeger, M. Zhao. Efficient training of OOMs. In *NIPS*, 2005.

[6] D. Hsu, S. Kakade, and T. Zhang. A spectral algorithm for learning hidden Markov models. In *COLT*, 2009.

[7] M. T. Izadi and D. Precup. Point-based planning for predictive state representations. In *Proc. Canadian AI*, 2008.

[8] H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12:1371–1398, 2000.

[9] M. R. James, T. Wessling, and N. A. Vlassis. Improving approximate value iteration using memories and predictive state representations. In *AAAI*, 2006.

[10] N. K. Jong and P. Stone. Towards employing psrs in a continuous domain. Technical Report UT-AI-TR-04-309, University of Texas at Austin, 2004.

[11] M. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[12] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[13] P. McCracken and M. Bowling. Online discovery and learning of predictive state representations. In *Proc. NIPS*, 2005.

[14] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. IJCAI*, 2003.

[15] M. Rosencrantz, G. J. Gordon, and S. Thrun. Learning low dimensional predictive representations. In *Proc. ICML*, 2004.

[16] S. Ross and J. Pineau. Model-based Bayesian reinforcement learning in large structured domains. In *Proc. UAI*, 2008.

[17] G. Shani, R. I. Brafman, and S. E. Shimony. Model-based online learning of POMDPs. In *Proc. ECML*, 2005.

[18] S. Siddiqi, B. Boots, and G. J. Gordon. Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2010)*, 2010.

[19] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

[20] S. Singh, M. James, and M. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proc. UAI*, 2004.

[21] S. Singh, M. L. Littman, N. K. Jong, D. Pardoe, and P. Stone. Learning predictive state representations. In *Proc. ICML*, 2003.

[22] S. Soatto and A. Chiuso. Dynamic data factorization. Technical report, UCLA, 2001.

[23] E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[24] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[25] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer, 1996.

[26] E. Wiewiora. Learning predictive representations from a history. In *Proc. ICML*, 2005.

[27] D. Wingate. *Exponential Family Predictive Representations of State*. PhD thesis, University of Michigan, 2008.

[28] D. Wingate and S. Singh. On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In *Proc. AAMAS*, 2007.

[29] D. Wingate and S. Singh. Efficiently learning linear-linear exponential family predictive representations of state. In *Proc. ICML*, 2008.

[30] B. Wolfe, M. James, and S. Singh. Learning predictive state representations in dynamical systems without reset. In *Proc. ICML*, 2005.

[31] M. Zhao, H. Jaeger, and M. Thon. A bound on modeling error in observable operator models and an associated learning algorithm. *Neural Computation*, 2009.