Unsupervised Model-Based Prediction of User Actions

Kevin R. Dixon KRD@CS.CMU.EDU
Pradeep K. Khosla PKK@ECE.CMU.EDU
Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract

In this paper we derive algorithms that construct *simple* continuous-density hidden Markov models based on unlabeled observations of user actions. These models are then be used to predict future actions of the user. The algorithms were designed for domains where user-generated training data are costly to obtain and, consequently, the number of free parameters must be kept low. More specifically, we derive an algorithm that constructs ϵ-compact CDHMMs, that is Markov chains with an irreducible number of states, based on real-valued observations from users performing tasks. We show that the worst-case computational complexity of the algorithm is a second-order polynomial in the number and length of the tasks. We derive an optimal estimator for the expectation of future observations, based on recent observations, and give the computational complexity for the estimator. We also present a novel application: predictive robot programming. We show that the algorithms derived in this paper can recognize and predict the waypoints of robot programs in a rotation- and translation-independent fashion, and we also derive the maximum-likelihood equations for locally optimal scale-invariant prediction. We present experimental results of the performance of our algorithms for predictive robot programming.

1. Introduction

Despite the numerous advances in human-computer interaction (HCI), most automation systems still require users to convey knowledge to computers and robots through procedural-programming techniques. For the vast majority of the population this transfer of knowledge is limited by the programming expertise of the user. Indeed, expertise in programming is its own job skill. Most users have neither the experience nor the inclination to program computers and robots to perform their tasks. In industrial settings, many companies do not have the resources to automate production. For example, it is estimated that over 90% of arc welding is still performed by hand due, in part, to the complexity and duration of the programming process. The downtime required to reprogram the facilities may interrupt production and the expense necessary to obtain programming expertise may be too great. A system that simplifies the automation of tasks could increase the use of computers and robots in a variety of fields previously off limits and raise productivity. We have developed algorithms that assist in the transfer of knowledge by observing users performing tasks and leveraging this information to decrease the time required to automate tasks by predicting future observations.

The general problem of a machine learning by observing a human has long been a goal of many researchers. The Learning By Observation (LBO) paradigm is characterized by a computer assimilating observations of a user performing a task (or tasks) and then synthesizing this information so that the computer can perform the task in novel configurations. We formulate the LBO problem as one of Stochastic-Source Prediction (SSP), that is predicting future values of a random process[1] based on prior observations of its behavior. The SSP problem has received quite a deal of attention since its applications are prevalent across

[1]We use the statistics definition of "random process": an indexed sequence of random variables. This contrasts to the combinatorics/cryptography implication of "random process": a sequence where previous observations are no help in predicting future observations (this would be a "white process" in statistics jargon).

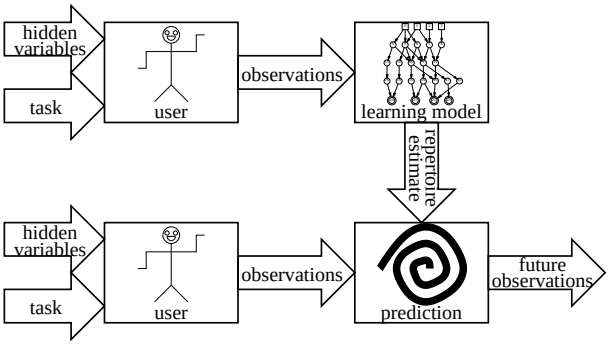

Figure 1. Conceptual diagram of the two-phase operation of our system. First, take observations to generate a user model. Second, use the model to compute future observations.

many fields. Economists use time-series analysis to recognize trends and predict the stock market. Engineers use model-based predictive control to stabilize systems and achieve optimal performance. Computer scientists use dynamic-caching algorithms to predict memory page faults in operating systems. Each of these applications has SSP at its heart. When casting the user of a system to be a stochastic source, LBO reduces to SSP. However, predicting a general random process is intractable and it is therefore necessary to apply assumptions about the behavior of the underlying stochastic source. For example, the random process can be modeled as autoregressive, moving average, or both; it can be considered stationary, ergodic, or Markov. The assumptions are dictated by the underlying physical system and desired complexity of the model, and our assumptions are driven by the modeling of (human) users through observations of their activity. Conceptually, our LBO system operates in two phases (Figure 1). The first, or learning, phase creates a model based on prior observations of the user. The second, or prediction, stage computes predictions of future user observations based on the model. There are several difficulties in predicting and synthesizing user actions. Observations obtained from sensors are, in general, noise corrupted since all sensors have uncertainty associated with them. However, the primary difficulty of predicting user actions is the inherent imprecision and poor repeatability of humans. Even with perfectly accurate sensors there will be uncertainty as to what task the user was trying to perform. It is also unfeasible to instrument fully any realistic working environment and, as a consequence, there will be latent causes, or hidden variables, for certain user actions. Any system that attempts to model user actions must address these types of uncertainty. Furthermore, observations in our system are gathered from human activity, which can be of a significant real-world duration. This results in a very low availability of data upon which to train and requires that our system keep the number of tunable parameters to a minimum. As will be discussed later, our user model is a Continuous-Density Hidden Markov Model (CDHMM) with acyclic connectivity.

In Section 2 we overview the key ideas behind modeling user actions. We develop the learning algorithm from a theoretical perspective in Section 3 and derive the optimal prediction estimator in Section 4. We outline a novel application of the SSP problem in Section 5 by describing our work in predictive robot programming, detailing both theoretical and practical aspects of the problem. We place this research in the context of related work in Section 6. Finally, conclusions and future work are given in Section 7.

2. Modeling User Actions

In this section, we formally and informally define some terms used in this paper. A *user observation* is a real-valued vector sampled at discrete time intervals. To account for inherent user imprecision and measurement inaccuracies, user observations are considered to be noise corrupted. Sequences of user observations are called *tasks* and the unknown set of all possible tasks that the user may perform is called the *repertoire*. Informally, the user generates a robot program by first selecting a task from the repertoire according to an unknown *a priori* distribution. When performing a task, each user observation is generated by some hidden state of the task. We assume that the user moves between states in the task according to a stationary, but arbitrary, probability distribution that depends only on the current state of the task. Since users tend not to think in terms of latent random processes, it is impractical to require a user to describe which *internal state* generated a given observation. Therefore, user observations are unlabeled in the sense that there is no "ground-truth" mapping between user observations and internal state. Since the internal states of the user are unknown and all information is conveyed to our system via noise-corrupted real-valued observations, we model user actions as Continuous-Density Hidden Markov Models (CDHMMs).

More formally, the model of user actions, or repertoire, is given by the 7-tuple $R = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{Q}_\lambda, \boldsymbol{\mathcal{X}}, a, b, \pi)$ where

- $\mathcal{Q}$ is a finite set of states;
- $\mathcal{Q}_0 \subset \mathcal{Q}$ is the non-empty set of starting states;
- $\mathcal{Q}_\lambda \subset \mathcal{Q}$ is the non-empty set of terminating states

$(\mathcal{Q}_0 \cap \mathcal{Q}_\lambda = \emptyset)$;

- $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^g$ is the observation set of dimension g;
- $a : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$ is the state-transition pmf: the probability of transitioning from state q_i to q_j is $a_{j|i} \triangleq \mathrm{P}(c_n{=}q_j|c_{n-1}{=}q_i, R), \sum_{q_j \in \mathcal{Q}\setminus\mathcal{Q}_0} a_{j|i}{=}1, \forall q_i \in \mathcal{Q}\setminus\mathcal{Q}_\lambda$;
- $b : \boldsymbol{\mathcal{X}} \times \mathcal{Q} \to [0, \infty)$ is the observation pdf: the likelihood of state q_j generating observation $\mathbf{x}_n$ is $b_j(\mathbf{x}_n) \triangleq \mathrm{p}(\mathbf{x}_n|c_n{=}q_j, R), \int_{\mathbf{x}_n \in \boldsymbol{\mathcal{X}}} b_j(\mathbf{x}_n) d\mathbf{x}_n{=}1, \forall q_j \in \mathcal{Q}$;
- $\pi : \mathcal{Q}_0 \to [0, 1]$ is the initial-state pmf, the probability of starting in state q_j is $\pi_j \triangleq \mathrm{P}(c_0{=}q_j|R)$, $\sum_{q_j \in \mathcal{Q}_0} \pi_j = 1$.

The random variable indicating the current state at time n is written c_n. Because all tasks that can be created by humans are necessarily finite in duration, the underlying graph of R is acyclic.

3. Learning from Prior Observations

In high-level terms, we assimilate observations from an arbitrary number of tasks to estimate the repertoire of the user. As a general philosophical assertion, it is desirable to create the most simple, or compact, model that explains the observations. This is of particular importance when training data are scarce, as simple models imply fewer free parameters to assign. Furthermore, in many interesting problems, it is not possible to know *a priori* the general structure of the model. Central to the development our algorithms are the low availibility of training data and an unknown graph topology. In Section 6, we refer to existing algorithms that exploit large corpora of data or knowledge of graph topology.

Since the topology of the repertoire is unknown, our system is initialized *tabula rasa*. From this blank slate, there appears to be two approaches to create compact models. The first approach is to assume a simple initial model and incrementally increase its complexity to account for inconsistencies between the simple model and the data. The second approach is to create a complex model initially and incrementally decrease its complexity by identifying similarities in the model. When using graph-based models, these approaches are known as *state splitting* and *state merging* respectively. There are potential advantages and disadvantages to each; neither is a clear-cut winner. Because state-merging schemes tend to start with sparse initial groupings of data, from our perspective, it seems relatively easy to identify similarities in these small groupings of data. This is in contrast to state-splitting schemes, which start with a large initial groupings of data and, consequently, tend to be a bit more difficult to identify the inconsistencies in these groupings. But, again, neither approach is a clear-cut winner.

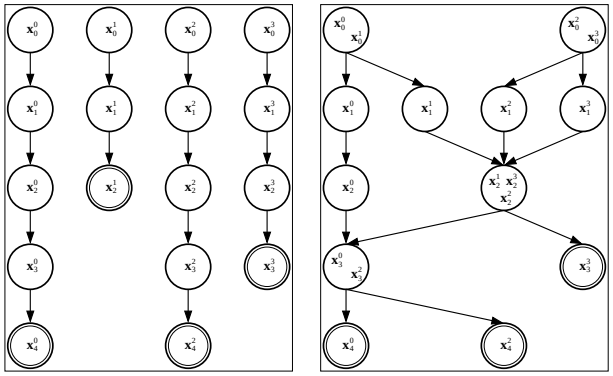

Figure 2. Hypothetical DAG before assimilating the set of prior observations (left) and after state-merging algorithm (right).

3.1 Derivation of the Learning Algorithm

Essentially, the algorithm operates by first building the maximal-state Directed Acyclic Graph (DAG), $G = (V, E)$, suggested by the prior user observations and then repeatedly merges statistically *similar* states so that a *simple* CDHMM results (Figure 2). Each node in the DAG, $v_i \in V$, is comprised of a multiset of observations, $\boldsymbol{\mathcal{X}}_{v_i} \subseteq \mathcal{M}(\boldsymbol{\mathcal{X}})$, where $\mathcal{M}(\mathcal{A})$ denotes any finite multiset over the set $\mathcal{A}$. The edges, $e_{v_i \to v_j} \in E$, contain usage counts, $N_{e_{v_i \to v_j}} \in \mathbb{Z}_{\geq 0}$. In a leveled DAG a node exists at only one level, or depth. Leveled DAGs are a proper subset of general DAGs. Let the set of nodes at depth n be written as $V_n \subset V$. Let the multiset of all tasks be $\mathbf{X} = \{\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^M\}$, where each task is a sequence of observations $\mathbf{X}^k = \{\mathbf{x}_0^k, \mathbf{x}_1^k, \ldots, \mathbf{x}_{N_k}^k\}$. The sample mean of the multiset $\mathcal{A}$ is written as $\langle \mathcal{A} \rangle$ and given by

$$\langle \mathcal{A} \rangle \equiv \frac{1}{|\mathcal{A}|} \sum_{x \in \mathcal{A}} x.$$

Definition 1. *A DAG is* ϵ-compact *if, for all nodes* $v_i \in V_n$ *and* $v_j \in V_n$ *at some common depth* n *and for some function* $d : \mathcal{M}(\boldsymbol{\mathcal{X}}) \times \boldsymbol{\mathcal{X}} \to [0, \infty)$,

- $d(\boldsymbol{\mathcal{X}}_{v_i}, \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle) \leq \epsilon$;
- $d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle) > \epsilon$, *for all* $v_i \neq v_j$.

for some $\epsilon \in [0, \infty)$.

Essentially, Definition 1 provides a precise meaning to a DAG being "simple". From a state-merging perspective, an ϵ-compact DAG has an irreducible number of nodes because there are no nodes that can be merged

and yield an ϵ-compact DAG. An ϵ-compact DAG does not imply a *globally* minimal number of nodes, however it does imply a type of locally optimal solution. Finding ϵ-compact DAGs efficiently (by state-merging, state-splitting, or otherwise) would be inherently useful in any problem searching for "simple" graph-based models, particularly those where the training data is severly limited. When and ϵ-compact DAG is converted to a CDHMM, this translates to fewer free parameters in the user model. There are many possible functions to evaluate ϵ-compactness and in this work we define

$$d(\boldsymbol{\mathcal{Y}}, \mathbf{u}) \triangleq \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} (\mathbf{x} - \mathbf{u})^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \mathbf{u}), \tag{1}$$

where $\mathbf{C}$ is a symmetric positive-definite (PD) matrix. Equation 1 is simply a sum of squared Mahalanobis distances with respect to a reference vector, $\mathbf{u}$.

Lemma 1. *$d(\boldsymbol{\mathcal{Y}}, \mathbf{u}) \geq 0$, for all $\boldsymbol{\mathcal{Y}} \subseteq \mathcal{M}(\boldsymbol{\mathcal{X}})$ and any $\mathbf{u} \in \boldsymbol{\mathcal{X}}$.*

Proof. by definition of PD matrix $\mathbf{C}$. The function will be strictly positive unless $\boldsymbol{\mathcal{Y}} = \emptyset$ or $\mathbf{u} = \mathbf{x}$, for all $\mathbf{x} \in \boldsymbol{\mathcal{Y}}$. □

Lemma 2. *$d(\boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}}, \mathbf{u}) = d(\boldsymbol{\mathcal{A}}, \mathbf{u}) + d(\boldsymbol{\mathcal{B}}, \mathbf{u})$, where $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, and $\boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}}$ are multisets.*

Proof.

$$\begin{aligned} d(\boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}}, \mathbf{u}) &\triangleq \sum_{\mathbf{x} \in \boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}}} (\mathbf{x} - \mathbf{u})^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \mathbf{u}) \\ &= \sum_{\mathbf{x} \in \boldsymbol{\mathcal{A}}} (\mathbf{x} - \mathbf{u})^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \mathbf{u}) + \sum_{\mathbf{x} \in \boldsymbol{\mathcal{B}}} (\mathbf{x} - \mathbf{u})^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \mathbf{u}) \\ &= d(\boldsymbol{\mathcal{A}}, \mathbf{u}) + d(\boldsymbol{\mathcal{B}}, \mathbf{u}). \end{aligned}$$

□

Lemma 3. *$\langle \boldsymbol{\mathcal{Y}} \rangle = \arg\min_{\mathbf{u} \in \boldsymbol{\mathcal{X}}} d(\boldsymbol{\mathcal{Y}}, \mathbf{u})$, where $\langle \boldsymbol{\mathcal{Y}} \rangle$ is the sample mean of the non-empty finite multiset $\boldsymbol{\mathcal{Y}} \subseteq \mathcal{M}(\boldsymbol{\mathcal{X}})$.*

Proof. Since $\mathbf{C}$ is symmetric

$$\begin{aligned} \mathbf{0} &= \frac{\partial}{\partial \mathbf{u}} \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} (\mathbf{x} - \mathbf{u})^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \mathbf{u}) \\ &= \frac{\partial}{\partial \mathbf{u}} \left(\sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} \mathbf{x}^{\mathsf{T}} \mathbf{C} \mathbf{x} - 2 \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} \mathbf{u}^{\mathsf{T}} \mathbf{C} \mathbf{x} + \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} \mathbf{u}^{\mathsf{T}} \mathbf{C} \mathbf{u} \right) \\ &= -2\mathbf{C} \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} \mathbf{x} + 2|\boldsymbol{\mathcal{Y}}| \mathbf{C} \mathbf{u}. \\ \Rightarrow \mathbf{u}^* &= \frac{1}{|\boldsymbol{\mathcal{Y}}|} \sum_{\mathbf{x} \in \boldsymbol{\mathcal{Y}}} \mathbf{x} \equiv \langle \boldsymbol{\mathcal{Y}} \rangle. \end{aligned}$$

This final step requires that $\mathbf{C}$ be invertible, which is always true since $\mathbf{C}$ is defined to be PD. To ensure a minimum, we check the Hessian. By taking the second derivative, we have

$$\frac{\partial^2}{\partial \mathbf{u} \partial \mathbf{u}^{\mathsf{T}}} d(\boldsymbol{\mathcal{Y}}, \mathbf{u}) = 2|\boldsymbol{\mathcal{Y}}| \mathbf{C},$$

which must be a symmetric PD matrix, by definition of $\mathbf{C}$ and assumption of $|\boldsymbol{\mathcal{Y}}| > 0$. This implies that $\langle \boldsymbol{\mathcal{Y}} \rangle$ is a minimum. □

Note that Lemma 3 is similar to the derivation of the sample mean for *iid*draws from a multivariate Gaussian using the log-ML approach.

Lemma 4. *For any non-negative threshold $\tau \in [0, \infty)$, there exists a submultiset $\widetilde{\boldsymbol{\mathcal{Y}}} \subseteq \boldsymbol{\mathcal{Y}}$ of the multiset $\boldsymbol{\mathcal{Y}}$ such that $\tau < d(\widetilde{\boldsymbol{\mathcal{Y}}}, \langle \widetilde{\boldsymbol{\mathcal{Y}}} \rangle) \iff \tau < d(\boldsymbol{\mathcal{Y}}, \langle \boldsymbol{\mathcal{Y}} \rangle)$.*

Proof.
$\Leftarrow$: since $\tau < d(\boldsymbol{\mathcal{Y}}, \langle \boldsymbol{\mathcal{Y}} \rangle)$, let $\widetilde{\boldsymbol{\mathcal{Y}}} = \boldsymbol{\mathcal{Y}}$ and then clearly $\tau < d(\widetilde{\boldsymbol{\mathcal{Y}}}, \langle \widetilde{\boldsymbol{\mathcal{Y}}} \rangle)$. $\Rightarrow$: we have $\widetilde{\boldsymbol{\mathcal{Y}}} \subseteq \boldsymbol{\mathcal{Y}}$ and

$$\begin{aligned} \tau &< d(\widetilde{\boldsymbol{\mathcal{Y}}}, \langle \widetilde{\boldsymbol{\mathcal{Y}}} \rangle) && \text{(By assumption)} \\ &\leq d(\widetilde{\boldsymbol{\mathcal{Y}}}, \langle \boldsymbol{\mathcal{Y}} \rangle) && \text{(Lemma 3)} \\ &\leq d(\widetilde{\boldsymbol{\mathcal{Y}}}, \langle \boldsymbol{\mathcal{Y}} \rangle) + d(\boldsymbol{\mathcal{Y}} \backslash \widetilde{\boldsymbol{\mathcal{Y}}}, \langle \boldsymbol{\mathcal{Y}} \rangle) && \text{(Lemma 1)} \\ &= d(\boldsymbol{\mathcal{Y}}, \langle \boldsymbol{\mathcal{Y}} \rangle) && \text{(Lemma 2)} \end{aligned}$$

which completes both sides of the claim. □

Corollary 4.1. *For any non-negative threshold $\tau \in [0, \infty)$ and multisets $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$, there exist submultisets $\widetilde{\boldsymbol{\mathcal{A}}} \subseteq \boldsymbol{\mathcal{A}}$ and $\widetilde{\boldsymbol{\mathcal{B}}} \subseteq \boldsymbol{\mathcal{B}}$ such that $\tau < d(\widetilde{\boldsymbol{\mathcal{A}}} \cup \widetilde{\boldsymbol{\mathcal{B}}}, \langle \widetilde{\boldsymbol{\mathcal{A}}} \cup \widetilde{\boldsymbol{\mathcal{B}}} \rangle) \iff \tau < d(\boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{A}} \cup \boldsymbol{\mathcal{B}} \rangle)$*

Proof. By direct extension of Lemma 4. □

The first three lemmas are mere properties of Equation 1. However, the final lemma and its corollary provide a road-map on how to derive an algorithm to construct ϵ-compact DAGs and, ultimately, *simple* CDHMMs. Essentially, if we encounter observation sets from two nodes such that

$$\epsilon < d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle), \tag{2}$$

then we can add *any* observation, $\mathbf{x}_n$, to either node, providing that $d(\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\} \rangle) \leq \epsilon$, and be assured that the inequality (Equation 2) will hold. The algorithm `Learn-HMM` is based on the result of, and the intuition gained from, Lemma 4 and Corollary 4.1. Figure 3 contains the pseudocode for constructing ϵ-compact DAGs.

Algorithm `Learn-HMM`
$\epsilon \in [0, \infty)$ is a similarity threshold.
$\mathbf{X} = \{\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^M\}$ is the multiset of tasks.

```
1: V := ∅, E := ∅
2: G := (V, E)
3: for all X^i ∈ {X^0, X^1, ..., X^M}
4:    G := assimilate-task (G, X^i, ε)
5: end for all
6: R := convert-DAG-to-HMM (G)
```

Figure 3. The complete HMM learning algorithm.

As a note to `Learn-HMM` (Figure 3), though the observation set of a CDHMM exists purely as a mathematical construct, it is simple to show that $\boldsymbol{\mathcal{X}}$ is the convex hull of the multiset of all observations, $\mathbf{X} = \{\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^M\}$, since the convex hull is closed under linear operators (this will be used in Section 4).

Theorem 5 (Compactness). *The* `Learn-HMM` *algorithm in Figure 3 produces ϵ-compact DAGs.*

Proof. Because the algorithm only creates edges between nodes at sequential depths (Line 12, Figure 4), it will produce only *leveled* DAGs, which are a proper subset of general DAGs. The first predicate of Definition 1 ($d(\boldsymbol{\mathcal{X}}_{v_i}, \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle) \leq \epsilon$) is trivially true since the algorithm will not add an observation, $\mathbf{x}_n$, to a node, v_i, unless $d(\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\} \rangle) \leq \epsilon$ (Line 3, Figure 4). Since all $\boldsymbol{\mathcal{X}}_{v_i} = \emptyset$ initially (Line 1, Figure 3 and Line 8, Figure 4) and $d(\{\mathbf{x}\}, \langle \{\mathbf{x}\} \rangle) = 0, \forall \mathbf{x} \in \boldsymbol{\mathcal{X}}$, therefore all nodes $d(\boldsymbol{\mathcal{X}}_{v_i}, \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle) \leq \epsilon$ at any point during the execution of the algorithm.

We now move on to the second predicate of Definition 1 ($d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle) > \epsilon$, for all $v_i \neq v_j$). Let us take two nodes, v_i and v_j, at the same depth in the DAG. Without loss of generality, we assume that node v_j was created after v_i. We know that nodev_j would never have been created unless there exists an observation $\mathbf{x}_{v_j} \in \boldsymbol{\mathcal{X}}_{v_j}$ and a submultiset $\widetilde{\boldsymbol{\mathcal{X}}_{v_i}} \subseteq \boldsymbol{\mathcal{X}}_{v_i}$ such that $\epsilon < d(\widetilde{\boldsymbol{\mathcal{X}}_{v_i}} \cup \{\mathbf{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{X}}_{v_i}} \cup \{\mathbf{x}_{v_j}\} \rangle)$ (Line 7, Figure 4). From Lemma 4 and Corollary 4.1, we have shown

$$\begin{aligned} \epsilon &< d(\widetilde{\boldsymbol{\mathcal{X}}_{v_i}} \cup \{\mathbf{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{X}}_{v_i}} \cup \{\mathbf{x}_{v_j}\} \rangle) \\ &\iff \\ \epsilon &< d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle) \end{aligned}$$

Since we never remove observations from nodes, and all nodes always have corresponding non-empty observation multisets (Line 10, Figure 4), the second predicate is true at any point during the execution of the algorithm. Therefore both predicates of Definition 1 are met and the claim holds. □

Function `assimilate-task`
$G = (V, E)$ is the leveled DAG.
$\mathbf{X}^i = \{\mathbf{x}^i_0, \mathbf{x}^i_1, \ldots, \mathbf{x}^i_{N_i}\}$ is the task to assimilate.
$\epsilon \in [0, \infty)$ is a similarity threshold.

```
1: for all x_n ∈ {x^i_0, x^i_1, ..., x^i_{N_i}}
2:    ε_min := min_{v_i ∈ V_n} d(X_{v_i} ∪ {x_n}, ⟨X_{v_i} ∪ {x_n}⟩)
3:    if ε_min ≤ ε then
4:       v_j := arg min_{v_i ∈ V_n} d(X_{v_i} ∪ {x_n}, ⟨X_{v_i} ∪ {x_n}⟩)
5:       X_{v_j} := X_{v_j} ∪ {x_n}
6:    end if
7:    else if ε < ε_min then
8:       create empty node v_j
9:       V := V ∪ {v_j}
10:      X_{v_j} := {x_n}
11:      if n > 0 then
12:         E := E ∪_{v_k ∈ V^{n-1}} {e_{v_k → v_j}}
13:      end if
14:      if n < N_i − 1 then
15:         E := E ∪_{v_k ∈ V^{n+1}} {e_{v_j → v_k}}
16:      end if
17:   end else if
18:   if n > 0 then
19:      N_{e_{v_k → v_j}} := N_{e_{v_k → v_j}} + 1
20:   end if
21:   v_k := v_j
22: end for all
23: return (V, E)
```

Figure 4. Assimilating a task into a leveled DAG.

The learning algorithm, `Learn-HMM` , relies on a non-negative scalar threshold, $\epsilon \in [0, \infty)$. While this is a mathematically correct approach, this threshold does not have an obvious real-world analogue that gives the user insight into the operation of the algorithm. We can step back from the formulation and recast the algorithm using Probably Approximately Correct (PAC) learning. In the PAC-learning paradigm, we are given a probability threshold $\delta \in (0, 1]$ such that

$$\Pr\{d(\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\} \rangle) \leq \epsilon\} > 1 - \delta.$$

In words, the distance function must be less that ϵ with probability of at least $1 - \delta$. Here we assume that the user makes errors *independent* of other tasks and follows a Gaussian distribution, $\mathcal{N}(\langle \boldsymbol{\mathcal{X}}_{v_i} \rangle, \boldsymbol{\Sigma})$. For notational convenience we define $\boldsymbol{\mu} \triangleq \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\} \rangle$.

Function `convert-DAG-to-HMM`
$G = (V, E)$ is the leveled DAG.

```
 1: N := maximum depth of G
 2: Q := ∅, Q_0 := ∅, Q_λ := ∅
 3: for n := 0, ..., N − 1
 4:   for all v_i ∈ V^n
 5:     create repertoire state q_i
 6:     create b_i from X_{v_i}
 7:     if n < N − 1 then
 8:       for all v_j ∈ V^{n+1}
 9:         a_{j|i} := N_{e_{v_i→v_j}} / |X_{v_i}|
10:       end for all
11:     end if
12:     else if n = N − 1 then
13:       Q_λ := Q_λ ∪ {q_i}
14:     end else if
15:     if n = 0 then
16:       π_i := |X_{v_i}| / Σ_{v_j ∈ V^0} |X_{v_j}|
17:       Q_0 := Q_0 ∪ {q_i}
18:     end if
19:     Q := Q ∪ {q_i}
20:   end for all
21: end for
22: return (Q, Q_0, Q_λ, X, a, b, π)
```

Figure 5. Converting a leveled DAG to an HMM.

In this case

$$
\begin{aligned}
&\mathrm{p}\big(\mathbf{x}_0^{v_i}, \ldots, \mathbf{x}_{N_i-1}^{v_i}, \mathbf{x}_n | \boldsymbol{\mu}, \boldsymbol{\Sigma}\big) \\
&\quad = \prod_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}} \mathrm{p}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\
&\quad = \prod_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}} \frac{\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}}{\sqrt{(2\pi)^g|\boldsymbol{\Sigma}|}} \\
&\quad = \frac{\exp\left\{-\frac{1}{2}\sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}} (\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}}{\sqrt{(2\pi)^g|\boldsymbol{\Sigma}|}^{|\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}|}} \\
&\quad \doteq \frac{\exp\left\{-\frac{1}{2}d(\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}, \boldsymbol{\mu})\right\}}{\sqrt{(2\pi)^g|\boldsymbol{\Sigma}|}^{|\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}|}} \\
&\quad \leq \frac{\exp\left\{-\frac{1}{2}\epsilon\right\}}{\sqrt{(2\pi)^g|\boldsymbol{\Sigma}|}^{|\boldsymbol{\mathcal{X}}_{v_i} \cup \{\mathbf{x}_n\}|}}.
\end{aligned}
$$

For a given probability threshold, $\delta \in (0, 1]$ we can find the similarity threshold, $\epsilon \in [0, \infty)$ as

$$\epsilon \in \left\{ y \,\middle|\, \frac{\delta}{2} = Q(y) \right\}, \tag{3}$$

where $Q(y)$ is the Q-function, the area under a zero-mean unit Gaussian from y to infinity ($Q(y) = \int_y^\infty \mathcal{N}(0,1)dt$) and has a unique solution. Essentially, Equation 3 determines the hyperellipse within which the observations from a *latent state* in the repertoire must lie.

3.2 Running Time of the Learning Algorithm

We now turn our attention to the running-time of the `Learn-HMM` algorithm. It is impossible to know *a priori* how many nodes will be created by the algorithm, and it appears unlikely that some type of expectation (or "average case") can be constructed. However, it is possible to determine the worst-case running-time of the algorithm. The worst-case scenario is when no nodes are merged; every observation in each task is a node unto itself. Then the number of nodes in the DAG is equal to the number of observations assimilated at that time. Though not required by the algorithm, we simplify the running-time analysis by assuming that all tasks are of a common length N. Let the number of tasks be M. In the worst case, when the algorithm has assimilated all observations, there will be MN nodes in the DAG. The learning algorithm operates in two decoupled steps by first assimilating observations into a DAG (Line 3, Figure 3) and then converting the DAG to a CDHMM (Line 6, Figure 3).

By glancing at the pseudocode in Figure 4, it is clear that the cost of computing Equation 1 dominates the computation. Before analysing the cost of assimilating observations, we take note of some sufficient statistics that will be computed. Again, the observations assigned to node v_i is denoted by the multiset $\boldsymbol{\mathcal{X}}_{v_i} \in \mathcal{M}(\boldsymbol{\mathcal{X}})$. The sufficient statistics for Equation 1 are the cardinality of the observation multiset, the scalar weighted-quadratic sum of observations, and the vector sum of observations

$$\xi_{v_i} \triangleq |\boldsymbol{\mathcal{X}}_{v_i}|; \quad \kappa_{v_i} \triangleq \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} \mathbf{x}^{\mathsf{T}}\mathbf{C}\mathbf{x}; \quad \boldsymbol{\sigma}_{v_i} \triangleq \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} \mathbf{x}.$$

These sufficient statistics have simple update rules when taking the union with another multiset. From

the definition of Equation 1 we get the expansion

$$\begin{aligned} d(\boldsymbol{\mathcal{X}}_{v_i}, \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle) &\triangleq \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} (\mathbf{x} - \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle)^{\mathsf{T}} \mathbf{C} (\mathbf{x} - \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle) \\ &= \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} \mathbf{x}^{\mathsf{T}} \mathbf{C} \mathbf{x} - 2 \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle^{\mathsf{T}} \mathbf{C} \mathbf{x} + \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}_{v_i}} \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle^{\mathsf{T}} \mathbf{C} \langle \boldsymbol{\mathcal{X}}_{v_i} \rangle \\ &\doteq \kappa_{v_i} - \frac{2}{\xi_{v_i}} \boldsymbol{\sigma}_{v_i}{}^{\mathsf{T}} \mathbf{C} \boldsymbol{\sigma}_{v_i} + \frac{\xi_{v_i}}{\xi_{v_i}^2} \boldsymbol{\sigma}_{v_i}{}^{\mathsf{T}} \mathbf{C} \boldsymbol{\sigma}_{v_i} \\ &= \kappa_{v_i} - \frac{1}{\xi_{v_i}} \boldsymbol{\sigma}_{v_i}{}^{\mathsf{T}} \mathbf{C} \boldsymbol{\sigma}_{v_i}. \end{aligned}$$

The sufficient statistics for the union of two multisets, $\boldsymbol{\mathcal{X}}_{v_k} = \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}$, are simply computed as

$$\xi_{v_k} = \xi_{v_i} + \xi_{v_j}; \quad \kappa_{v_k} = \kappa_{v_i} + \kappa_{v_j}; \quad \boldsymbol{\sigma}_{v_k} = \boldsymbol{\sigma}_{v_i} + \boldsymbol{\sigma}_{v_j}.$$

Using the sufficient statistics and their corresponding update equations, the computational cost of computing Equation 1 is *independent* of the cardinality of either multiset. The cost of computing this union is dominated by computing the weighted-quadratic sums, κ_{v_i} and $\boldsymbol{\sigma}_{v_i}{}^{\mathsf{T}} \mathbf{C} \boldsymbol{\sigma}_{v_i}$. These operations cost $\mathcal{O}(g^2)$, where g is the dimension of an observation vector. We compute Equation 1 N times the number of tasks already assimilated (Line 2, Figure 4), resulting in the arithmetic series

$$\mathcal{O}(\sum_{m=1}^{M} mNg^2) \in \mathcal{O}(M^2 N g^2).$$

As a result, the worst-case cost of assimilating M tasks of length N with an observation vector of dimension g is order $\mathcal{O}(M^2 N g^2)$.

The second phase of the algorithm converts a DAG into a CDHMM. The computation time of converting a DAG to a CDHMM is dominated by the cost of computing the state-transition probabilities (Line 9, Figure 5). From the worst-case scenario assumption, we have MN nodes in the DAG and due to the leveled property of the DAGs, we know that there are M nodes at each depth. The inner loop for computing transition probabilities (Line 9, Figure 5) is a double arithmetic series on M *and* N. Each pass through the loop, we must create an observation-pdf (Line 6, Figure 5), which we define to be of cost $\mathcal{O}(b)$. This implies that the computation complexity of converting the worst-case MN-node DAG to a CDHMM is $\mathcal{O}(M^2 N^2 b)$.

Each phase in the algorithm executes only once in a strict serial fashion and the respective complexities are additive. Therefore, the worst-case scenario of M tasks each of length N is

$$\mathcal{O}(M^2 N(g^2 + Nb)).$$

4. Predicting User Actions

Once the repertoire has been estimated, we use this model to predict future observations. While the user is performing the current task, we compute the prediction for the next observation, $\widehat{\mathbf{x}}_n$. There are many potential criteria for this optimal estimator, such as Maximum Likelihood (ML), Maximum *A Posteriori* (MAP), Minimum Entropy (ME), etc. Our criterion is the *expected value* (EV) of the next observation, given the prior observations, the estimated user repertoire, and observations from the current task. Before deriving the estimator, we first define some notation. The current task is $\mathbf{X}^c = \{\mathbf{x}_0^c, \ldots, \mathbf{x}_{n-1}^c\}$ and a subtask is written $\mathbf{X}_{m:n}^i \triangleq \{\mathbf{x}_m^i, \ldots, \mathbf{x}_n^i\}$. The "forward" variables are given by

$$\begin{aligned} \alpha_j^n &\triangleq \mathrm{p}(c_n{=}q_j, \mathbf{X}_{0:n}^c | R) \\ &= \sum_{q_i \in \mathcal{Q}} \mathrm{p}(\mathbf{x}_n | c_n{=}q_j, R)\, \mathrm{p}\big(c_n{=}q_j, c_{n-1}{=}q_i, \mathbf{X}_{0:n-1}^c | R\big) \\ &\doteq \begin{cases} b_j(\mathbf{x}_n) \sum_{q_i \in \mathcal{Q}} a_{j|i} \alpha_i^{n-1}, & n > 0 \\ b_j(\mathbf{x}_0) \pi_j, & n = 0 \end{cases}. \end{aligned} \tag{4}$$

The probability of being in state q_j at the next time step, given all observations, is the pmf

$$\begin{aligned} \nu_j^n &\triangleq \mathrm{P}\big(c_n{=}q_j | \mathbf{X}_{0:n-1}^c, R\big) \\ &= \frac{\mathrm{p}\big(c_n{=}q_j, \mathbf{X}_{0:n-1}^c | R\big)}{\mathrm{p}\big(\mathbf{X}_{0:n-1}^c | R\big)} \\ &= \frac{\sum_{q_i \in \mathcal{Q}} \mathrm{p}\big(c_n{=}q_j, c_{n-1}{=}q_i, \mathbf{X}_{0:n-1}^c | R\big)}{\sum_{q_k \in \mathcal{Q}} \mathrm{p}\big(c_{n-1}{=}q_k, \mathbf{X}_{0:n-1}^c | R\big)} \\ &= \frac{\sum_{q_i \in \mathcal{Q}} \mathrm{P}(c_n{=}q_j | c_{n-1}{=}q_i, R)\, \mathrm{p}\big(c_{n-1}{=}q_i, \mathbf{X}_{0:n-1}^c | R\big)}{\sum_{q_k \in \mathcal{Q}} \mathrm{p}\big(c_{n-1}{=}q_k, \mathbf{X}_{0:n-1}^c | R\big)} \\ &\doteq \frac{\sum_{q_i \in \mathcal{Q}} a_{j|i} \alpha_i^{n-1}}{\sum_{q_k \in \mathcal{Q}} \alpha_k^{n-1}}. \end{aligned} \tag{5}$$

Using these definitions, the EV estimator is computed as

$$\begin{aligned} \widehat{\mathbf{x}}_n &= \mathrm{E}_{\mathbf{x} | \mathbf{X}_{0:n-1}^c, R}\{\mathbf{x}\} \\ &= \mathrm{E}_{c_n | \mathbf{X}_{0:n-1}^c, R}\Big\{ \mathrm{E}_{\mathbf{x} | c_n, \mathbf{X}_{0:n-1}^c, R}\{\mathbf{x}\} \Big\} \\ &= \sum_{q_j \in \mathcal{Q}} \mathrm{P}\big(c_n{=}q_j | \mathbf{X}_{0:n-1}^c, R\big) \int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathbf{x} \mathrm{p}(\mathbf{x} | c_n{=}q_j, R)\, d\mathbf{x} \\ &\doteq \sum_{q_j \in \mathcal{Q}} \nu_j^n \int_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \mathbf{x} b_j(\mathbf{x}) d\mathbf{x}. \end{aligned} \tag{6}$$

Essentially, Equation 6 says that the expected value of the next observation is the expected value of each state (the mean of the observation-generation pdf, $\int \mathbf{x} b_j(\mathbf{x}) d\mathbf{x}$), weighted by the probability that the state generates the next observation (ν_j^n). We can also repeatedly apply Equation 5 to compute the expectation of observations at any point in the future.

4.1 Confidence

Regardless of the criterion used (ML, MAP, ME, EV), a prediction will exist even if the observations are not consistent with the estimated repertoire. Prediction schemes must incorporate a value that indicates how *confident* the system is in its prediction. The confidence value, $\phi_n \in [0, 1]$, should indicate the *internal* model uncertainty arising from observing the current task. Confidence of $\phi_n = 1$, indicates that the observations fit perfectly with the model, while $\phi_n = 0$ indicates minimum certainty. The confidence value should also be independent of the number of observations and states in the estimated repertoire. In our work, we define confidence as the Kullback-Leibler divergence taken log base $|\mathcal{Q}|$ between the next-state pmf, $\nu_\cdot^n$ and the uniform distribution

$$\begin{aligned}
\phi_n &\triangleq \frac{\mathcal{D}_{\mathrm{KL}}(\nu_\cdot^n \parallel \frac{1}{|\mathcal{Q}|})}{\log_2 |\mathcal{Q}|} \qquad (7)\\
&= \frac{\sum_{q_j \in \mathcal{Q}} \nu_j^n \log_2(|\mathcal{Q}| \nu_j^n)}{\log_2 |\mathcal{Q}|}\\
&= \frac{\sum_{q_j \in \mathcal{Q}} \nu_j^n \left(\log_2 |\mathcal{Q}| + \log_2 \nu_j^n\right)}{\log_2 |\mathcal{Q}|}\\
&= \frac{\log_2 |\mathcal{Q}| \sum_{q_j \in \mathcal{Q}} \nu_j^n}{\log_2 |\mathcal{Q}|} + \frac{\sum_{q_j \in \mathcal{Q}} \nu_j^n \log_2 \nu_j^n}{\log_2 |\mathcal{Q}|}\\
&= 1 - \frac{\sum_{q_j \in \mathcal{Q}} -\nu_j^n \log_2 \nu_j^n}{log_2 |\mathcal{Q}|}\\
&= 1 - \frac{H(c_n)}{log_2 |\mathcal{Q}|}.
\end{aligned}$$

Since c_n is a discrete random variable with $|\mathcal{Q}|$ possible outcomes, its entropy is $H(c_n) \in [0, \log_2 |\mathcal{Q}|]$, implying $\phi_n \in [0, 1]$. Loosely speaking, ϕ_n is the distance of the current uncertainty of the model from complete uncertainty (the uniform density). When the confidence, ϕ_n, exceeds some predetermined threshold then $\widehat{\mathbf{x}}_n$ is considered a valid prediction.

4.2 Computational Complexity of Prediction

The cost of computing predictions from Equation 6 has two components. The first involves taking the expectation of all observation pdfs. Since most pdfs are parameterized by their mean, its expectation can be computed in constant time. The second part involves computing the next-state pmf, Equation 5. This equation can be computed at standard dynamic-programming time, though we could take advantage of the leveled, sparse connectivity of the CDHMM to reduce the computation time greatly. Let the number of states be $|\mathcal{Q}|$, the current time be n, and the cost of evaluating an observation pdf be b. Each of $|\mathcal{Q}|$ states costs $\mathcal{O}(|\mathcal{Q}|b)$ at each of n time steps. This recurrance relation leads to a computational cost for prediction of

$$\mathcal{O}(n|\mathcal{Q}|^2 b).$$

5. Predictive Robot Programming

Programming a manipulator robot is an arduous task. A typical robot program consists of three main components: a sequence of positions through which the robot must travel, conditional branching statements, and process-specific instructions. Of these constituent problems, users spend the bulk of their time merely defining the sequence of positions, called waypoints. While critical to the success of all robot programs, defining the waypoints is currently an overly complex and time-consuming process.

Robot programming has evolved into two mutually exclusive paradigms, offline and online programming, each having its advantages and disadvantages. In offline programming, users move a simulated version of the robot to each waypoint using a CAD model of the workspace. In online programming, users move the robot itself to each waypoint using some type of control device in the actual workspace.

Offline-programming packages allow users to design a robot program in simulation without bringing down production and can optimize according to almost any imaginable criterion. Typical optimizations involve production speed, material usage, and power consumption. Offline packages generally require that programs be written in a sophisticated procedural-programming language. The transfer of the offline program to the robot controller requires translating the offline programming language to a form that the robot can understand. Not surprisingly, arcane problems can occur during this translation, especially with process-specific instructions, controller models, and inverse kinematics. To achieve the high accuracy required in many applications, the physical workspace must be well calibrated with the simulated environment. Otherwise, online fine-tuning will be needed, which detracts from the largest benefit of offline programming: lack of production downtime.

Despite the advantages of offline packages, online programming is, by far, more commonly used in practice. In online programming, an actual part is placed in the workspace exactly as it would be during production and the user moves the end-effector between waypoints using some type of control device, typically a joystick or push buttons. Even though online systems generate procedural-programming code, users can create waypoints without editing this code and, as a result, is typically viewed as less intimidating and more intuitive than offline programming. One potentially extreme disadvantage of online systems is that production and programming cannot occur in parallel, meaning production must be halted during reprogramming. If reprogramming cannot be completed during normal downtime, such as weekends, then the company will incur cost in the form of lost production. Therefore, the set of tasks viable for online programming is constrained by programming time. A reduction in programming time would permit robots into areas previously off limits. Although users generally view online programming as less intimidating, they still spend an inordinate amount of time simply moving the robot between waypoints instead of transferring any real knowledge. Furthermore, due to the difficult robot-positioning process, users tend to discard their previous work and create waypoints from scratch every time. This is very wasteful since robot programs tend to contain repeated subtasks and product designs contain many similarities to previous designs.

We have developed a Predictive Robot-Programming (PRP) system that allows users to leverage their previous work to decrease future programming time. Specifically, this system aims to assist users by predicting where they may move the end-effector and automatically positioning the robot at the predicted waypoint. The premise of PRP is that previous actions are useful in predicting the future. This is usually the case since, as mentioned earlier, robot programs tend to contain many similarities and common patterns. Complicating any prediction scheme is the inherent imprecision and poor repeatability of humans. Even with perfectly accurate sensors there will be uncertainty as to what task the user was trying to perform. Thus, any PRP system must incorporate the notion of uncertainty during its operation and our PRP system directly addresses uncertainty during both modeling and prediction.

User observations in robot programming are the waypoints of the program. General-purpose six degree-of-freedom (6DOF) manipulators can be described by a Cartesian-space location and orientation of the end-effector. In this work, the surjective Cartesian-space description is preferable to the bijective joint-space representation because the Cartesian-space description forms a kinematics-free coordinate transform. Using this Cartesian-space end-effector description, a waypoint is given by the homogeneous coordinate transform matrix ${}^{n}\mathbf{w}_{G}$ that maps some global reference frame G to the frame of the nth waypoint. A robot program can be described by the sequence of waypoints $\mathbf{W} = \{{}^{G}\mathbf{w}_0, {}^{G}\mathbf{w}_0, \ldots, {}^{G}\mathbf{w}_N\}$ that specify the location and orientation of the end-effector at discrete intervals. A homogeneous coordinate-frame transform, ${}^{a}\mathbf{w}_{b}$, is defined by the matrix

$$ {}^{a}\mathbf{w}_{b} \triangleq \left[\begin{array}{c|c} {}^{a}\boldsymbol{R}_{b} & {}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right], $$

where ${}^{a}\boldsymbol{R}_{b}$ is a (3×3) orthonormal matrix describing the rotation from frame a to frame b and the (3×1) vector ${}^{c}_{a}\boldsymbol{p}_{b}$ specifies the translation from frame a to frame b in coordinate frame c. Many references, *e.g.* (Craig, 1989), have derived the following properties from the definitions above

$$ \begin{aligned} {}^{c}_{a}\boldsymbol{p}_{b} &= -{}^{c}_{b}\boldsymbol{p}_{a}, \\ {}^{c}_{a}\boldsymbol{p}_{b} &= {}^{c}_{a}\boldsymbol{p}_{d} + {}^{c}_{d}\boldsymbol{p}_{b}, \\ {}^{c}_{a}\boldsymbol{p}_{b} &= {}^{c}\boldsymbol{R}_{d}\,{}^{d}_{a}\boldsymbol{p}_{b}, \\ {}^{a}\boldsymbol{R}_{b} &= {}^{a}\boldsymbol{R}_{c}\,{}^{c}\boldsymbol{R}_{b}, \\ {}^{a}\boldsymbol{R}_{b} &= ({}^{b}\boldsymbol{R}_{a})^{-1} = ({}^{b}\boldsymbol{R}_{a})^{\mathsf{T}}, \\ {}^{a}\boldsymbol{R}_{a} &= \mathbf{I}. \end{aligned} $$

The product of two homogeneous coordinate transforms, called "compounding", is written

$$ \begin{aligned} {}^{a}\mathbf{w}_{b}\,{}^{b}\mathbf{w}_{c} &= \left[\begin{array}{c|c} {}^{a}\boldsymbol{R}_{b} & {}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \left[\begin{array}{c|c} {}^{b}\boldsymbol{R}_{c} & {}^{b}_{b}\boldsymbol{p}_{c} \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^{a}\boldsymbol{R}_{b}\,{}^{b}\boldsymbol{R}_{c} & {}^{a}\boldsymbol{R}_{b}\,{}^{b}_{b}\boldsymbol{p}_{c} + {}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^{a}\boldsymbol{R}_{c} & {}^{a}_{b}\boldsymbol{p}_{c} + {}^{a}_{a}\boldsymbol{p}_{b} \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &= \left[\begin{array}{c|c} {}^{a}\boldsymbol{R}_{c} & {}^{a}_{a}\boldsymbol{p}_{c} \\ \hline \mathbf{0} & 1 \end{array}\right] \\ &\doteq {}^{a}\mathbf{w}_{c}. \end{aligned} \tag{8} $$

Essentially, Equation 8 describes the relative change in rotation and translation between the frames a and c. The inverse transform ${}^{b}\mathbf{w}_{a}$, which "undoes" the trans-

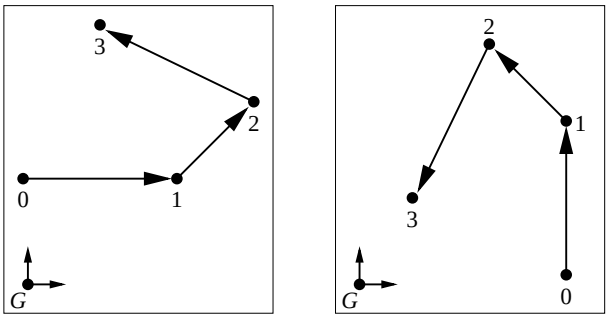

Figure 6. Though different in the global reference frame, G, the relative-movement information between these two programs is the same.

form ${}^a\mathbf{w}_b$, can be derived from the earlier properties

$$\begin{aligned}
{}^b\mathbf{w}_a &= \left[\begin{array}{c|c} {}^b\boldsymbol{R}_a & {}^b_b\boldsymbol{p}_a \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^a\boldsymbol{R}_b)^{\mathsf{T}} & -{}^b_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^a\boldsymbol{R}_b)^{-1} & -{}^b\boldsymbol{R}_a\,{}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} ({}^a\boldsymbol{R}_b)^{-1} & -({}^a\boldsymbol{R}_b)^{-1}\,{}^a_a\boldsymbol{p}_b \\ \hline \mathbf{0} & 1 \end{array}\right] \\
&= ({}^a\mathbf{w}_b)^{-1}.
\end{aligned}$$

Note that the inverse transform *always* exists since the only matrix inversion involves an orthonormal matrix, which must be full rank.

As we demonstrate shortly, capturing the relative-movement information, instead of absolute-position information, has the distinct advantage of yielding rotation and translation independence. For example, the two robot programs shown in Figure 6 are quite different in absolute terms, but have the same relative-movement description. Using this relative-waypoint representation, the PRP system is able recognize patterns *and* predict waypoints independent of rotation and translation.

Definition 2. *Given a sequence of homogeneous coordinate-frame transforms with respect to a common frame, G,*

$$\mathbf{W} = \{{}^G\mathbf{w}_0, {}^G\mathbf{w}_0, \ldots, {}^G\mathbf{w}_N\},$$

its Sequential Relative Homogeneous Coordinate-frame Transform (*SRHCT*) *is given by*

$$\begin{aligned}
\widetilde{\mathbf{W}} &= \{{}^0\mathbf{w}_G\,{}^G\mathbf{w}_1, {}^1\mathbf{w}_G\,{}^G\mathbf{w}_0, \ldots, {}^{N-1}\mathbf{w}_G\,{}^G\mathbf{w}_N\} \\
&= \{{}^0\mathbf{w}_1, {}^1\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.
\end{aligned}$$

Theorem 6 (Transform Independence). *A robot program specified by its SRHCT is independent of an initial rotation and translation.*

Proof. Suppose we have a robot program with an arbitrary number of waypoints, $\mathbf{W} = \{{}^G\mathbf{w}_0, {}^G\mathbf{w}_0, \ldots, {}^G\mathbf{w}_N\}$. We give the robot program an initial rotation and translation by pre-multiplying each waypoint by some homogeneous coordinate transform, ${}^H\mathbf{w}_G$, to yield

$$\begin{aligned}
\mathbf{W}^H &= \{{}^H\mathbf{w}_G\,{}^G\mathbf{w}_0, {}^H\mathbf{w}_G\,{}^G\mathbf{w}_1, \ldots, {}^H\mathbf{w}_G\,{}^G\mathbf{w}_N\} \\
&= \{{}^H\mathbf{w}_0, {}^H\mathbf{w}_1, \ldots, {}^H\mathbf{w}_N\}.
\end{aligned}$$

The SRHCT of the rotation and translated program is then

$$\begin{aligned}
\widetilde{\mathbf{W}}^H &= \{{}^0\mathbf{w}_H\,{}^H\mathbf{w}_1, {}^1\mathbf{w}_H\,{}^H\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_H\,{}^H\mathbf{w}_N\} \\
&= \{{}^0\mathbf{w}_1, {}^1\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.
\end{aligned}$$

Note that the SRHCT of the original robot program is, by definition,

$$\begin{aligned}
\widetilde{\mathbf{W}} &= \{{}^0\mathbf{w}_G\,{}^G\mathbf{w}_1, {}^1\mathbf{w}_G\,{}^G\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_G\,{}^G\mathbf{w}_N\} \\
&= \{{}^0\mathbf{w}_1, {}^1\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}.
\end{aligned}$$

Since $\widetilde{\mathbf{W}}^H \equiv \widetilde{\mathbf{W}}$, irrespective of the initial arbitrary rotation and translation, a robot program specified by its SRHCT yields rotation and translation independence. □

Theorem 6 only implies that robot programs can be *recognized* independent of an initial rotation and translation. We now develop the simple equations needed to predict waypoints in a rotation- and translation-independent manner. A task is then comprised of the relative waypoints of a robot program and, for convenience, we rewrite each relative waypoint into the stacked column vector, $\mathbf{x}_n = \mathrm{vec}({}^n\mathbf{w}_{n+1})$. The ith SRHCT robot program is written as $\mathbf{X}^i = \{\mathbf{x}^i_0, \mathbf{x}^i_1, \ldots, \mathbf{x}^i_{N_i}\}$. Then the sequence of prior observations to the algorithm `Learn-HMM` is the sequence of SRHCT robot programs $\mathbf{X} = \{\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^M\}$. Note that there is no requirement that the robot programs have a common length or even describe the same physical task. The algorithm could assimilate tasks corresponding to *e.g.* arc welding a bed frame, spot welding a ship hull, painting a car, etc. Once the robot programs have been assimilated into a CDHMM and the user begins creating a new robot program, we compute predictions of the next waypoint, $\widehat{\mathbf{x}}_n$, based on Equation 6. However, $\widehat{\mathbf{x}}_n$ is a stacked column vector of the relative waypoint ${}^{n-1}\widehat{\mathbf{w}}_n$, which specifies the expected relative movement of the end-effector from its

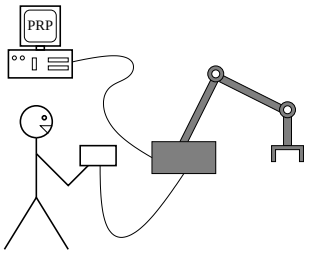

Figure 7. Diagram of the physical setup of the PRP.

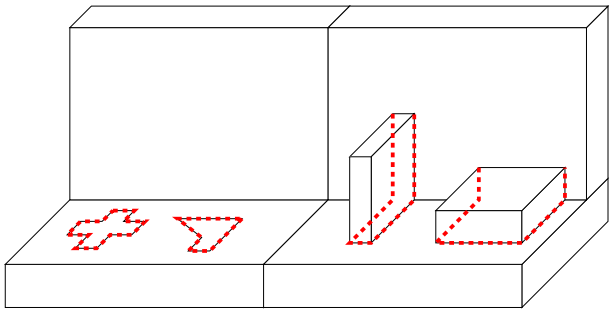

Figure 8. The four patterns for user demonstrations with thirteen, six, ten, and ten waypoints respectively.

current position, ${}^{G}\mathbf{w}_{n-1}$. Therefore the expected position of the next waypoint is found by postmultiplying the estimator, ${}^{n-1}\widehat{\mathbf{w}}_n$ by the current position of the end-effector in the global frame

$$ {}^{G}\widehat{\mathbf{w}}_n = {}^{G}\mathbf{w}_{n-1}\,{}^{n-1}\widehat{\mathbf{w}}_n $$

In our physical implementation of the system, the user presses a button on the teach pendant to indicate that the current end-effector position should be considered the next waypoint in the current program, shown in Figure 7. As mentioned earlier, in robot programming, commonality occurs in a subtasks not entire tasks. With this in mind, the PRP system considers observations up to a *horizon*. Considering a fixed horizon, h, of observations changes the initial conditions on Equation 4 to be $\alpha_j^{n-h} = b_j(\mathbf{x}_{n-h})/|\mathcal{Q}|$.

5.1 Experimental Results

We collected user observations from forty-four robot programs that described different tasks. In our terminology, the *repertoire* of the user consists of the four tasks shown in Figure 8. The two right-most programs, comprised of ten waypoints each, represent standard arc-welding tasks. The two left-most programs, comprised of thirteen and six waypoints respectively, are planar geometrical movements. To create the program, a user moves the robot end-effector with a joystick (*cf.* Figure 9). When the user feels that the

Figure 9. Creating waypoints for the patterns in Figure 8 with an ABB IRB140.

end-effector is sufficiently close to the waypoint, she presses a button on the teach pendant. The forty-four robot programs were assimilated into one HMM that described the repertoire of the user. The algorithm took well under a second to construct the estimated repertoire model from the programs (Figure 10 and Figure 11). To give some intuition regarding the δ parameter (*cf.* Equation 3), we constructed the HMM in Figure 10 (39 states, 51 transitions) using a small value of δ, while Figure 11 (85 states, 139 transitions) was constructed using a larger value of δ. It is typical that larger δ results in more HMM states because a larger δ necessarily results in a smaller ϵ. A small ϵ generally gives the algorithm fewer opportunities to combine DAG states (Line 3, Figure 4). As with any machine-learning algorithm, there is a danger of overfitting the data if δ becomes "too large" and does not allow the algorithm to generalize, which corresponds to merging states in this algorithm.

To determine the accuracy of the system, we computed the prediction performance of the system on the human-generated data. The data set was too small for a cross-validation set, so we computed the leave-one-out statistics (sometimes called the *jackknife estimator*, (Duda et al., 2001)) of the Cartesian prediction error, summarized in Table 1. With an appropriate value of δ, the average prediction error of the system was roughly 2.5 centimeters. The mean linear movement between waypoints in Figure 8 is 19 centimeters, meaning that the average prediction error of the system was roughly 13% in terms of total distance traveled by the end-effector. This low error indicates that the algorithms can compute accurate predictions based

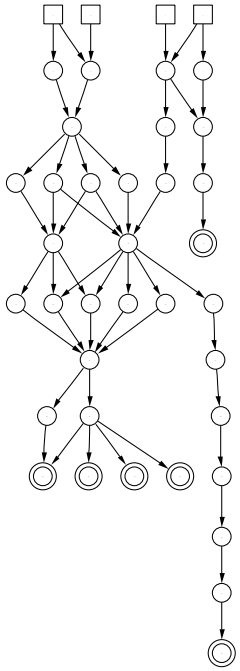

Figure 10. HMM constructed from small δ from tasks with different lengths.

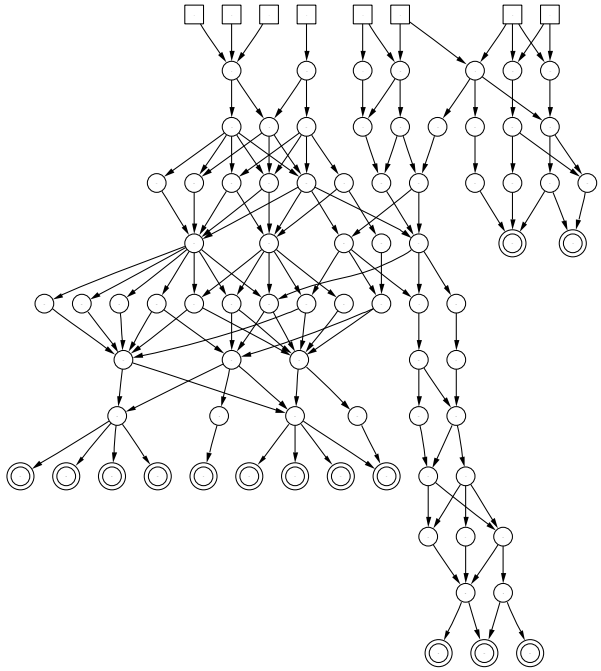

Figure 11. HMM constructed from large δ from tasks with different lengths.

Small δ	**mean**	**std dev**
Prediction Error	0.04260 meters	0.01580 meters
Error Percent	22.42%	8.316%
States	52.53	2.872
Transitions	89.46	8.646
Large δ	**mean**	**std dev**
Prediction Error	0.02587 meters	0.01291 meters
Error Percent	13.61%	6.789%
States	185.7	2.740
Transitions	284.5	4.325

Table 1. Leave-one-out estimates for the prediction performance of HMMs with variations on δ.

Prediction Criterion	**mean (s)**	**std dev (s)**	**change**
None	292.2	78.61	N/A
$\phi_n > 0.8$ $h = 3$	193.2	34.02	-33.88%
$\phi_n > 0.5$ $h = 2$	178.0	33.39	-39.08%

Table 2. Programming-time required to complete the tasks in Figure 8 with no prediction, high-confidence prediction, and low-confidence prediction respectively.

on fairly sparse data.

While knowing that the system computes accurate predictions is comforting, it is mainly of academic interest. A user of the system will be most interested in the reduction of the time required to create robot programs. In this scenario, the user is asked to complete one of the tasks from Figure 8, using an estimated repertoire based on the *a priori* programs. The user can allow the controller to move the end-effector if the PRP system has a valid prediction (*i.e.* confidence from Equation 7 exceeds a threshold). It is not uncommon for the user to decide that the prediction must be refined with the joystick, and this fine-tuning time is included as well. If an obstacle is in the path to the prediction, the user can stop the PRP system from moving the robot by releasing the dead-man switch on the teach pendant. The results are summarized in the Table 2. The first row of Table 2 is the time required by a user to program the tasks in Figure 8 with no predictions from the PRP system. The second row shows the programming time when the PRP system suggests only high-confidence predictions ($\phi_n > 0.8$) based on a relatively long horizon of recent observations ($h = 3$). The third row shows the programming time when the PRP system is allowed to suggest "sloppy" predictions, those of potentially low confidence ($\phi_n > 0.5$) based on a relatively short horizon ($h = 2$). Not surprisingly, allowing the PRP system to suggest low-confidence predictions *generally* results in greater prediction error. Though not statistically sig-

nificant, it is worth noting that allowing low-confidence predictions rather than only high-confidence predictions tended to reduce programming time. This suggests that users still find these less accurate predictions helpful. [2] Though not represented in Table 2, it seems that the primary benefit of predicting waypoints comes from automatically setting the orientation of the end-effector, more so than for the Cartesian location. This is probably due to the tendency of the inverse-kinematics routines found in many controllers to "tie the robot in knots". Under normal operation, the user must occasionally move each joint individually to prevent a physical joint-stop from being reached. This did not happen when our PRP system was computing waypoints for the user.

The covariance matrix, $\mathbf{\Sigma}$, required by Equation 1 and Equation 6 to model Gaussian user errors, was computed by asking users to move to known locations and orientations in the workspace and then computing the second moment of the residual error. As we will discuss later, the covariance matrix is assumed to be block diagonal, with one block corresponding to Cartesian errors and the other block corresponding to errors in orientation.

5.2 Representing Waypoints

A little more detail is needed on the representation of waypoints. As mentioned earlier, a waypoint consists of two components: location and orientation. Locations are defined with respect to some reference frame and, in the robot-programming domain, are invariably represented using Cartesian coordinates. Orientations define the rotation between two frames (*i.e.* an "input" and "output" frame) and, in the robot-programming domain, use a variety of representations.

5.2.1 Homogeneous Versus Non-Homogeneous

While in the previous section, we constrained the discussion to only homogeneous transforms, this is merely a mathematical "nicety", and there is no hard requirement that the location and orientation information of a waypoint be defined with respect to the same frame. Earlier, we mentioned that representing robot programs by their SRHCT yield translation- and rotation-independent predictions. But, as usual, there is no free lunch. If left in "open loop", basing predictions on SRHCT results in an accumulating error, similar to dead-reckoning errors in a mobile robot (Thrun, 1998). Of particular problem is how small errors in rotation compound themselves. In Figure 12, we corrupt a square by over-rotating each leg by 9°. The head of the fourth leg should be at the origin but due to the compounding errors, it is 47.58% of a leg-length away from the origin. But the compounding of errors in Figure 12 is certainly a worst-case scenario because, in the PRP domain, a human is controlling the robot and "closes the loop" to nullify much of the residual prediction error. In the three-dimensional case when trying to "eyeball" a robot end-effector, humans often make errors *much* larger than 9° and predictions using the SRHCT will suffer severely. While the compounding of errors is not a pressing concern, predictions should be as accurate as possible. To this end, we can employ non-homogeneous transforms to minimize the impact of the poor ability of humans to judge orientation errors. An obvious solution that works well in practice is to remove orientation estimates from the computation of the predicted location. This is accomplished by specifying the predicted location in a *global frame* and use orientation only to predict the change in orientation of the end effector. Let the predicted change in location be ${}^{G}_{n-1}\widehat{\boldsymbol{p}}_n$ and predicted change in orientation be ${}^{n-1}\widehat{\boldsymbol{R}}_n$. Let the current location and orientation of the end-effector be ${}^{G}_{G}\boldsymbol{p}_{n-1}$ and ${}^{G}\boldsymbol{R}_{n-1}$ respectively. We can compute the predicted location and orientation of the predicted waypoint as

$$ {}^{G}\widehat{\mathbf{w}}_n = \begin{cases} {}^{G}_{G}\widehat{\boldsymbol{p}}_n &= {}^{G}_{G}\boldsymbol{p}_{n-1} + {}^{G}_{n-1}\widehat{\boldsymbol{p}}_n \\ {}^{G}\widehat{\boldsymbol{R}}_n &= {}^{G}\boldsymbol{R}_{n-1}\ {}^{n-1}\widehat{\boldsymbol{R}}_n \end{cases} . \qquad (9) $$

Once again, there is no free lunch: It is straightforward to show that Equation 9 will yield translation-independent predictions but *not* rotation-independent predictions.

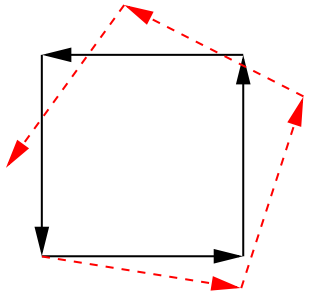

Figure 12. An error of 9° per rotation results in a compounded error of almost 50% of the leg length.

[2] Not recorded in Table 2 were the number of collisions with obstacles. Not surprisingly, there were a few collisions when the PRP system suggested low-confidence predictions. However, collisions are not uncommon when humans program automation tasks. For example, arc welding often requires submillimeter separation from the workpiece. Consequently, industrial manipulators are designed to handle the eventual collision without inflicting damage.

5.2.2 Representation of Orientation

The second important aspect of prediction is how to represent the three-dimensional orientation information needed for prediction. Rotation matrices are a poor choice, since the predicted rotation matrix would be based on an expectation (Equation 6). It is extremely unlikely that a weighted sum of orthonormal matrices willyield an orthonormal matrix. In this case, the predicted orientation will almost certainly violate one of the properties of a rotation matrix. Other common representations for three-dimensional orientation involve Euler angles. However, similar orientations may have extremely different Euler-angle representations, *e.g.* the *Z-Y-X* Euler angles $\{\frac{\pi}{2}, \frac{\pi}{2}, 0\}$ and $\{0, \frac{\pi}{2}, -\frac{\pi}{2}\}$ are equivalent. These discontinuities make Euler angles somewhat unreliable. The solution in two dimensions, of using sines and cosines to ensure angle continuity, becomes a rotation matrix in the three-dimensional case. For these reasons, we use a unit quaternion to represent orientation information (Craig, 1989). While quaternions also have discontinuous representations, they tend to be less problematic in practice. A quaternion vector must be of unit length, and renormalizing the quaternion after taking the expectation in Equation 6 yields no loss of information.

5.3 Derivation for Scale Invariance

If the PRP system has assimilated a robot program moving along a rectangle, it will be of little or no help in predicting the waypoints of different-sized rectangles, even rectangles with the same aspect ratio. The ability to recognize *and* predict scaled versions of previously assimilated tasks is called *scaled invariance*. As mentioned earlier, the PRP system is able to recognize patterns independent of their pose and location with respect to some global frame by using rotation- and translation-independent features, the SRHCT. While the same approach would work for *recognizing* scaled tasks, no feature transform appears feasible for *predicting* the waypoints of scaled tasks. For instance, we could reduce a robot program to a sequence of rays (in the mathematical sense of "ray"). To recognize a task, based on previously assimilated tasks, we could compute the likelihood of the various rays. However, these "half-infinite lines" are of little help when trying to predict the position of the next waypoint they only yield the direction of the prediction. We considered a few scale-invariant transforms but all yielded equally annoying "quirks". Thus, we have turned to a more brute-force approach: iterative locally optimal maximum likelihood.

In this formulation, we want to find the value, $\psi \in (0, \infty)$, that scales the observations in the current task, $\mathbf{X}^c_{0:n} = \{\mathbf{x}^c_0, \ldots, \mathbf{x}^c_n\}$, such that the likelihood of the scaled task is maximized given the estimated repertoire of the user

$$\begin{aligned} \widehat{\psi}^* &= \arg\max_{\psi} \mathrm{p}(\mathbf{x}^c_0, \ldots, \mathbf{x}^c_n | \psi, R) \\ &= \arg\max_{\psi} \sum_{q_j \in \mathcal{Q}} \mathrm{p}(c_n = q_j, \mathbf{X}^c_{0:n} | \psi, R) \\ &\doteq \arg\max_{\psi} \sum_{q_j \in \mathcal{Q}} \alpha^n_{j,\psi}. \end{aligned} \tag{10}$$

While the transition pmf is observation independent, the observation pdf will be a function of the scale factor. We rewrite Equation 4 as

$$\alpha^n_{j,\psi} = \begin{cases} b_{j,\psi}(\mathbf{x}_n) \sum_{q_i \in \mathcal{Q}} a_{j|i} \alpha^{n-1}_{i,\psi}, & n > 0 \\ b_{j,\psi}(\mathbf{x}_0)\pi_j, & n = 0 \end{cases}.$$

In the PRP domain, we do not want to multiply each element of an observation vector, $\mathbf{x}_n$, by the scale factor, ψ. Typically, we would like to scale only the Cartesian coordinates of the waypoint and leave the orientation components alone. Since the observation pdf is a Gaussian distribution, this implies that the covariance matrix, $\mathbf{\Sigma}$, is symmetric block-diagonal, with one block corresponding to the Cartesian coordinates and the other block corresponding to the orientation information. In real-world terms this yields the mild assumption that a user makes Cartesian errors independent of orientation errors. With this in mind, we define the "scale matrix" such that

$$\begin{aligned} \mathbf{\Psi}_{i,j} &= \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \text{ and not scaled} \\ \psi, & \text{if } i = j \text{ and scaled} \end{cases} \\ \frac{\partial}{\partial \psi} \mathbf{\Psi}_{i,j} &= \begin{cases} 0, & \text{if } i \neq j \\ 0, & \text{if } i = j \text{ and not scaled} \\ 1, & \text{if } i = j \text{ and scaled} \end{cases} \\ &\doteq \mathbf{\Psi}'_{i,j}. \end{aligned} \tag{11}$$

Then a scaled observation is $\mathbf{\Psi}\mathbf{x}_n$, which is a column vector with the Cartesian elements scaled by ψ and the orientation elements unscaled.

$$\alpha^n_{j,\psi} = \begin{cases} b_j(\mathbf{\Psi}\mathbf{x}_n) \sum_{q_i \in \mathcal{Q}} a_{j|i} \alpha^{n-1}_{i,\psi}, & n > 0 \\ b_j(\mathbf{\Psi}\mathbf{x}_0)\pi_j, & n = 0 \end{cases}. \tag{12}$$

Because the observation pdf is Gaussian and $\mathbf{\Psi}$ is a

diagonal (symmetric) matrix

$$\begin{aligned}
\frac{\partial}{\partial\psi}b_{j,\psi}(\mathbf{x}_n) &= \frac{\partial}{\partial\psi}\mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\\
&= \frac{\partial}{\partial\psi}\frac{\exp\left\{-\frac{1}{2}(\boldsymbol{\Psi}\mathbf{x}_n-\boldsymbol{\mu}_j)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Psi}\mathbf{x}_n-\boldsymbol{\mu}_j)\right\}}{\sqrt{(2\pi)^g|\boldsymbol{\Sigma}|}}\\
&= \mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\,\frac{\partial}{\partial\psi}(\boldsymbol{\Psi}\mathbf{x}_n-\boldsymbol{\mu}_j)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Psi}\mathbf{x}_n-\boldsymbol{\mu}_j)\\
&= \mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\left\{(\boldsymbol{\Psi}'\mathbf{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\Psi}\mathbf{x}_n)\right\}\\
&\doteq \left\{(\boldsymbol{\Psi}'\mathbf{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\Psi}\mathbf{x}_n)\right\}b_{j,\psi}(\mathbf{x}_n).
\end{aligned}$$

We will also need the following notation

$$\begin{aligned}
&\mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&\quad= \sum_{q_i\in\mathcal{Q}}\mathrm{p}\big(c_n{=}q_j,c_{n-1}{=}q_i,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&\quad= \sum_{q_i\in\mathcal{Q}}\mathrm{P}(c_n{=}q_j|c_{n-1}{=}q_i,R)\,\mathrm{p}\big(c_{n-1}{=}q_i,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&\quad\doteq \sum_{q_i\in\mathcal{Q}}a_{j|i}\alpha^{n-1}_{i,\psi},
\end{aligned}$$

and its corresponding partial derivative

$$\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big) \;\doteq\; \sum_{q_i\in\mathcal{Q}}a_{j|i}\frac{\partial}{\partial\psi}\alpha^{n-1}_{i,\psi}.$$

The partial derivative of Equation 12 with respect to the scale factor is

$$\begin{aligned}
\frac{\partial}{\partial\psi}\alpha^n_{j,\psi} &= \frac{\partial}{\partial\psi}\mathrm{p}(c_n{=}q_j,\mathbf{X}^c_{0:n}|\psi,R)\\
&= \frac{\partial}{\partial\psi}\mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\,\mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&= \mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\,\frac{\partial}{\partial\psi}\mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)+\\
&\quad\ \mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\,\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&= \mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\,\frac{\partial}{\partial\psi}\mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)+\\
&\quad\ \mathrm{p}(\mathbf{x}_n|c_n{=}q_j,\psi,R)\,\frac{\partial}{\partial\psi}\mathrm{p}\big(c_n{=}q_j,\mathbf{X}^c_{0:n-1}|\psi,R\big)\\
&\doteq \Big(\sum_{q_i\in\mathcal{Q}}a_{j|i}\alpha^{n-1}_{i,\psi}\Big)\Big(\frac{\partial}{\partial\psi}b_{j,\psi}(\mathbf{x}_n)\Big)+\\
&\quad\ \Big(b_{j,\psi}(\mathbf{x}_n)\Big)\Big(\sum_{q_i\in\mathcal{Q}}a_{j|i}\frac{\partial}{\partial\psi}\alpha^{n-1}_{i,\psi}\Big)\\
&= \left\{(\boldsymbol{\Psi}'\mathbf{x}_n)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j-\boldsymbol{\Psi}\mathbf{x}_n)\right\}\alpha^n_{j,\psi}+\\
&\quad\ b_{j,\psi}(\mathbf{x}_n)\sum_{q_i\in\mathcal{Q}}a_{j|i}\frac{\partial}{\partial\psi}\alpha^{n-1}_{i,\psi}, \qquad (13)
\end{aligned}$$

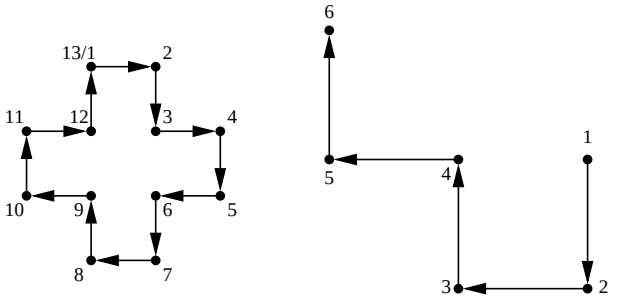

Figure 13. The waypoints (1–6) from the pattern on the right form a scaled subtask ($\psi = 2$) of waypoints (6-11) from the pattern on the left.

since Equation 13 is recursive, the appropriate initial conditions at the beginning of time are needed (the partial derivative of Equation 12). We cannot solve for the optimal ML estimator of the scale factor explicitly since Equation 13 is a sum of transcendental terms. We can only hope to find locally optimal ML solutions by iterative line-search techniques, such as cubic interpolation, gradient ascent, etc. (Bertsekas, 1995).

Once we find the (locally optimal) ML estimator of the scale, $\widehat{\psi}^*$, we compute predictions based on the scaled task, by substituting Equation 12 into Equation 6. We can also incorporate the notion of a horizon, as in Section 5, to identify scaled subtasks contained in other tasks (*cf.* Figure 13). The scaled task computed by premultiplying each observation by the ML scale matrix, $\boldsymbol{\Psi}$, from Equation 11

$$\mathbf{X}^{i,\widehat{\psi}^*} \;=\; \{\boldsymbol{\Psi}\mathbf{x}^i_0,\boldsymbol{\Psi}\mathbf{x}^i_1,\ldots,\boldsymbol{\Psi}\mathbf{x}^i_{N_i}\}. \qquad (14)$$

However, even if we are able to identify scaled (sub)tasks based on having a model, we must still incorporate scaled tasks into a model. When we execute the `Learn-HMM` algorithm, this leads to a "chicken and egg" problem: without a model, we cannot determine the ML scale of a task; without any tasks, we cannot create a model. In Figure 14, we give an algorithm that uses the tasks already assimilated to estimate the ML scale. We then scale the task according to Equation 14 and assimilate this scaled task into the DAG. However, the estimate is unreliable if a scaled version of the task has never been assimilated before. In this case, we assimilate the task unscaled. The bracketing of the scale can arise from the physical nature of manipulator robots. All robots have finite accuracy and reachability, and these numbers can serve as a bracket on the ML scale, *e.g.* $\widehat{\psi}^* \in [0.01, 100]$.

Algorithm `Learn-Scaled-HMM`
$\epsilon \in [0, \infty)$ is a similarity threshold.
$\mathbf{X} = \{\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^M\}$ is the multiset of tasks.
$\psi_{\min}$ is the minimum bracket for the scale.
$\psi_{\max} > \psi_{\min}$ is the maximum bracket for the scale.

```
1: V := ∅, E := ∅
2: G := (V, E)
3: for all X^i ∈ {X^0, X^1, ..., X^M}
4:    ψ̂* := compute-scale (G, X^i, ψ_min, ψ_max)
5:    X^{i,ψ̂*} from Equation 14
6:    G := assimilate-task (G, X^{i,ψ̂*}, ε)
7: end for all
8: R := convert-DAG-to-HMM (G)
```

Figure 14. The algorithm for assimilating scaled tasks into an HMM.

Function `compute-scale`
$G = (V, E)$ is the leveled DAG.
$\mathbf{X}^i = \{\mathbf{x}^i_0, \mathbf{x}^i_1, \ldots, \mathbf{x}^i_{N_i}\}$ is the task to assimilate.
$\psi_{\min}$ is the minimum bracket for the scale.
$\psi_{\max} > \psi_{\min}$ is the maximum bracket for the scale.

```
1:  N := maximum depth of G
2:  if |X^i| ≤ N then
3:     R := convert-DAG-to-HMM (G)
4:     ψ := arg max_ψ p(X^i | ψ, R)
5:     if ψ ∈ [ψ_min, ψ_max] then
6:        ψ̂* := ψ
7:     end if
8:     else then
9:        /* scale is outside bracket: assume 1.0 */
10:       ψ̂* := 1
11:    end else
12: end if
13: else then
14:    /* X^i too long for G */
15:    ψ̂* := 1
16: end else
17: return ψ̂*
```

Figure 15. Computing the ML scale for a task.

6. Related Work

The problem of stochastic-source prediction is generally decomposed into the two steps mentioned in this paper: creating a model of the source and using the model to predict the future. Optimal prediction, given a model, is a fairly mature subject and can be found in many references such Duda et al. (2001). On the other hand, the modeling of stochastic sources is still an active area of research. Typically, the source is assumed to have the Markov property (sometimes extended to include kth-order Markov sources though these are equivalent to first-order Markov sources). Information-theoretic approaches give bounds on the "finite-state predictability" of positive-entropy Markov sources (Feder et al., 1992).

In the seminal work on HMM applications, Rabiner (1989) outlines the "Three Basic Problems for HMMs" and notes that the "third, and by far the most difficult, problem of HMMs is to determine a method to adjust the model parameters (A, B, π) to maximize the probability of the observation sequence given the model." However, we consider the *much* more difficult problem of discovering the structure of the model, in addition to determining the optimal parameters for it. Rudich (1985) showed that the structure of discrete-symbol Markov chains can be determined in the limit from its output. However, there are results that suggest Probabilistic Finite Automata (PFAs) and HMMs are not optimally trainable globally in time polynomial in the alphabet size, unless random polynomial time is equivalent to non-deterministic polynomial time ($RP = NP$) (Abe & Warmuth, 1992).

In lieu of these disappointing revelations, researchers focus their attention on special subclasses of HMMs and PFAs to deliver more optimistic results. Ron et al. (1998) derive an algorithm that constructs correct, in the PAC-learning sense, *leveled* Acyclic Probabilistic Finite Automata (APFAs) in time polynomial to the PAC-learning parameters and requires a polynomial training set, provided the states of the target APFA are *distinguishable*. Carrasco and Oncina (1999) outline an algorithm that approximates PFA distributions in the limit. These algorithms only need an upper bound on the number of states in the model and have low training-data requirements. However, the running-time and training-data results are polynomial in the *alphabet size*. It is not clear that these approaches would scale well to the high-dimensional continuous spaces needed for predictive robot programming using vector-quantization techniques.

A long-standing problem of the forward-backward algorithm is its tendency to leave many superfluous parameters in the model and reliance on large corpora of data. There are "standard tricks" to reduce the amount of training data required by the forward-backward algorithm, such as left-right models and assuming diagonal covariance matrices (for CDHMMs).

But these modifications do not perform well on the small training sets allowed by PRP. Brand (1999) derives an entropy-based prior over HMM parameters and develops a MAP extinction scheme to reduce the number of parameters in the model, though large corpora of data are still very much needed. Other HMM-training algorithms have been developed recently (Singh et al., 2001) that use large cross-validation sets to determine the locally optimal topology.

In the optimal-control domain, there has been some research on the partitioning of state spaces of Markov Decision Processes (MDPs). It has been shown that the time complexity of partitioning an MDP is at least as difficult as computing optimal policies on the original MDP (Ren & Krogh, 2002). These very informative results rely on the full observability of the Markov chain and MDP cost function. In our work, the bulk of the computation is spent on inducing the structure of the Markov chain from the data and we are interested in finding simple models that explain the data, not simpler models with (nearly) isometric properties.

In the HCI domain, prediction and synthesis based on user observations goes by the name of Learning By Observation, Programming by Demonstration, Teaching by Example, or some permutation thereof. In (Yang et al., 1994), researchers used standard HMM-training techniques to learn telerobotic manipulation skills. These skills were open loop, in that there was no force or visual feedback to the user demonstrating the task, and were based on Finite-State Automata (FSAs) with each node executing a memoryless, nonlinear control law and transitions determined by HMM training on user examples. In (Ikeuchi et al., 1993), researchers created a system that generates robot programs for assembly tasks based on observations of humans performing the same task. Pomerleau (1994) constructed a system that allowed users to generate an essentially limitless supply of training examples steering a car. From these examples, the computer system learned to imitate the car-steering skill of the user, using a camera as input. In this work, the skill was a memoryless, nonlinear control law. Yang et al. (1994) used HMM-training techniques to decompose human telerobotic manipulation tasks into a sequence of *primitives*. A primitive was an open-loop, memoryless, nonlinear control law. Temporal sequences of primitives comprised *skills*, which were FSAs with state transitions determined by the HMM-training algorithm. Friedrich et al. (1996) used Time-Delay Neural Networks (TDNNs) to decompose human actions into a sequence of predefined primitives, storing the decomposition in a symbolic fashion. A search, using the pre- and post-condition STRIPS model (Fikes & Nilsson, 1971), is then executed to determine a sequence of primitives that describes the sequence observed from the user. The segmentation derived from the TDNNs serves as an initial bias on the search to create the automatically generated program. However, all programs run without any sensor input and are unsuitable for most realistic operating conditions. Kang and Ikeuchi (1995) used Dynamic Time Warping (DTW) to decompose observations of humans performing assembly tasks into symbolic, predefined primitives. Matarić (1999) used neuroscience and psychophysical models to select a "basis set" of primitives. Observations from users are segmented to determine the primitives to execute in a temporally sequenced or additive fashion. User observations have also been incorporated to "prime" iterative optimal-control techniques in robot-learning tasks (Schaal, 1997).

7. Conclusions and Future Work

In this paper we have presented algorithms for the unsupervised model-based prediction of user actions. By modeling user actions as noise-corrupted real-valued vectors, we cast the LBO problem as one of SSP. Essentially, our SSP algorithms operate in two phases. The first phase estimates a repertoire of the user and due to our formulation, this naturally results in the use of CDHMMs. We derived results showing that the learning algorithm produces ϵ-compact graphical models in second-order polynomial computational complexity. The second phase of the SSP algorithms use the CDHMMs to predict future values of user actions.

In order to decrease online robot-programming time, we have created a novel application: predictive robot programming. Our algorithms were able to predict very accurately the next waypoint in the robot program using a very small training set. Furthermore, our system has been shown in laboratory experiments to reduce robot-programming time significantly.

In the future, we would like to complete the theoretical foundations of the algorithms by bounding the prediction error of the system. It appears as though this result will be a PAC bound on the error, which should naturally rise to a probabilistic sample-size requirement. We will also develop modified versions of the algorithms to allow for task correction and recognition. Task correction and recognition involve linking multiple CDHMMs in a probabilistic "grammar". With this grammar, it should be possible to identify when a user diverges from a pre-determined set of rules governing standard behavior or which specific task the user is performing, similar to phonemes in speech recognition.

However, these modifications will require a (partial) labeling of the data.

Acknowledgments

We would like to thank Mike Seltzer and Chris Paredis for asking tough questions and giving enlightening suggestions. Martin Strand and John Dolan also provided helpful comments for the application of the algorithms to predictive robot programming. We used GraphViz (`dotty`) from AT&T Research to display HMM topologies. We would like to thank the Intel Corporation for providing the computing hardware. This work was sponsored by ABB Corporate Research.

References

Abe, N., & Warmuth, M. K. (1992). On the computational complexity of approximating probability distributions by probabilistic automata. *Machine Learning*, *9*.

Bertsekas, D. P. (1995). *Nonlinear programming*. Athena Scientific.

Brand, M. (1999). An entropic estimator for structure discovery. *Advances in Neural Information Processing Systems* (pp. 723–729).

Carrasco, R. C., & Oncina, J. (1999). Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, *33*, 1–19.

Craig, J. J. (1989). *Introduction to robotics: Mechanics and control*. Addison Wesley. Second edition.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. Wiley-Interscience. Second edition.

Feder, M., Merhav, N., & Gutman, M. (1992). Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, *38*, 1258–1270.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Friedrich, H., Münch, S., Dillmann, R., Bocionek, S., & Sassin, M. (1996). Robot programming by demonstration (RPD): Supporting the induction by human interaction. *Machine Learning*, *23*, 163–189.

Ikeuchi, K., Kawade, M., & Suehiro, T. (1993). Towards assembly plan from observation: Task recognition with planar, curved and mechanical contacts. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2294–2301).

Kang, S. B., & Ikeuchi, K. (1995). A robot system that observes and replicates grasping tasks. *IEEE Fifth International Conference on Computer Vision* (pp. 1093–1099).

Matarić, M. J. (1999). Visuo-motor primitives as a basis for learning by imitation. *Imitation in Animals and Artifacts*. Cambridge, MA: MIT Press.

Pomerleau, D. (1994). *Neural network perception for mobile robot guidance*. Doctoral dissertation, Carnegie Mellon University.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257–286.

Ren, Z., & Krogh, B. H. (2002). State aggregation in Markov decision processes. To appear in *IEEE Conference on Decision and Control*.

Ron, D., Singer, Y., & Tishby, N. (1998). On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, *56*.

Rudich, S. (1985). Inferring the structure of a Markov chain from its output. *IEEE Symposium on Foundations of Computer Science* (pp. 321–326).

Schaal, S. (1997). Learning from demonstration. *Advances in Neural Information Processing Systems* (pp. 1040–1046).

Singh, R., Raj, B., & Stern, R. M. (2001). Automatic generation of sub-word units for speech recognition systems. Submitted to *IEEE Transactions on Speech and Audio Processing*.

Thrun, S. B. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, *99*, 21–71.

Yang, J., Xu, Y., & Chen, C. S. (1994). Hidden Markov model approach to skill learning and its application to telerobotics. *IEEE Transactions on Robotics and Automation*, *10*, 621–631.