Predictive Robot Programming: Theoretical and Experimental Analysis

Kevin R. Dixon[1] krd@cs.cmu.edu
John M. Dolan[2] jmd@cs.cmu.edu
Pradeep K. Khosla[1,2] pkk@ece.cmu.edu

[1]Dept of Electrical & Computer Engineering
[2]The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract

As the capabilities of manipulator robots increase, they are performing more complex tasks. The cumbersome nature of conventional programming methods limits robotic automation due to the lengthy programming time. We present a novel method for reducing the time needed to program a manipulator robot: Predictive Robot Programming (PRP). The PRP system constructs a statistical model of the user by incorporating information from previously completed tasks. Using this model, the PRP system computes predictions about where the user will move the robot. The user can reduce programming time by allowing the PRP system to complete the task automatically. In this paper, we derive a learning algorithm that estimates the structure of continuous-density hidden Markov models from tasks the user has already completed. We analyze the performance of the PRP system on two sets of data. The first set is based on data from complex, real-world robotic tasks. We show that the PRP system is able to compute predictions for about 25% of the waypoints with a median prediction error less than 0.5% of the distance traveled during prediction. We also present laboratory experiments showing that the PRP system results in a significant reduction in programming time, with users completing simple robot-programming tasks over 30% faster when using the PRP system to compute predictions of future positions.

1 Introduction

Programming a manipulator robot is an arduous task. A robot program typically consists of three main components: a sequence of positions through which the robot must travel, conditional branching statements, and process-specific instructions. Of these components, robot programmers usually spend the majority of their time defining the sequence of positions, called *waypoints*. While critical to the success of all robot programs, specifying waypoints is currently an overly complex and time-consuming process. Consequently, one of the main inhibitors of robotic automation is the time needed to program the manipulator.

Robot programming has evolved into two mutually exclusive methods, offline and online programming, each having its advantages and disadvantages. In offline programming, users move a simulated robot to each waypoint using a simulated model of the workspace. Offline-programming packages allow users to design robot programs in simulation without bringing down production and can optimize programs according to almost any imaginable criterion. Typical optimizations involve production speed, material usage, or power consumption. To achieve the high accuracy required in many applications, the physical workspace must be well calibrated with respect to the simulated environment. Otherwise, extensive online fine-tuning will be needed, which detracts from the largest benefit of offline programming: lack of production downtime. Offline systems generally require that programs be written in a sophisticated procedural-programming language. As such, users of these systems must be experts at the industrial process as well as computer programming. Indeed, expertise in either of these fields is a job skill in its own right.

Despite the advantages of offline packages, online programming is more commonly used in practice. In online programming, an actual part is placed in the workspace exactly as it would be during production. The user creates the robot program by moving 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 robot programs without editing this code, and it is typically viewed

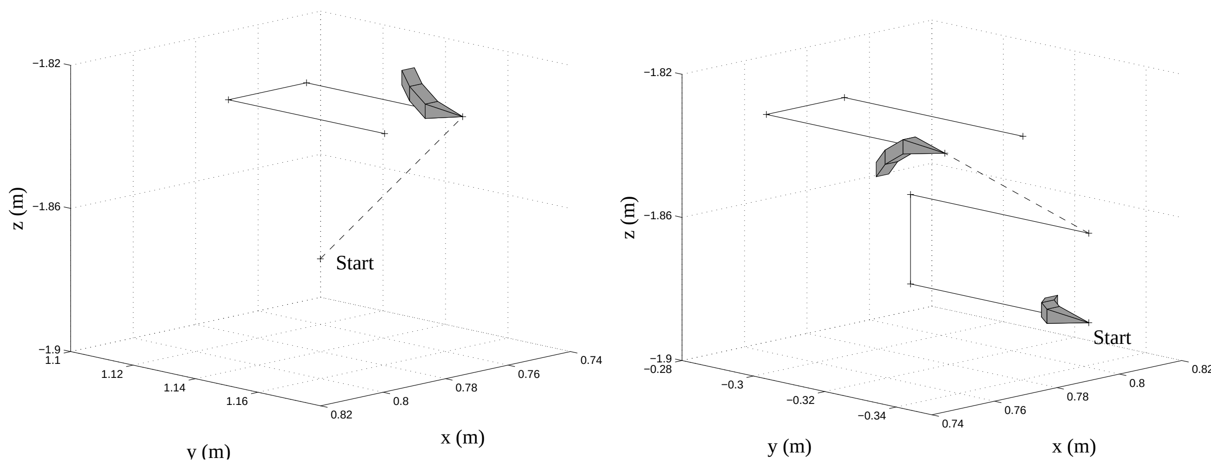


Figure 1: Waypoints from two subroutines in the same robot program. While different at the subroutine level, there are repeated subtasks occurring in translated and rotated form, such as the “U-shaped” pattern. The relative movement of the robot with respect to the end-effector during these subtasks is the same.

as less intimidating and more intuitive than offline programming. One potential disadvantage of online systems is that production and programming cannot occur in parallel; production must be halted during reprogramming. If reprogramming cannot be completed during normal downtime, such as weekends, then cost will be incurred in the form of lost production. Therefore, the set of tasks viable for online programming is constrained by programming time. A reduction in this time would allow its use in areas previously off limits.

In this paper, we present a novel Predictive Robot-Programming (PRP) system that allows users to leverage their previous work to decrease future programming time. Specifically, this system assists users by predicting where they may move the end-effector and automatically positions the robot at the estimated waypoint. The PRP idea of automatically completing a task based on a few observations is conceptually similar to word-completion routines in word-processing programs and text-messaging in mobile phones. In that application, the user is presented with a word completion based on a few keystrokes, in the hope of reducing the time needed to type a message. The domain of word completion is well defined: a dictionary. In PRP, there is an uncountable number of tasks that a manipulator robot may perform and there exist no corpora of example programs. Further complicating any prediction scheme is the inherent imprecision and poor repeatability of humans and the tendency of users to perform the same task in different manners. Consider the case of welding the joints of a rectangle. If the goal is to weld the object together, then the corners may be welded in any order. Such ambiguity complicates any prediction scheme, and our PRP system addresses this uncertainty both during modeling and prediction.

Manipulator robots perform a wide variety of industrial applications, and as the capabilities of robots increase, they are performing more sophisticated tasks. Simple tasks can take days to program, while more complex tasks can take weeks or months. Despite their complexity, most tasks can usually be decomposed into simpler subtasks. These subtasks may be repeated throughout the program directly or in some modified form. Most robot programmers either do not recognize the similarity or create these subtasks from scratch each time due to the cumbersome nature of current programming environments. For example, in Figure 1 we show the waypoints from two different subroutines in an arc-welding robot program. While clearly different at the subroutine level, there are repeated subtasks, such as a “U-shaped” pattern, which is rotated differently in the two subroutines. However, the patterns have the same movements with respect to the end-effector. Such repeated subtasks tend to be specific to a particular robot program. That is, the similarity arises from the physical workpiece and the industrial process at hand. If a different workpiece is supplied or a different industrial process employed, then the pattern of similarity would change. While the user is creating waypoints, the PRP system must be able to identify similarities, if any, to the many previously created subtasks and then suggest future waypoints. If the user allows the PRP system to move the robot, then the end-effector is automatically positioned at the predicted waypoint. Compared to the time needed to move a robot manually, automatic positioning of a robot is essentially instantaneous, which reduces overall programming time. The prediction process can execute in a closed- or open-loop fashion. During closed-loop operation, the PRP system moves the robot to a single predicted position and the user may fine-tune the waypoint with a conventional control device. In the open-loop mode, the PRP system automatically completes the task for the user without adjustment to the waypoints. Both prediction modes have their advantages and disadvantages. Closed-loop prediction tends to produce more accurate waypoints because user feedback reduces error. Open-loop prediction is faster since the PRP system does not have to wait for user intervention.

We place our research in the context of related work in Section 2 and derive the underlying learning algorithm in Section 3. Essentially, the PRP system induces the topological structure and parameters of a Continuous-Density Hidden Markov Model (CDHMM) based on waypoints that the user has previously created. This CDHMM is then used to predict future waypoints. The experimental results are divided into the offline- and online-programming domains. For the offline-programming experiments (Section 4), we analyze the performance of the PRP system in predicting the waypoints of complex, real-world industrial tasks. Specifically, the programs consist of several thousand waypoints and were created to automate arc-welding production at various factories. Each program was designed to produce a different product, from bed frames to round tables. On each program, we are able to generate a large percentage of highly accurate predictions. For the online-programming experiments (Section 5), we show that the PRP system contributes to a significant reduction in the time needed to program simple tasks in a laboratory setting. Finally, we give conclusions and a discussion of the results in Section 6.

2 Related Work

In recent years there has been increased interest in decreasing robot-programming time, primarily by simplifying the interaction between users and robots. For the vast majority of the population, programming ability limits the set of tasks that can be automated. Most potential robot users have neither the experience nor the inclination to program robots to perform many of their tasks. Several researchers have developed multimodal interfaces to simplify human-robot interaction. Perzanowski et al. (2001) use a speech, gesture, and graphical interface to study natural interactions between humans and robots. Iba et al. (2002) have developed a system that incorporates speech and gesture, instead of a keyboard and joystick, to program a vacuum-cleaning mobile robot. These types of systems target users who are experts at a particular task but may have limited programming ability. Several researchers have also developed systems that observe users performing tasks and synthesize the information so that a robot can perform the tasks. This paradigm goes by the name of Learning By Observation (LBO), Programming by Demonstration, Teaching from Example, or some permutation thereof.

Both PRP and LBO operate in the same two-phase process: learning and execution. The learning phase of PRP and LBO involves the construction of a model of user actions based on prior behavior. However, during the execution phase, the PRP system only predicts the *next* waypoint in the trajectory whereas LBO systems must complete the entire task. The best-known example of LBO is the neural-network system, ALVINN, that learns to drive a car by observing human drivers with a camera (Pomerleau, 1991). From these observations, the system learned to imitate the car-steering skill of the driver. However, one of the hallmarks of LBO is the extreme scarcity of data upon which to train the system. This is because LBO systems gather observations from human activity, which may be of significant duration. As such, it may require hours, days, or months to obtain a sufficient number of tasks for successful learning. To cope with the scarcity of training data, many researchers decompose an observation sequence into a discrete, symbolic representation and incorporate extensive domain knowledge of the tasks. Friedrich et al. (1996) segment a task, demonstrated by a user, into a sequence of predefined primitives using Time-Delay Neural Networks (TDNNs), storing the decomposition in a symbolic fashion. A search, using a pre- and post-condition theorem-proving model (Fikes & Nilsson, 1971), determines a sequence of primitives that describes the actions of the user. The segmentation derived from the TDNNs serves as an initial bias on the search, and the user may also modify resulting program. Similarly, Nicolescu and Matarić (2001) created a system that decomposes a demonstration into a symbolic set of robotic behaviors. Using dynamic programming and human feedback, a deterministic acyclic automaton “behavior network” is constructed from multiple examples of a task. This allowed the system to extract user intentions from a relatively small training set. Chen and Zelinsky (2003) use a symbolic configuration-space description to represent a simple assembly task. Suboptimal human demonstrations were ameliorated by “filtering” the observations and by incorporating extensive domain knowledge with heuristic methods. Several researchers have also used Hidden Markov Models (HMMs) to recognize human actions. Typically, these systems create an individual HMM for each action that the user may perform during a given task. The topologies of the constituent HMMs are determined *a priori* according to some predefined structure, such as left-right models (Hannaford & Lee, 1991). The HMMs are typically optimized using a labeled demonstration by the Baum-Welch algorithm (Rabiner, 1989). To encode higher-level knowledge, these models are sometimes connected together in a task-specific manner, creating a sort of "grammar" that constrains the set of possible actions (Hovland et al., 1996). This grammatical concept has recently been extended to recognize unknown actions (Iba et al., 2003). HMM-based systems have also been used in the telerobotic manipulation domain for analysis of force/torque actions (Hannaford & Lee, 1991), consistency analysis of human actions (Tso & Liu, 1997), learning new skills (Yang et al., 1997), and real-time operator assistance (Hundtofte et al., 2002; Li & Okamura, 2003). In addition to their applications in speech recognition (Rabiner, 1989), HMMs have also been used extensively in recognizing handwriting of various languages (Bahlman & Burkhardt, 2001; Solis et al., 2002).

As mentioned previously, we use a CDHMM to model user behavior. Other HMM-based LBO systems have relied on knowing the structure of the target task in advance. However, our approach differs from previous work in that we do not know the set tasks that the user may perform *a priori*. To cope with this issue, we have developed an algorithm that estimates the *structure*, or topology, of a CDHMM based on waypoints that the user has previously created. After estimating the structure of the model, any of the well-known fixed-topology algorithms can be applied to optimize the model parameters, such as the Baum-Welch algorithm.

In the machine-learning field there is a substantial amount of research on HMMs. There are results suggesting that optimal training of general HMMs is not possible in polynomial time (Abe & Warmuth, 1992). Consequently, some researchers have focused on special subclasses of HMMs to deliver better results. Ron et al. (1998) have derived a state-merging algorithm that constructs correct, in the Probably Approximately Correct sense, discrete-symbol Probabilistic Finite Automata (PFAs). The running time is polynomial in the size of the sample set and number of possible observations (alphabet size), provided the states of the target PFA are *distinguishable* and *acyclic*. There is a similar state-merging algorithm that estimates general PFA topologies in the limit of infinite training data (Carrasco & Oncina, 1999). The theoretical results of these works rely on enumerable alphabet sizes, whereas PRP requires multivariate real-valued vectors to describe a robot program. Several researchers have also pursued more heuristic approaches to HMM structure estimation. For example, Stolcke and Omohundro (1994) incorporated a prior topology distribution favoring simple models and performed a best-first state-merging to induce the structure of an HMM from observations. Brand (1999) used an entropy-based prior to cause parameter extinction in ergodic HMMs. But such heuristics make it difficult to provide performance guarantees. Singh et al. (2002) used early-stopping techniques to determine the phonetic units for a speech recognizer. The process of determining the atomic units of speech has much in common with representing the "primitives" needed to complete a set of tasks. However, the large cross-validation sets found in the speech-recognition domain are not available in PRP.

There has also been recent interest in estimating the structure of HMMs in the statistics and information-theory fields, where it is called order estimation (Ephraim & Merhav, 2002). Results from these fields deal with bounding the asymptotic likelihood of under- or over-estimating the number of states (Merhav et al., 1989) and there have been generalizations to handle continuous observations (Rydén, 1995). While theoretically appealing, these approaches rely on the existence of a globally optimal maximum likelihood estimation procedure for HMMs, which is known to be intractable (Abe & Warmuth, 1992). There has been more recent work on restricting the classes of HMMs in order to derive viable results (Gassiat & Boucheron, 2003). However, these results are focused on asymptotic performance and not computational complexity. Even the tractable approaches only specify termination in a finite number of steps (Ephraim & Merhav, 2002) and, consequently, these guarantees are not appropriate for real-time use in a PRP system.

We have created a CDHMM learning algorithm motivated by the characteristic difficulties of LBO. First, the algorithm must be able to operate in real time, so its computational complexity must be low and it cannot rely on asymptotic performance. Furthermore, the algorithm cannot expect large amounts of training data, such as the large corpora of data in the speech-recognition domain. Our learning algorithm is designed to identify the relatively short-sequence similarities found in PRP, such as those in Figure 1. We have therefore focused our attention on a low-complexity algorithm that produces the "right" waypoint at the "right" time, rather than asymptotic sequence guarantees such as those described by Ephraim and Merhav (2002).

3 The Learning Algorithm

3.1 Representing Waypoints

In three-dimensional space, the position of an object can be described by a location and an orientation. While the location is typically given by a rectangular $\{x, y, z\}$ description there are many orientation representations used. Three common descriptions include rotation matrices, Euler angles, and unit quaternions (Mason, 2001). While each rep-

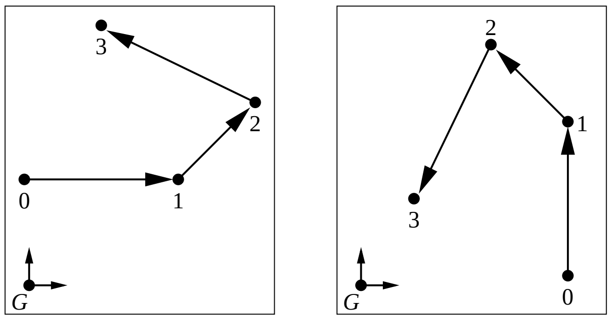
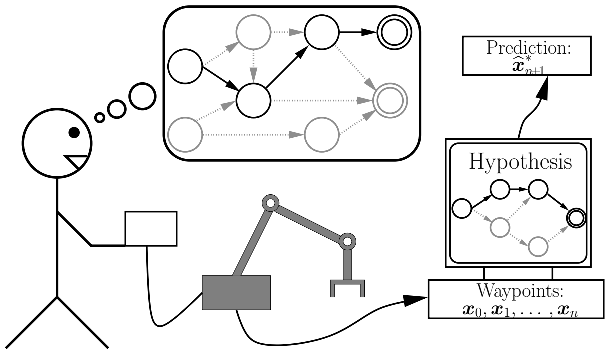


Figure 2: Though different in a global frame, G, the relative movement of these patterns is the same.



Prediction: $\widehat{\boldsymbol{x}}^*_{n+1}$
Hypothesis
Waypoints: $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

Figure 3: Conceptual modeling of the user.

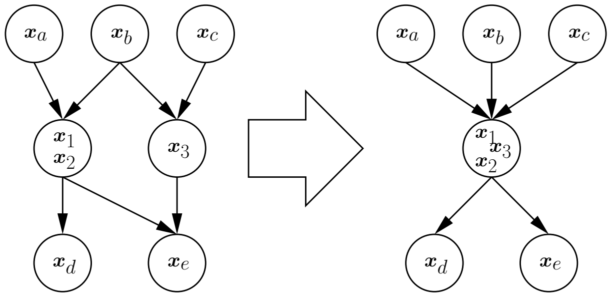


Figure 4: Hypothetical merging of two nodes in a graph.

resentation has drawbacks, unit quaternions have yielded the best performance in our experiments.

In PRP, the ability to recognize and predict patterns independent of their location and orientation in the workspace is extremely useful. Let a robot program be given by the sequence of waypoints $\boldsymbol{W} = \{{}^G\boldsymbol{w}_0, {}^G\boldsymbol{w}_1, \ldots, {}^G\boldsymbol{w}_N\}$, where the waypoint ${}^G\boldsymbol{w}_n$ maps the global frame, G, to the frame of the nth waypoint. This representation specifies waypoints in a global, or absolute, manner. Let the relative-movement representation be $\widetilde{\boldsymbol{W}} = \{{}^0\boldsymbol{w}_1, {}^1\boldsymbol{w}_2, \ldots, {}^{N-1}\boldsymbol{w}_N\}$, where ${}^{n-1}\boldsymbol{w}_n$ is the change in position between waypoint ${}^G\boldsymbol{w}_{n-1}$ and waypoint ${}^G\boldsymbol{w}_n$. It is simple to show that $\widetilde{\boldsymbol{W}}$ is independent of an initial coordinate-frame transform. In other words, using the relative-movement representation permits the recognition and prediction of patterns in a rotation- and translation-independent manner. The relative-movement description requires that the user specify the orientation of the end-effector very precisely. If the user repeats a task with a slight difference in orientation, then the Cartesian errors between the two demonstrations can quickly accumulate and become large, similar to the compounding of dead-reckoning errors in a mobile robot (Thrun, 1998). However, the ability to compute rotation- and translation-independent predictions is critical to the success of the PRP system since the similarities embedded in a robot program occur at different orientations and locations in the workspace, *cf.* Figure 1.

For notational convenience, we use a column vector to represent a waypoint, $\boldsymbol{x}_n = \text{vec}({}^n\boldsymbol{w}_{n+1})$. Describing orientation information with unit quaternions means that $\boldsymbol{x}_n$ is a (7×1) vector.

3.2 Learning Algorithm Overview

In the learning phase, the system constructs a model based on waypoints that the user has already created. We make no requirement that the sequences are the same length or perform the same physical task. Due to the poor repeatability and low precision of humans, the waypoints can be considered noise-corrupted. Furthermore, for a particular task there may be many possible ways that the user could achieve the desired goal and it is difficult to instrument fully any realistic working environment. Consequently there will always be hidden, or latent, causes for user behavior. With these assumptions, one natural model of the user is a Continuous-Density Hidden Markov Model (CDHMM).

We have created a learning algorithm that is designed to cope with the characteristic difficulties of LBO: sparse training data, real-time operation, and a wide range of target tasks. We assume that the PRP system has no *a priori* knowledge of what programs the user may create. Therefore we must induce the structure of the CDHMM, as well as parameters for the model, from observations alone (Figure 3). We consider the user as generating a each task according to a random walk through an unknown *target* CDHMM. The learning algorithm begins by assigning one waypoint to each node in a graph and encoding temporal information by edge connectivity, as in Figure 5(a). The algorithm then searches for the most *similar* nodes in the graph. If these nodes are sufficiently similar then the nodes are *merged*. That is, a new node is created containing the

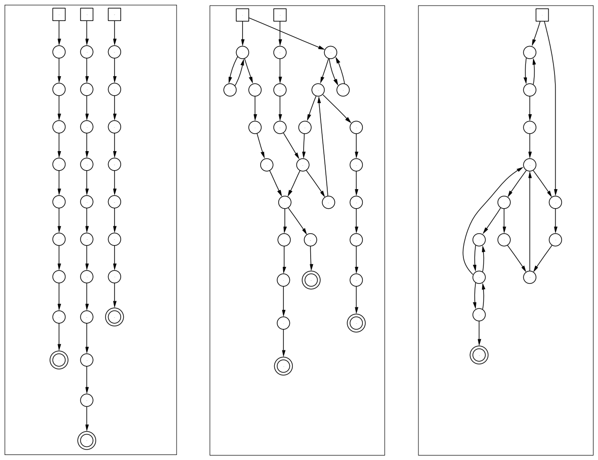

(a) (b) (c)

Figure 5: The maximal-node graph is constructed from three demonstrations of a task in Figure 5(a). Results of state merging using strict and loose similarity are shown in Figure 5(b) and Figure 5(c) respectively.

waypoints from the old nodes and the edges of the old nodes are reconnected to the new node, *cf.* Figure 4. This merging process repeats until no similar nodes remain, *cf.* Figure 5. At this point, the graph contains the *estimated* topology of the target CDHMM, *i.e.* the node connectivity, as well as the edge counts and waypoints assigned to each node. In principle, any of the well-known fixed-topology HMM estimation procedures could be used to convert the structure embodied by the graph to a CDHMM, such as the Baum-Welch algorithm (Rabiner, 1989). However, we use a simple one-shot procedure to determine the parameters of the CDHMM. For example, the edge counts can be quickly converted to transition probabilities by simple division. The observation-generation probability density functions (pdfs) can be estimated by fitting a parametric model to the waypoints assigned to each node, *e.g.* the mean and covariance of a Gaussian. We use one-shot estimation, in lieu of an iterative Expectation-Maximization (EM) approach (Bilmes, 1997), for computational expediency and because we typically lack the data required for complete EM re-estimation procedures.

The remainder of this section formalizes the learning algorithm. We begin by defining node similarity and show that the merging algorithm only produces graphs with a locally minimal number of nodes (Section 3.3). We then show that the worst-case running time of the algorithm is quadratic in the number of waypoints (Section 3.4). The topological structure in the graph is converted to a CDHMM using a simple one-shot estimation (Section 3.5). Next we derive a bound showing that as more tasks are incorporated into the CDHMM, the probability of generating the "correct" waypoint increases exponentially (Section 3.6). We then show how the CDHMM computes predictions of future waypoints and describe a parameter that the PRP system uses to indicate its prediction confidence (Section 3.8).

3.3 Learning Algorithm Derivation

We denote the multiset over the set $\boldsymbol{\mathcal{A}}$ as $\mathcal{M}(\boldsymbol{\mathcal{A}})$ and its sample mean as $\langle\boldsymbol{\mathcal{A}}\rangle$. To estimate the structure of the CDHMM, we use a type of directed multigraph called an *observation graph* and given by the 7-tuple $G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ where

- V is the set of nodes;
- V^0 is the set of initial nodes;
- E is the set of directed edges;
- $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^7$ is the set of waypoints of dimension 7;
- $\boldsymbol{\mathcal{V}} : V \to \mathcal{M}(\boldsymbol{\mathcal{X}})$ is the multiset of waypoints assigned to each node;
- $f : E \to \mathbb{Z}_{\geq 0}$ is the edge-count function;
- $g : V^0 \to \mathbb{Z}_{\geq 0}$ is the initial-count function.

The depth of an observation graph is determined by a breadth-first search from the set of initial nodes, V^0. Since the graph may be cyclic, a node may exist at multiple depths.

Central to the learning algorithm is the definition of similarity. We extend the use of the Mahalanobis distance to compute the similarity between the waypoints assigned to different nodes in the graph. Our measure (Morgan, 2000) of similarity is defined as

$$\begin{aligned} \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_k}, \boldsymbol{u}) &\triangleq \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{v_k}} \|\boldsymbol{x}-\boldsymbol{u}\|^2_{\boldsymbol{C}} \\ &\doteq \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{V}}_{v_k}} (\boldsymbol{x}-\boldsymbol{u})^{\mathsf{T}}\boldsymbol{C}(\boldsymbol{x}-\boldsymbol{u}), \end{aligned} \quad (1)$$

where the symmetric Positive Definite (PD) precision matrix $\boldsymbol{C}$ provides a notion of the *a priori* expectation of node variance. We also define $\mu_{\boldsymbol{C}}(\emptyset, \boldsymbol{u}) = 0$.

Lemma 1. *$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \boldsymbol{u})$, defined in Equation 1, is a measure on the multiset $\mathcal{M}(\boldsymbol{\mathcal{X}})$ and minimized when $\boldsymbol{u}$ is the sample mean of $\boldsymbol{\mathcal{V}}_{v_i}$.*

The straightforward proof is omitted.

Two nodes are considered similar if $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}\rangle) \leq \epsilon$, where $\epsilon \geq 0$ provides a continuous definition of similarity. Small values of ϵ imply a strict definition of similarity while large values imply a loose definition. Intuitively, if a node v_i can be

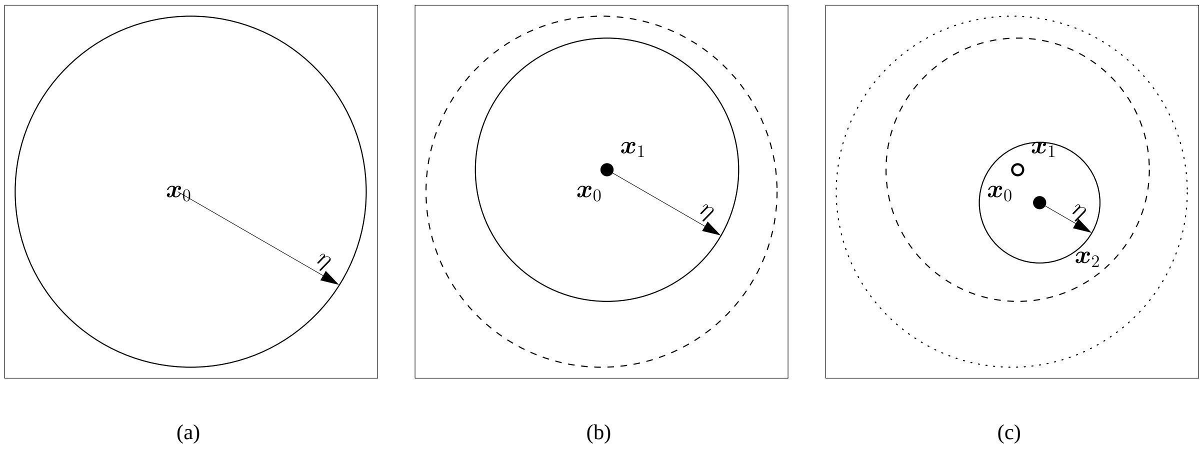


Figure 6: Illustration of adding the two-dimensional vectors $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$ to node v_i. The solid circle represents the region within which a future vector must lie to ensure that $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$. Consequently the circle has radius $\eta = \epsilon - \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle)$. The dashed circle is the image of the previous region. The centroid of the circles shift according to the sample mean, $\langle \boldsymbol{\mathcal{V}}_{v_i} \rangle$, of the waypoints already assigned to the node.

found with the capacity to add all the waypoints from node v_j then the two nodes will be merged. This process repeats until no similar nodes exist. The nodes in the maximal graph in Figure 5(a) are merged using a strict and a loose definition of similarity, in Figure 5(b) and Figure 5(c) respectively. The resulting graphs are different since the definition of similarity was different. In Section 4.2, we show how modifying the definition of similarity effects the prediction performance of the PRP system. With respect to computing the similarity between waypoint sequences, Equation 1 is *memoryless* in that the function does not consider the similarity of ancestor or descendant waypoints. Consequently, it is possible for two sequences to be similar for a single waypoint while being dissimilar for all others. While recursive similarity may be appropriate in some domains (Ron et al., 1998), memoryless similarity seems more appropriate for identifying similarities such as those shown in Figure 1. In a sense, the parameter ϵ behaves like an initial node "capacity", or a region within which future waypoints must lie. Let $\boldsymbol{x}_0$ be the first waypoint assigned to node v_i. Any future waypoints assigned to node v_i must lie inside a hyperellipse of radius ϵ whose axes are defined by the eigendecomposition of $\boldsymbol{C}$ and centered about $\boldsymbol{x}_0$. Due to the non-negative nature of Equation 1, as more waypoints are added to node v_i the capacity for the node to accept more waypoints decreases, shown graphically in Figure 6. In the following lemma we show that the acceptable region necessarily becomes smaller as waypoints are assigned to a node and that the subsequent regions are completely contained within the initial hyperellipse. This behavior is central in deriving an algorithm that produces irreducible observation graphs.

Lemma 2. *For any multiset $\boldsymbol{\mathcal{B}}$ and for all submultisets $\boldsymbol{\mathcal{A}} \subseteq \boldsymbol{\mathcal{B}}$ and for all supermultisets $\boldsymbol{\mathcal{C}} \supseteq \boldsymbol{\mathcal{B}}$,*

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{A}} \rangle) \leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{B}} \rangle) \leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{C}}, \langle \boldsymbol{\mathcal{C}} \rangle).$$

Proof.

$$\begin{aligned}
\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{A}} \rangle) &\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) + \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}} \setminus \boldsymbol{\mathcal{A}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&= \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{B}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) \\
&\leq \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) + \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{C}} \setminus \boldsymbol{\mathcal{B}}, \langle \boldsymbol{\mathcal{C}} \rangle) \\
&= \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{C}}, \langle \boldsymbol{\mathcal{C}} \rangle),
\end{aligned}$$

which completes the claim. □

For a state-merging approach, such as ours, we want to define what it means for a graph to be irreducible, or *compact*. Intuitively, a compact graph is one where all similar nodes are merged but all dissimilar nodes are left unmerged. The following definition states this in precise terms.

Definition 1. *An observation graph is ϵ-*compact *with respect to $\mu_{\boldsymbol{C}} : \mathcal{M}(\boldsymbol{\mathcal{X}}) \rightarrow [0, \infty)$, if for all nodes $v_i \neq v_j$ at the same depth*

- $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$*;*

- $\epsilon < \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$,

for some $\epsilon \geq 0$.

While the definition of ϵ-compact implies a sort of irreducible graph, it does not necessarily imply a *globally* minimal number of nodes. It is entirely possible that merging nodes in a different order would result in fewer nodes. From this perspective, ϵ-compactness implies a type of *locally* minimal number of nodes.

Algorithm `Learn-Structure`

$\boldsymbol{X} = \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$ is the multiset of waypoint sequences.

$\epsilon \geq 0$ is the similarity threshold.

```
 1: $V := \emptyset, E := \emptyset$
 2: $G_{\boldsymbol{\mathcal{X}}} := (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$
 3: for all $\boldsymbol{X}^i \in \{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^M\}$
 4:   for all $\boldsymbol{x}_n \in \boldsymbol{X}^i = \{\boldsymbol{x}_0^i, \boldsymbol{x}_1^i, \ldots, \boldsymbol{x}_{N_i}^i\}$
 5:     $\epsilon_{\min} := \min_{v_i \in V} \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\} \rangle)$
 6:     if $\epsilon_{\min} \leq \epsilon$ then
 7:       $v_{\text{new}} := \arg\min_{v_i \in V} \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\} \rangle)$
 8:       $\boldsymbol{\mathcal{V}}_{v_{\text{new}}} := \boldsymbol{\mathcal{V}}_{v_{\text{new}}} \cup \{\boldsymbol{x}_n\}$
 9:     end if
10:     else if $\epsilon_{\min} > \epsilon$ then
11:       create empty node $v_{\text{new}}$
12:       $V := V \cup \{v_{\text{new}}\}$
13:       $\boldsymbol{\mathcal{V}}_{v_{\text{new}}} := \{\boldsymbol{x}_n\}$
14:       $g_{v_{\text{new}}} := 0$
15:       if $n > 0$ then
16:         $E := E \bigcup \{e_{\text{prev}\to\text{new}}\}$
17:         $f_{e_{\text{prev}\to\text{new}}} := 0$
18:       end if
19:     end else if
20:     if $n > 0$ then
21:       $f_{e_{\text{prev}\to\text{new}}} := f_{e_{\text{prev}\to\text{new}}} + 1$
22:     end if
23:     else if $n = 0$ then
24:       $V^0 := V^0 \cup \{v_{\text{new}}\}$
25:       $g_{v_{\text{new}}} := g_{v_{\text{new}}} + 1$
26:     end else if
27:     $v_{\text{prev}} := v_{\text{new}}$
28:   end for all
29: end for all
```

Theorem 3. `Learn-Structure` *only produces ϵ-compact observation graphs.*

Proof. The algorithm satisfies the first predicate of Definition 1 ($\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \leq \epsilon$) since it will not add a waypoint, $\boldsymbol{x}_n$, to a node, v_i, unless $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \{\boldsymbol{x}_n\} \rangle) \leq \epsilon$. Consider the second predicate of Definition 1 ($\epsilon < \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$). Take any two nodes and, without loss of generality, assume node v_i was created before v_j. There must exist a waypoint $\boldsymbol{x}_{v_j} \in \boldsymbol{\mathcal{V}}_{v_j}$ and a submultiset $\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \subseteq \boldsymbol{\mathcal{V}}_{v_i}$ such that

$$\epsilon \quad < \quad \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\} \rangle).$$

Loosely speaking, the waypoint $\boldsymbol{x}_{v_j}$ could not "fit" into any existing node (`Learn-Structure` Line `10`). Otherwise, node v_j would have never been created. From Lemma 2,

$$\begin{aligned} \epsilon \quad &< \quad \mu_{\boldsymbol{C}}(\widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\}, \langle \widetilde{\boldsymbol{\mathcal{V}}_{v_i}} \cup \{\boldsymbol{x}_{v_j}\} \rangle) \\ &\Longleftrightarrow \\ \epsilon \quad &< \quad \mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle) \end{aligned}$$

and therefore the algorithm only produces ϵ-compact observation graphs. $\square$

3.4 Learning Algorithm Running Time

Lemma 4. *The sufficient statistics of $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle)$ are*

$$\xi_{v_i} = |\boldsymbol{\mathcal{V}}_{v_i}|; \qquad \kappa_{v_i} = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{x}; \quad \boldsymbol{\sigma}_{v_i} = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}} \boldsymbol{x}.$$

The update rule for computing the union of two multisets ($\boldsymbol{\mathcal{V}}_{v_k} = \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}$) is

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

Using sufficient statistics, Equation 1 can be computed as

$$\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i}, \langle \boldsymbol{\mathcal{V}}_{v_i} \rangle) \quad \equiv \quad \kappa_{v_i} - \frac{1}{\xi_{v_i}} \boldsymbol{\sigma}_{v_i}{}^{\mathsf{T}} \boldsymbol{C} \boldsymbol{\sigma}_{v_i}.$$

The proof is omitted but is found by expanding Equation 1. The significance of Lemma 4 is that the function $\mu_{\boldsymbol{C}}(\boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j}, \langle \boldsymbol{\mathcal{V}}_{v_i} \cup \boldsymbol{\mathcal{V}}_{v_j} \rangle)$ can be computed in constant time, irrespective of the number of waypoints assigned to either node. This is central in providing a bound on the computational complexity of the algorithm.

Theorem 5. *The worst-case computational complexity of* `Learn-Structure` *to assimilate M tasks of length N is $O(M^2N^2)$.*

Proof. Though not required by the algorithm, we simplify the running-time analysis by assuming that each task is of length N. We make two reasonable assumptions

- Appending a value to a multiset is constant-time ($\mathcal{A} := \mathcal{A} \cup \{\boldsymbol{x}\}$).

- Accessing any node is constant-time ($v_i \in V$).

With these assumptions, there is one line of non-constant cost inside the locus of `Learn-Structure` , finding the minimum similarity measure (Line 5). This line forms an arithmetic series on the number of nodes in the observation graph. From Lemma 4, the update rule for the union of two multisets is independent of their cardinalities and is constant-time. Since this cost is embedded in an arithmetic series on the number of nodes, the worst-case running-time occurs when the number of nodes is maximized, *i.e.* when no nodes are merged. Therefore, the worst-case computational complexity for assimilating M tasks of length N is

$$O(\sum_{m=1}^{M}\sum_{n=1}^{N} mn) \quad \in \quad O(M^2N^2),$$

which is quadratic in the number of waypoints. □

3.5 Estimating CDHMM Parameters

In this section we present a simple one-shot procedure for estimating CDHMM parameters from an observation graph. We use one-shot estimation, in lieu of an EM-type approach, for computational expediency and because we typically lack the data required for complete EM re-estimation procedures. A CDHMM is given by the quintuple $\boldsymbol{\lambda} = (\mathcal{Q}, \boldsymbol{\mathcal{X}}, a, b, \pi)$ where

- $\mathcal{Q}$ is a finite set of states;
- $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^d$ is the observation set of dimension d;
- $a : \mathcal{Q} \times \mathcal{Q} \rightarrow [0, 1]$ is the stationary state-transition probability mass function (pmf);
- $b : \boldsymbol{\mathcal{X}} \times \mathcal{Q} \rightarrow [0, \infty)$ is the stationary observation pdf;
- $\pi : \mathcal{Q} \rightarrow [0, 1]$ is the stationary initial-state pmf;

The function `graph-to-HMM` simply computes the maximum-likelihood CDHMM parameters from the information contained in the observation graph and is the "Maximization" step from the Baum-Welch algorithm. For example, the probability of transitioning from state q_i to state q_j is the number of times the observation graph recorded a transition from node v_i to node v_j, divided by the total number of waypoints assigned to node v_i, written as $a_{j|i} = f_{e_{i\rightarrow j}}/|\boldsymbol{\mathcal{V}}_{v_i}|$. The initial-state probabilities are computed similarly. We do not assume any distribution for the observation-generation pdfs, $b_i(\boldsymbol{x})$, beyond requiring that the mean of the pdf be the sample mean of the waypoints assigned to the node, $\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|q_i\} = \langle\boldsymbol{\mathcal{V}}_{v_i}\rangle$. In practice, however, we typically fit a Gaussian distribution to the waypoints assigned to each node.

Function `graph-to-HMM`

$G_{\boldsymbol{\mathcal{X}}} = (V, V^0, E, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{V}}, f, g)$ is the observation graph.
M is the number of sequences assimilated into $G_{\boldsymbol{\mathcal{X}}}$.

```
 1: $\mathcal{Q} := \emptyset$
 2: $\boldsymbol{\lambda} = (\mathcal{Q}, \boldsymbol{\mathcal{X}}, a, b, \pi)$
 3: for all $v_i \in V$
 4:    create new CDHMM state $q_i$
 5:    $\mathcal{Q} := \mathcal{Q} \cup \{q_i\}$
 6:    create $b_i$ from $\boldsymbol{\mathcal{V}}_{v_i}$ with $\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|q_i\} = \langle\boldsymbol{\mathcal{V}}_{v_i}\rangle$
 7:    $\pi_i := g_{v_i}/M$
 8:    for all $v_j \in V$
 9:       $a_{j|i} := f_{e_{i\rightarrow j}}/|\boldsymbol{\mathcal{V}}_{v_i}|$
10:    end for all
11: end for all
12: return $(\mathcal{Q}, \boldsymbol{\mathcal{X}}, a, b, \pi)$
```

It is simple to show that the running time of this function is $O(|V|^2 + |V|b)$, where $|V|$ is the number of nodes in the observation graph and $O(b)$ is the cost of computing the observation pdf parameters.

3.6 Bounding State Error

Assume that the tasks are actually generated by an unknown *target* CDHMM in an independent identically distributed (*iid*) manner. We derive a bound showing that as more tasks are incorporated, states in the target CDHMM are estimated with exponentially increasing accuracy. It is typical to provide HMM bounds regarding an observation *sequence* (Ephraim & Merhav, 2002), whereas we provide a bound on individual waypoints. This appears to be the strongest statement we can make since our definition of similarity, Equation 1, is memoryless. First, we precisely define the idea that an estimated CDHMM will produce "close" to the correct waypoint at the correct time. Let c_n be the random variable denoting the current state of the CDHMM at time n and the trace of matrix $\boldsymbol{A}$ be written as $\mathrm{tr}[\boldsymbol{A}]$.

Definition 2. *A CDHMM,* $\boldsymbol{\lambda}_1$*,* γ-represents *a state in another CDHMM,* $q_2 \in \boldsymbol{\lambda}_2$*, if there exists a state,* $q_1 \in \boldsymbol{\lambda}_1$*, at the same depth* n *such that*

$$\|\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|c_n = q_1, \boldsymbol{\lambda}_1\} - \mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|c_n = q_2, \boldsymbol{\lambda}_2\}\|_{\boldsymbol{C}} < \gamma.$$

If $\boldsymbol{\lambda}_1$ γ-represents state q_2 in $\boldsymbol{\lambda}_2$ then we write $\boldsymbol{\lambda}_1 \xrightarrow{\gamma} q_2 \in \boldsymbol{\lambda}_2$. This defines what it means for different CDHMMs to generate waypoints within γ of each other at the correct time. Intuitively, it is easier to estimate a state in the target CDHMM if it generates observations more frequently. Furthermore, if a state emits observations with high variance then it will be more difficult to distinguish between different states. This intuition is formalized in the following theorem.

Theorem 6. *Let $\widehat{\boldsymbol{\lambda}}$ be a CDHMM estimated from* `Learn-Structure` *and* `graph-to-HMM` *with $\epsilon \geq 0$ and M* iid *tasks of length at least n, generated by some target CDHMM $\boldsymbol{\lambda}^*$. If the state $q_* \in \boldsymbol{\lambda}^*$ generates observations with finite first and second moments,*

$$\begin{aligned}&\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\}\\ &\quad > \left(1-(1-p_*)^M\right)\left(1-\frac{\operatorname{tr}[\boldsymbol{C}\operatorname{Var}(\boldsymbol{x}|q_*)]}{(\gamma-\sqrt{\epsilon})^2}\right),\end{aligned}$$

where $\gamma > \sqrt{\epsilon}$, $p_* = \mathrm{P}(c_n = q_* | \boldsymbol{\lambda}^*) > 0$, and $\operatorname{Var}(\boldsymbol{x}|q_*)$ is the observation-generation variance of state q_*.

Proof. Let m be the random variable summing the number of times $c_n = q_*$ over each of the M tasks. The probability that state q_* with prior probability p_* generated *at least* one waypoint at time n in M *iid* tasks is

$$\begin{aligned}\Pr\{m > 0 | p_*, M\} &= 1 - \Pr\{m = 0 | p_*, M\}\\ &= 1 - (1-p_*)^M.\end{aligned}$$

Let $\boldsymbol{x} \sim \mathrm{p}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$, where $\mathrm{p}(\boldsymbol{x}|q_*, \boldsymbol{\lambda}^*)$ has finite mean $\boldsymbol{u}^*$ and finite variance $\boldsymbol{\Sigma}^*$. By the triangle inequality and reflexive property,

$$\|\boldsymbol{u}^* - \langle \boldsymbol{\mathcal{V}}_{v_i}\rangle\|_{\boldsymbol{C}} \leq \|\boldsymbol{x} - \boldsymbol{u}^*\|_{\boldsymbol{C}} + \|\boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i}\rangle\|_{\boldsymbol{C}}.$$

From Theorem 3, $\|\boldsymbol{x} - \langle \boldsymbol{\mathcal{V}}_{v_i}\rangle\|_{\boldsymbol{C}} \leq \sqrt{\epsilon}$, for any $\boldsymbol{x} \in \boldsymbol{\mathcal{V}}_{v_i}$. Since $\mathrm{E}_{\boldsymbol{x}}\{\boldsymbol{x}|q_i\} = \langle \boldsymbol{\mathcal{V}}_{v_i}\rangle$ (`graph-to-HMM` Line 6), the state $q_* \in \boldsymbol{\lambda}^*$ will be γ-represented by the estimated CDHMM if q_* generates at least one waypoint such that

$$\|\boldsymbol{x} - \boldsymbol{u}^*\|_{\boldsymbol{C}} \leq \gamma - \sqrt{\epsilon}.$$

By assumption $\gamma > \sqrt{\epsilon}$ and from the Multivariate Chebyshev's Inequality, [1]

$$\Pr\left\{\|\boldsymbol{x}-\boldsymbol{u}^*\|_{\boldsymbol{C}} < \gamma - \sqrt{\epsilon}\right\} > 1 - \frac{\operatorname{tr}[\boldsymbol{C}\boldsymbol{\Sigma}^*]}{(\gamma-\sqrt{\epsilon})^2}.$$

Multiplying this conditional probability by the prior yields

$$\begin{aligned}&\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\}\\ &\quad > (1-(1-p_*)^M)\left(1-\frac{\operatorname{tr}[\boldsymbol{C}\boldsymbol{\Sigma}^*]}{(\gamma-\sqrt{\epsilon})^2}\right),\end{aligned}$$

which completes the claim. $\square$

In more concrete terms,

$$\begin{aligned}&\Pr\left\{\widehat{\boldsymbol{\lambda}} \xrightarrow{\gamma} q_* \in \boldsymbol{\lambda}^*\right\}\\ &\quad > \underbrace{\left(1-(1-p_*)^M\right)}_{(\star)}\underbrace{\left(1-\frac{\operatorname{tr}[\boldsymbol{C}\operatorname{Var}(\boldsymbol{x}|q_*)]}{(\gamma-\sqrt{\epsilon})^2}\right)}_{(\diamond)}\end{aligned}$$

[1]Multivariate Chebyshev's Inequality, $\epsilon^2 \Pr\left\{\|\boldsymbol{x}-\boldsymbol{u}\|_{\boldsymbol{C}} \geq \epsilon\right\} \leq \operatorname{tr}[\boldsymbol{C}\operatorname{Var}(\boldsymbol{x})] + \|\boldsymbol{u}-\mathrm{E}\{\boldsymbol{x}\}\|_{\boldsymbol{C}}^2$

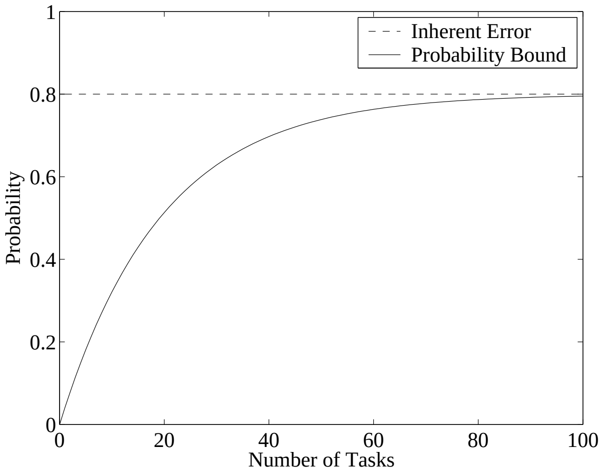


Figure 7: Asymptotic nature of the bound in Theorem 6 for a fixed inherent error rate with $p_* = 0.05$, $\gamma = 1$, and $\epsilon = 0.25$.

The term $(\diamond)$ is in many ways an "inherent" error rate. The inherent error decreases as the

- waypoint variance decreases;
- similarity radius (ϵ) decreases;
- acceptable error (γ) increases.

The term $(\star)$ increases exponentially as the number of tasks, M, increases with the rate determined by the probability that a state generates a waypoint, p_*. This implies that the bound in Theorem 6 is asymptotic to the inherent error rate of the target CDHMM, as in Figure 7, and the estimated CDHMM probably generates the correct waypoint at the correct time.

3.7 Probability of Similarity

To make the experimental results in this paper easier to present, we compress the infinite interval, $\epsilon \in [0, \infty)$, to a bounded interval, $\delta \in (0, 1]$. There are many possibilities to determine δ and we assume that users make positioning errors about a desired waypoint according to a Gaussian distribution. We then require that a waypoint be emitted with high probability,

$$\Pr\left\{\|\boldsymbol{x}-\langle \boldsymbol{\mathcal{V}}_{v_i}\rangle\|_{\boldsymbol{C}}^2 \leq \epsilon\right\} > 1 - \delta. \qquad (2)$$

The squared Mahalanobis distance of a Gaussian random variable is a chi-square random variable. We can then evaluate Equation 2 from the cumulative distribution function (cdf) of the chi-square distribution, shown graphically in Figure 8. As $\delta \to 0$ the definition of similarity becomes loose, $\epsilon \to \infty$, inducing a simpler CDHMM. As $\delta \to 1$

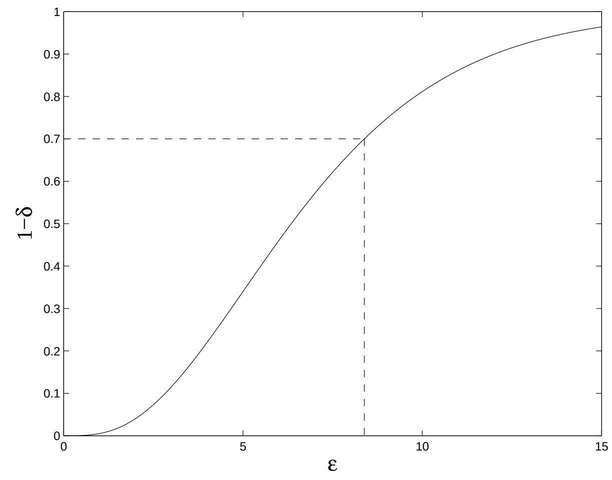


Figure 8: Cumulative Distribution Function of a chi-square random variable with 7 degrees of freedom, χ^2_7. Graphically, we compute Equation 2 by fixing a value of *e.g.* $1-\delta = 0.7$ and finding the intercept on the cdf at $\epsilon \approx 8.4$.

the definition of similarity becomes strict, $\epsilon \to 0$, inducing a more complex model. From this perspective, δ can be considered as a type of "complexity parameter". The experimental results in this paper will vary the parameter δ.

3.8 Prediction

In the prediction phase, the PRP system computes predictions of future waypoints using the CDHMM estimated by the learning algorithm. Optimal prediction, given a model, is a mature topic and can be found in many references (Duda et al., 2001). There are several reasonable choices for a prediction criterion such as Maximum Likelihood (ML), Maximum *A Posteriori*, expectation, etc. In our experiments, ML estimators have performed the best. To compute a prediction, we condition the CDHMM probability distributions on waypoints from the current task and determine the most likely next waypoint,

$$\widehat{\boldsymbol{x}}^*_n \quad = \quad \arg\max_{\boldsymbol{x}_n} \mathrm{p}\big(\boldsymbol{x}_n | \boldsymbol{X}^c_{0:n-1}, \boldsymbol{\lambda}\big)\,,$$

where $\boldsymbol{X}^c_{0:n-1} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}\}$ is the relative-movement representation of the current task. Using standard HMM notation, *i.e.* (Rabiner, 1989), we define the "forward variables" to be the conjunctive likelihood of being in state q_i at time $n-1$ while observing the current task,

$$\alpha_{n-1}(i) \quad \triangleq \quad \mathrm{p}\big(c_{n-1}{=}q_i, \boldsymbol{X}^c_{0:n-1} | \boldsymbol{\lambda}\big)\,.$$

This is a recursive equation that can be computed in standard dynamic-programming time (Rabiner, 1989). We then define the conditional next-state pmf as

$$\nu_n(j) \triangleq \quad \mathrm{P}\big(c_n{=}q_j | \boldsymbol{X}^c_{0:n-1}, \boldsymbol{\lambda}\big) \doteq \quad \frac{\sum\limits_{q_i \in \mathcal{Q}} a_{j|i}\alpha_{n-1}(i)}{\sum\limits_{q_k \in \mathcal{Q}} \alpha_{n-1}(k)}.$$

With this notation, we evaluate the ML prediction as

$$\widehat{\boldsymbol{x}}^*_n \quad \doteq \quad \arg\max_{\boldsymbol{x}_n} \sum_{q_j \in \mathcal{Q}} b_j(\boldsymbol{x}_n)\nu_n(j). \tag{3}$$

It is also straightforward to derive predictions for any time in the future. Equation 3 involves the maximization of a linear mixture of nonlinear functions. Except in degenerate cases, Equation 3 cannot be solved in closed form and we must resort to iterative multivariate maximization techniques, which can be relatively costly. However, in practice, a locally optimal solution can be found quickly.

Regardless of the criterion used (ML, expectation, etc.), a prediction will exist even if the current task is not consistent with the estimated CDHMM. This may result in computing inaccurate predictions. Ideally, the PRP system would only suggest the most accurate predictions to the user. However, the desired waypoint is not known at the time of the prediction, so the PRP system indicates its *confidence*, $\phi_n \in [0,1]$, based on information available at the time of prediction. Confidence of $\phi_n = 1$ indicates that waypoints from the current task fit perfectly with the model, while $\phi_n = 0$ indicates minimum certainty. To this end, we compute the divergence of the CDHMM from complete internal uncertainty while observing the current task. Let the number of states in the CDHMM be $|\mathcal{Q}|$. In our work, we define confidence as the Kullback-Leibler divergence taken $\log$ base $|\mathcal{Q}|$ between the next-state random variable, c_n, and the uniform distribution

$$\begin{aligned}
\phi_n \quad &\triangleq \quad \frac{\mathcal{D}_{\mathrm{KL}}(c_n \parallel \frac{1}{|\mathcal{Q}|})}{\log_2 |\mathcal{Q}|} \\
&= \quad \frac{\sum\limits_j \nu_n(j) \log_2(|\mathcal{Q}|\nu_n(j))}{\log_2 |\mathcal{Q}|} \\
&= \quad 1 - \frac{H(c_n)}{\log_2 |\mathcal{Q}|},
\end{aligned} \tag{4}$$

where $H(\cdot)$ is entropy. Since c_n is a discrete random variable with $|\mathcal{Q}|$ possibilities, the confidence is bounded on the closed interval $0 \le \phi_n \le 1$ for any state distribution and any waypoint sequence. Intuitively, if many states in the CDHMM are likely to produce the next waypoint, then the prediction confidence will be low. On the other hand, a prediction based on the contributions from few states will result in high confidence.

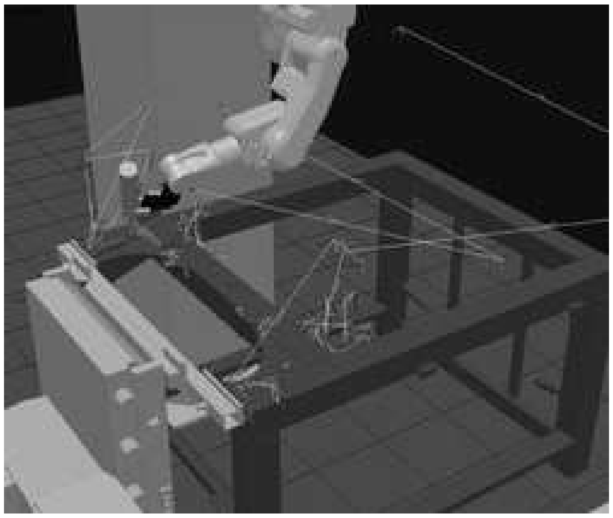


Figure 9: Screen shot from the offline package showing the workspace and waypoints of a program with 18 subroutines.

3.9 Learning Algorithm Summary

In this section, we derived a learning algorithm that estimates the structure of a CDHMM from waypoint sequences. Given an ordered set of waypoints, the CDHMM is uniquely determined by the parameter $\epsilon \geq 0$, which is a function of the parameter $\delta \in (0, 1]$, *cf.* Section 3.7. Decreasing the parameter, $\delta \to 0$, induces a simpler CDHMM, while increasing the parameter, $\delta \to 1$, induces a more complex CDHMM. Predictions of future waypoints are computed using an ML estimation procedure, Equation 3, by conditioning the CDHMM probability distributions on waypoints from the current task. In order to avoid burdening the user with inaccurate suggestions, the PRP system only suggests predictions with high *confidence*, ϕ_n (Equation 4).

4 Offline Programming

In this section, we analyze the performance of the PRP system on five programs created using an offline-programming environment (Figure 9). The programs have between 252 and 1899 waypoints with 16 to 196 subroutines. Collectively, these programs took over 70 work days to complete by a professional robot programmer. The programs were created to automate arc-welding production at several factories in Sweden. Each program was designed to produce a different type of product, from round tables to bed frames. Since the robot programs were developed independently of our PRP system, the behavior of the user was unmodified by these experiments. The programs provide the substrate to verify experimentally the ability of PRP to generate accurate predictions of users creating waypoints for complex, real-world tasks. In Section 4.2, we analyze the prediction accuracy of the PRP system as the complexity of the estimated CDHMM changes, by varying the parameter δ, and show that the system is capable of performing real-time operation. In Section 4.3, we show that prediction confidence, ϕ_n, strongly correlates with prediction accuracy. In Section 4.4, we test the hypothesis that the user is well modeled by a CDHMM by observing the prediction accuracy of the PRP system as more waypoints are incorporated into the learning algorithm.

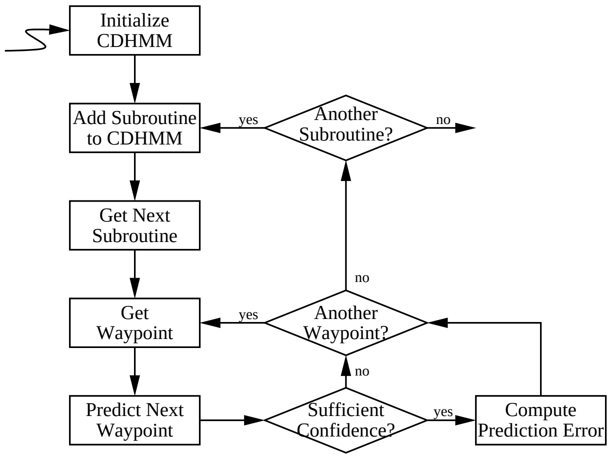


Figure 10: Flow chart for computing predictions for the offline programs.

4.1 Methodology

To compute waypoint predictions, we segment the programs along the subroutines contained in each program. We emulate the user creating the robot program by feeding the subroutine waypoints into the PRP system in a serial fashion. The PRP system computes a prediction of the next position by conditioning the CDHMM on previous waypoints from the subroutine (Equation 3). If the PRP system has sufficient confidence in the prediction (Equation 4) then we consider the prediction valid and compute its error. To determine the prediction error, we compare the predicted waypoint to the next waypoint in the subroutine and consider any difference to be an error in the prediction. At the end of each subroutine we incorporate its waypoints into the CDHMM, building an estimated user model. After predicting the final waypoint in a program, we reinitialize the CDHMM *tabula rasa* and start the prediction of the next program. A flow chart of this procedure is given in Figure 10.

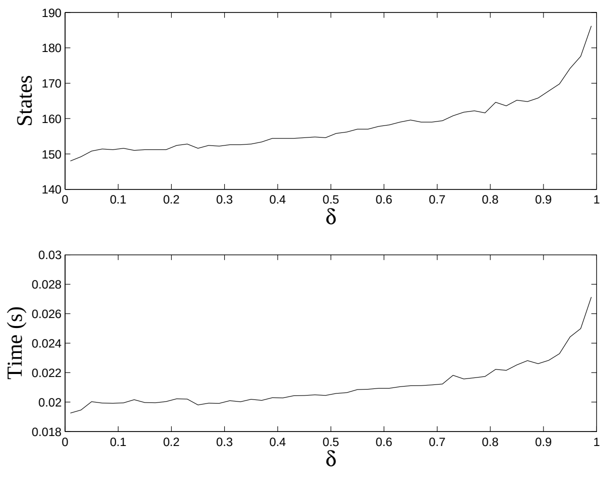


Figure 11: Model complexity as a function of δ.

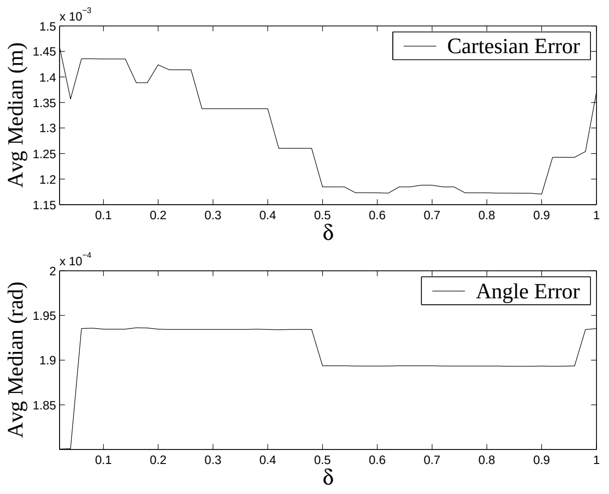


Figure 12: Median Cartesian error and angle error as a function of δ.

The definition of similarity, Equation 1, incorporates a symmetric PD precision matrix, $\boldsymbol{C}$, that represents the inverse covariance matrix of human error when defining a waypoint. We computed the covariance matrix by having users repeatedly move a robot to a point in space and analyzing the residual error.[2] Robotic arc-welding requires approximately 1 millimeter of Cartesian accuracy and about 0.1 radians of angular accuracy. The results presented in this work employ these physical tolerance constraints as a threshold for determining a "useful prediction". The prediction is useful in that a waypoint within these tolerances will generally require no fine-tuning by the user.

At the end of welding, it is common to execute a gross repositioning where the robot moves across the workspace to the next weld, typically on the order of a meter. These movements tend not to be predictable and a small percentage error in predicting the gross repositioning will dominate the mean over many precise movements. We are more interested in the typical prediction error, which is better conveyed by the median.

4.2 Performance as a Function of Model Complexity

In Figure 11, we plot two measures of model complexity, CDHMM states and running time. As expected, the number of CDHMM states tends to increase as δ increases. Also, running time increases as the number of CDHMM states increases. Importantly, Figure 11 shows the average time per waypoint taken to estimate a CDHMM *and* compute a prediction. This time, ~ 25 milliseconds, is suited for real-time use in a robot-programming environment.

To determine the performance of the system as a function of model complexity, we held out one program and computed the performance on the remaining four programs. The held-out program is meant as a sort of test set, but since data are so sparse it is difficult to draw any statistically significant conclusions based on this four-program training set and one-program test set. Indirectly, the parameter δ also determines the accuracy of predictions by controlling the complexity of the CDHMM. In Figure 12, we plot the average median error on the four-program training set as a function of δ. For all values of δ, the angle error is extremely small, less than 0.2 milliradians or about 0.3% of the angle changed during prediction. This is because robots tend to change orientation in a fairly predictable manner during welding and gross repositioning. By exploiting this information, the PRP system can generate extremely accurate angle predictions. However, the Cartesian movements of the robot are determined by the size of the objects in the workspace, a much more unpredictable quantity. When the parameter is too small, the learning algorithm produces a CDHMM that is too simple to represent the user. When the parameter is too large, the CDHMM begins to overfit the data. In PRP, as in many machine-learning applications, "everything should be made as simple as possible, but not simpler" and the *correct* complexity depends on the tasks at hand. In our case, the Cartesian error bottoms out on the interval $\delta \in (0.5, 0.9)$. This implies that a value in the neighborhood of $\delta \approx 0.7$ induces the right amount of complexity on the training set.

[2]The covariance matrix was computed from online-programming experiments, whereas this section deals with offline-programming. From our experience, the principal directions of variance appear unchanged, but the eigenvalues are smaller when using an offline environment.

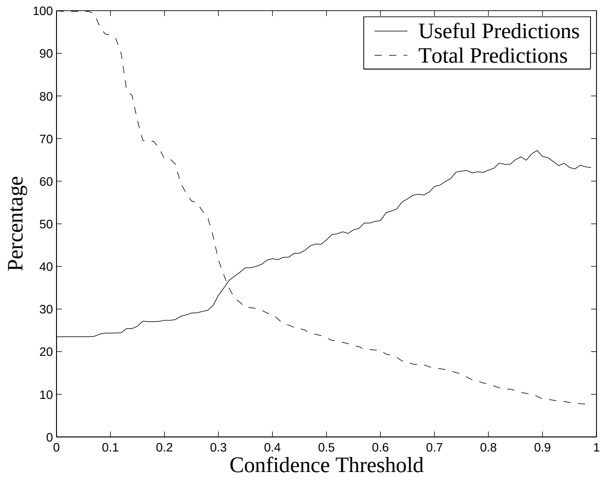


Figure 13: Percentage of predictions and useful predictions as a function of the confidence threshold.

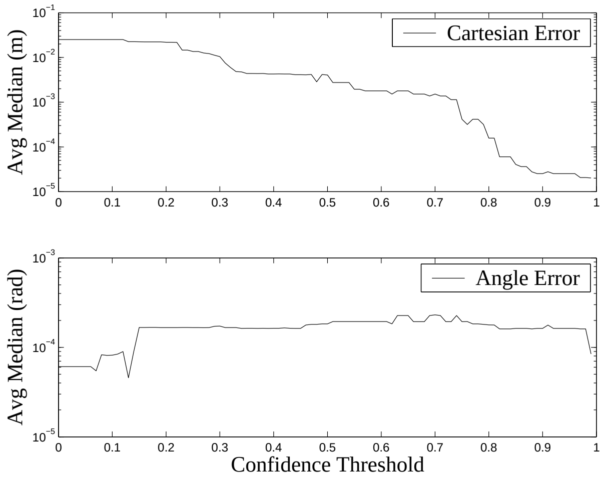


Figure 14: Median Cartesian error and angle error as a function of the confidence threshold.

4.3 Performance as a Function of Confidence

In this section we determine the correlation of prediction confidence to prediction accuracy. Specifically, we discard predictions with confidence below a threshold and analyze the accuracy of the remaining predictions. The hypothesis is that higher-confidence predictions are more accurate than lower-confidence predictions. Inevitably, as the confidence threshold increases there will be accurate predictions discarded due to insufficient confidence. Loosely speaking, the PRP system may have a "lucky guess" and, without knowing the target waypoint at the time of prediction, it is impossible to discriminate between lucky guesses and well-informed estimates. Using the training set, we can determine a confidence threshold that strikes a balance between quality and quantity. We fix the value of $\delta = 0.7$ and have the learning algorithm construct a CDHMM based on Figure 10 for each program in the four-program training set and analyze the performance of the PRP system as a function of confidence threshold. In Figure 13, we plot the percentage of useful predictions as a function of the confidence threshold on the four-program training set. The percentage of useful predictions generally increases as the confidence threshold increases until about $\phi_n \geq 0.8$, when the percentage plateaus. In Figure 14, we plot the average median error as a function of the confidence threshold. For all confidence thresholds the angle error is extremely small, between 0.06 and 0.2 milliradians. The median angle movement during prediction was about 65 milliradians, meaning that the angle prediction error is between 0.09% and 0.3% of the angle change. However, the Cartesian prediction error improves as the confidence threshold increases, until it bottoms out around 0.03 millimeters for $\phi_n \geq 0.8$, well below the physical tolerance required by arc welding. The median Cartesian movement during prediction was about 100 millimeters, meaning that the Cartesian prediction error is about 0.03% of the movement of the robot during a prediction. Empirically, based on this data set, confidence is related to Cartesian error with a normalized correlation coefficient of $\rho = -0.89$, significant to a p-value of $p \ll 0.01$. This strong correlation implies that, on the training set, higher-confidence predictions tend to be more accurate.

4.4 Temporal Performance

In this subsection we use parameters that performed well on the training set, $\delta = 0.7$ and $\phi_n \geq 0.7$, to analyze the performance on the hold-out test program. The CDHMM is initialized *tabula rasa* and incorporates waypoints from the test program incrementally according to the procedure in Figure 10. In Figure 15, we plot the median prediction error on the hold-out program as the user creates waypoints. After incorporating about 400 waypoints, the median Cartesian error stabilizes around 0.25 millimeters. The median Cartesian movement during prediction was 50 millimeters, meaning that the Cartesian prediction error is about 0.5% of the movement of the robot during a prediction. Likewise, the median angle error stabilizes around 0.02 milliradians after about 150 waypoints. The median angle movement during prediction was 0.1 milliradians, meaning that the angle prediction error is about 20% of the angle change. Both of these values are well below the physical tolerance of 1 millimeter and 0.1 radians required by arc welding. This asymptotic-like behavior of the PRP prediction error sug-

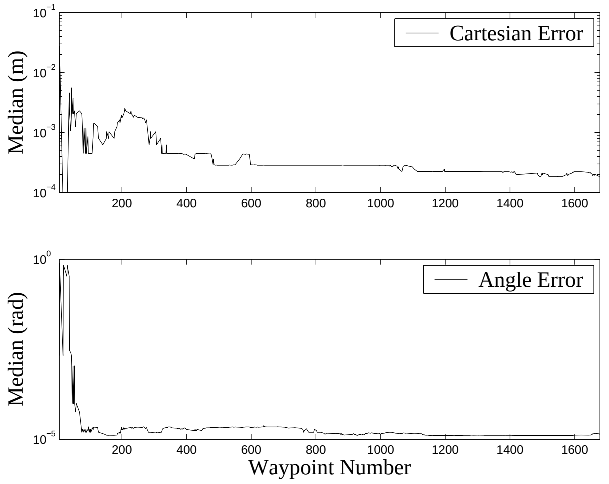


Figure 15: Median error as a function of the number of waypoints incorporated into the CDHMM.

gests that the estimate of the user improves until acquiring sufficient information about the task.[3] The PRP algorithm computed predictions with confidence $\phi_n \geq 0.7$ for about 25% of the waypoints on the hold-out program. This hold-out program originally took over eight work weeks to complete using an offline-programming package. Allowing the PRP system to move the robot to predicted waypoints automatically could save several days in programming time. It it important to remember that none of the programs, either in the training set or the hold-out, were created with a PRP system in mind. In other words, the programs were not created with any motivation for reusing previous work. The predictability of the waypoints results from the inherent similarity of the underlying task. It seems plausible that a programmer creating waypoints with a PRP system would be more likely to behave in a predictable fashion, resulting in greater time savings, making robot-programming environments incorporating a PRP system even less cumbersome.

4.5 Discussion of Results

The results contained in this section validate the core principles of PRP. For the different programs analyzed, the PRP system computed predictions with confidence $\phi_n \geq 0.7$ for between 10% and 30% of possible waypoints. The median Cartesian prediction error for each program was between 0.03 millimeters and 0.25 millimeters, or 0.03% and 0.5% of the distance moved during prediction. The median angle prediction error for each program was between 0.06 milliradians and 0.2 milliradians, or 0.09% and 20% of the angle change during prediction. All of these values are well under the useful physical tolerances required by arc welding of 1 millimeter of Cartesian accuracy and 0.1 radians of angular accuracy. On average, about 60% of predictions computed by the PRP system with a confidence of $\phi_n \geq 0.7$ were useful. The high accuracy of predictions indicates that the CDHMM can identify the inherent similarity contained in complex, real-world robot programs. Since the programs were created without consideration for PRP, the predictability of the waypoints is due to the underlying similarity of the tasks themselves. The PRP system was able to estimate a user model and compute predictions in about 25 milliseconds, which is sufficient for real-time use. The asymptotically decreasing nature of prediction error, as a function of the number of waypoints incorporated into the learning algorithm, shows that the modeling of a robot programmer by a CDHMM is appropriate. The strong correlation between prediction accuracy and prediction confidence gives the PRP system a causal statistic for determining which predictions may be inaccurate. By “filtering” predictions based on confidence, the PRP system can avoid burdening the user with unhelpful suggestions.

Since the programs were created independently of PRP, we have no way of directly translating these results to programming-time savings. However, we can estimate hypothetical savings in programming time based on the results by assigning a time benefit for useful predictions and a time penalty for non-useful predictions. A reasonable benefit for a useful prediction is roughly 90% since it may take the user a short amount of time to determine if the predicted waypoint is sufficiently close to the desired position. A penalty of 10% for a non-useful prediction seems appropriate since a prediction can be rejected out of hand by pressing an “undo” button. Using typical values for the PRP system parameters, $\delta = 0.7$ and $\phi_n \geq 0.7$, would result in a 10% reduction in programming time, which translates to 7 work days saved on the programs analyzed in this section. With a benefit of 75% and a penalty of 25%, the programming-time reduction is 7%. However, some waypoints, such as approach points, do not require much accuracy. Also, it seems likely that many non-useful predictions will also benefit the user; a prediction that is close to the intended waypoint, but still requires some fine-tuning, probably takes less time than user moving the robot manually. Consequently, the binary classification of predictions as useful and non-useful probably represents a lower bound on programming-time savings. In Section 5.3 we present results on reducing programming time in laboratory experiments.

[3]Parenthetically, all programs in the training set also showed the same general asymptotically decreasing shape in their error plots.

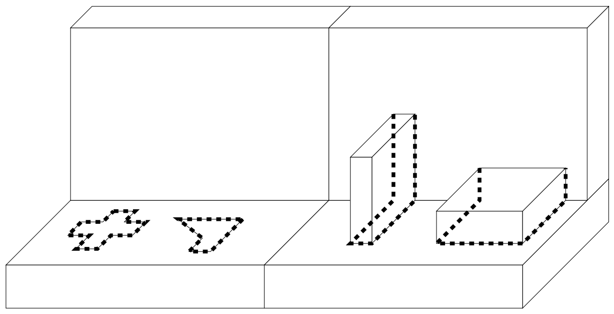


Figure 16: The four patterns for user demonstrations with 13, 6, 10, and 10 waypoints respectively.

5 Online Programming

In this section, we analyze the performance of the PRP system in an online-programming environment. We collected 44 robot programs from 3 users in a laboratory setting. Each user had previous experience with online robot programming and all users were allowed to practice moving the robot until they felt comfortable controlling it. In our terminology, the *repertoire* of the user consists of the four simple tasks shown in Figure 16. The two right-most programs, comprising 10 waypoints each, represent standard arc-welding tasks. The two left-most programs, comprising 13 and 6 waypoints respectively, are planar geometric movements. The mean distance between waypoints in these programs was 186 millimeters. These programs were created in a laboratory setting specifically to test the viability of PRP as a robot-programming tool. In Section 5.2, we analyze the performance of the PRP system when the learning algorithm incorporates programs in different orders. In Section 5.3, we show that the PRP system contributes to a significant reduction in programming time.

5.1 Methodology

To create a program, a user moves the robot with a joystick (Figure 17). When the user feels that the end-effector is sufficiently close to the desired waypoint, he presses a button on the teach pendant to indicate that the current robot position should be stored as the next waypoint in the program. We left the definition of “sufficiently close” to the user. This ambiguity resulted in substantially larger deviations than requiring that the program be viable for arc welding.

To compute the precision matrix from Equation 1, we asked users to move the end-effector repeatedly to an unreferenced point in space by controlling the robot with the joystick. We then computed the residual error and set the precision matrix equal to the inverse of the covariance, $\boldsymbol{C} = \mathrm{Var}(\boldsymbol{x})^{-1}$. Similarity in robot programs may occur in short subsequences. In these cases, conditioning the CDHMM on waypoints from the entire program may cause the PRP system to generate poor predictions (Equation 3), even if we discard predictions below a confidence threshold (Equation 4). This can be avoided by computing predictions based on a *horizon*, h, of recent waypoints,

Figure 17: Creating waypoints for the patterns in Figure 16 with an ABB IRB140.

$$\widehat{\boldsymbol{x}}_n^* \quad = \quad \arg\max_{\boldsymbol{x}_n} \mathrm{p}\big(\boldsymbol{x}_n | \boldsymbol{X}^c_{n-h:n-1}, \boldsymbol{\lambda}\big) \,.$$

The experiments in this section will incorporate this notion of a horizon explicitly.

5.2 Leave-One-Out Performance

As mentioned in Section 3, the CDHMM learning algorithm is sensitive to the order in which robot programs are incorporated into the model. To analyze this effect, we created 44 permutations of the 44 robot programs from Figure 16 so that their waypoints are introduced into the learning algorithm in a differing order. In Figure 18 we plot the median number of states as a function of δ. The error bars in Figure 18 show the minimum and maximum number of states caused by the different permutations. Figure 18 implies that varying the parameter δ has a much greater impact on model complexity than does the order in which the robot programs are introduced into the learning algorithm.

Using the 44 permutations, we also computed the leave-one-out statistics for the accuracy of two prediction criteria. The first criterion uses only high-confidence predictions ($\phi_n \geq 0.8$) over a relatively long horizon ($h = 3$). The second criterion allows for lower-confidence predictions ($\phi_n \geq 0.5$) over a shorter horizon ($h = 2$). In Fig-

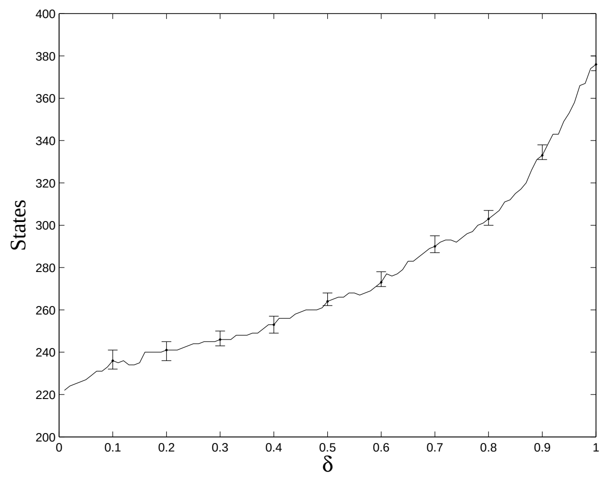


Figure 18: Median number of states as a function of δ, over the 44 different orderings of the online robot programs collected from Figure 16. The error bars indicate the maximum and minimum states induced by the permutations.

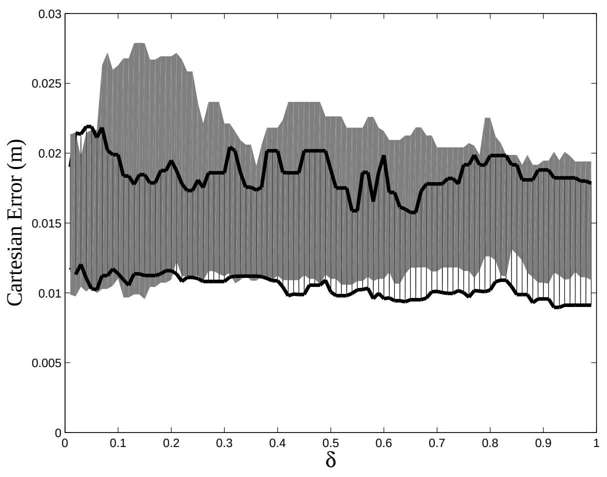


Figure 19: Cartesian errors caused by the different orderings of the online robot programs as a function of δ. The solid surface is the central 50% distribution for a high-confidence prediction criterion and the mesh surface shows the central 50% distribution for a low-confidence prediction criterion.

ure 19, we plot the Cartesian error for the 44 permutations as a function of δ for the two prediction criteria. The average median Cartesian error for both criteria, for most values of δ, was about 15 millimeters, or 8% of the distance traveled by the robot. The substantial overlap between the distributions means that neither criterion is different in a statistically significant sense. While the impact of model complexity, induced through δ, has a significant impact on Cartesian prediction accuracy[4], the influence from the order in which programs are incorporated into the CDHMM dominates.

5.3 Impact on Programming Time

A user of the PRP system will be most interested in the reduction of the time needed to create robot programs. To determine the impact on programming time, the users were asked to complete one of the tasks from Figure 16. The baseline is the time taken to create a robot program without PRP prediction support. The PRP system constructed a CDHMM from waypoints of other robot programs using a value of $\delta = 0.8$. We then asked users to create the programs with the PRP system offering prediction support with two different prediction criteria. As in Section 5.2, one criterion incorporates high-confidence predictions over a long horizon ($\phi_n \geq 0.8, h = 3$) and the other criterion allows low-confidence predictions over a short horizon ($\phi_n \geq 0.5, h = 2$). During programming, the user moves the robot with a joystick. When the user feels that the current end-effector position is sufficiently close to the target waypoint, he presses a button on the teach pendant. If the PRP system computes a prediction of sufficient confidence, this waypoint is suggested to the user with an audible signal. The user can ignore the suggestion or he can allow the PRP system to move the robot to the predicted waypoint automatically by holding down another button. The user can stop the PRP system from moving the robot by releasing the button if, for example, there is an obstacle in its path. It is not uncommon for the user to decide that the prediction must be refined. Fine-tuning can be done using conventional joystick positioning or the previous waypoint can be restored by pressing an “undo” button. This fine-tuning or undoing time was included in the total programming time for that PRP prediction criterion result.

The impact of the PRP system on programming time is summarized in Table 1. The first row shows the baseline programming time used to complete the tasks without PRP prediction support. The second and third rows show the impact of PRP prediction using the high-confidence and low-confidence criteria respectively. The Wilcoxon rank sum test confidence values (Fraser, 1957) in the final column indicate the statistical significance between the PRP prediction criteria and the baseline programming times. From Table 1 we see that programming time was reduced by over one third when using either PRP prediction criterion, with extremely high statistical confidence. The difference in pro-

[4]High-conf: $\rho = -0.83$, Low-conf: $\rho = -0.61$; both have a p-value of $p \ll 0.01$.

Prediction Criterion	mean (sec)	std (sec)	change	Wilcoxon Conf
None	292.2	78.61	N/A	N/A
$\phi_n \geq 0.8, h=3$	193.2	32.07	-33.88%	99.95%
$\phi_n \geq 0.5, h=2$	178.0	33.39	-39.08%	99.99%

Table 1: Programming time used to complete the tasks in Figure 16 with no prediction, high-confidence prediction, and low-confidence prediction.

gramming time between the two PRP prediction criteria was not significant.

5.4 Discussion of Results

The results contained in this section demonstrate the viability of PRP as a robot-programming tool. The prediction error for these online-programming tasks was much larger than the offline-programming counterparts. In the online-programming experiments, the prediction error was about 15 millimeters, or 8% of the distance traveled by the robot. In the offline-programming experiments, the prediction error was less than 0.25 millimeters, or 0.5% of the distance traveled by the robot. This difference is primarily due to users providing their own definition of "sufficiently close" in the online-programming case. Despite this variation, the PRP system was able to assist users in reducing programming time by over a third, in a statistically significant manner.

6 Conclusions

We presented a novel method for predicting the waypoints in manipulator robot programs called Predictive Robot Programming. We derived a learning algorithm that estimates the structure of CDHMMs based on previously created waypoints. This CDHMM is then used to predict future waypoints. On complex, real-world robot programs, the PRP system was able to generate a large percentage of highly accurate predictions. On all programs analyzed, the median Cartesian prediction error was well under a millimeter and the median angular prediction error was well under a milliradian. Both of these values are below the physical tolerances required by the target process, robotic arc welding. In a laboratory setting, we showed that the PRP system was able to reduce programming time by over a third in a statistically significant manner. These results suggest that PRP is a viable tool for reducing the time needed to program industrial manipulator robots.

Not recorded in Table 1 was the number of collisions with objects in the environment. However, collisions are not uncommon during conventional operation when humans create automation programs since, for example, arc welding requires submillimeter separation from the workpiece. Industrial manipulators are designed to handle the eventual collision without inflicting damage. However, this sheds light on one potential problem with PRP in an online-programming environment: showing the user the position of the predicted waypoint before moving the robot. There does not seem to be an efficient solution that allows the user to visualize *a priori* where the PRP system intends to move the robot. One potential solution is to show the user with a virtual-reality model (*i.e.* offline system) where the predicted waypoint is, allowing the user to determine if the prediction is appropriate. But this implementation may cause the user to spend more time determining if the waypoint is acceptable than by simply allowing the PRP system to move the robot automatically and (potentially) undoing it. Our solution was to have the user press a button if he decides to let the PRP system move the robot. The PRP system then moves toward the predicted waypoint slowly at first and then quickly comes to full speed. If the user thinks the prediction is unacceptable, or if a collision may occur, then he can release the button to stop the PRP system immediately. This solution appears to work well in practice.

Currently, the PRP system identifies similarity to previous subtasks using a single, monolithic CDHMM. It may be helpful to allow the user to provide the PRP system with labeled subtasks that may occur in the future. The PRP system could then create many smaller CDHMMs that describe these patterns. By incorporating semi-supervised learning techniques, the PRP system could improve its ability to recognize and predict these patterns, in addition to those it currently identifies.

Acknowledgments

We would like to thank the reviewers for insightful comments that greatly improved the presentation of the material. We are grateful to Martin Strand and Per-Ulrik Eriksson for very helpful discussion and providing the offline-robot programs. 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*.

Bahlman, C., & Burkhardt, H. (2001). Measuring HMM similarity with the Bayes probability of error and its application to online handwriting recognition. *Proceedings of the 6th International Conference on Document Analysis and Recognition*.

Bilmes, J. A. (1997). *A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models* (Technical Report TR-97-021). Berkeley International Computer Science Institute.

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.

Chen, J., & Zelinsky, A. (2003). Programming by demonstration: Coping with suboptimal teaching actions. *International Journal of Robotics Research*, *22*.

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

Ephraim, Y., & Merhav, N. (2002). Hidden Markov processes. *IEEE Transactions on Information Theory*, *48*.

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

Fraser, D. A. S. (1957). *Nonparametric methods in statistics*. John Wiley.

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.

Gassiat, E., & Boucheron, S. (2003). Optimal error exponents in hidden Markov models order estimation. *IEEE Transactions on Information Theory*, *48*.

Hannaford, B., & Lee, P. (1991). Hidden Markov model analysis of force/torque information in telemanipulation. *International Journal of Robotics Research*, *10*.

Hovland, G., Sikka, P., & McCarragher, B. (1996). Skill acquisition from human demonstration using a hidden Markov model. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Hundtofte, C. S., Hager, G. D., & Okamura, A. M. (2002). Building a task language for segmentation and recognition of user input to cooperative manipulation systems. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Iba, S., Paredis, C. J., & Khosla, P. K. (2002). Interactive multi-modal robot programming. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Iba, S., Paredis, C. J., & Khosla, P. K. (2003). Intention aware interactive multi-modal robot programming. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Li, M., & Okamura, A. M. (2003). Recognition of operator motions for real-time assistance using virtual fixtures. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Mason, M. T. (2001). *Mechanics of robotic manipulation*. MIT Press.

Merhav, N., Gutman, M., & Ziv, J. (1989). On the estimation of the order of a Markov chain and universal data compresion. *IEEE Transactions on Information Theory*, *35*.

Morgan, F. (2000). *Geometric measure theory: A beginner's guide*. Academic Press. Third edition.

Nicolescu, M. N., & Matarić, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, *31*.

Perzanowski, D., Schultz, A. C., Adams, W., Marsh, E., & Bugajska, M. (2001). Building a multimodal human robot interface. *IEEE Intelligent Systems*, *16*.

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Copmutation*, *3*.

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

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

Rydén, T. (1995). Estimating the order of hidden Markov models. *Statistics*, *26*, 345–354.

Singh, R., Raj, B., & Stern, R. M. (2002). Automatic generation of sub-word units for speech recognition systems. *IEEE Transactions on Speech and Audio Processing*, *10*, 89–99.

Solis, J., Avizzano, C. A., & Bergamasco, M. (2002). Teaching to write Japanese characters using a haptic interface. *International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Stolcke, A., & Omohundro, S. (1994). Inducing probabilistic grammars by Bayesian model merging. *International Conference on Grammatical Inference*.

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

Tso, S., & Liu, K. (1997). Demonstrated trajectory selection by hidden Markov model. *Proceedings of the IEEE International Conference on Robotics and Automation*.

Yang, J., Xu, Y., & Chen, C. S. (1997). Human action learning via hidden Markov model. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *27*.