Predictive Robot Programming

Kevin R. Dixon[†] krd@cs.cmu.edu
Martin Strand[‡] martin.strand@se.abb.com
Pradeep K. Khosla[†] pkk@ece.cmu.edu

[†] *Department of Electrical & Computer Engineering*
Carnegie Mellon University
Pittsburgh, PA 15213 USA

[‡] *ABB Corporate Research*
721 73 Västerås, SWEDEN

Abstract

One of the main barriers to automating a particular task with a robot is the amount of time needed to program the robot. Decreasing the programming time would facilitate automation in domains previously off limits. In this paper, we present a novel method for leveraging the previous work of a user to decrease future programming time: predictive robot programming. The decrease in programming time is accomplished by predicting waypoints in future robot programs and automatically moving the manipulator end-effector to the predicted position. To this end, we have developed algorithms that construct simple continuous-density hidden Markov models by a state-merging algorithm based on waypoints from prior robot programs. We then use these models to predict the waypoints in future robot programs. While the focus of this paper is the application of predictive robot programming, we also give an overview of the underlying algorithms used and present experimental results.

1 Introduction and Motivation

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

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
September 30 - October 4, 2002, EPFL, Switzerland

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.

In Section 2 we give an overview of the key ideas behind modeling and predicting user actions. We describe the learning algorithm in Section 3, the prediction scheme in Section 4, and present experimental results in Section 5. We place this research in the context of related work in Section 6 and give conclusions in Section 7.

2 Modeling and Prediction

Conceptually, the PRP system operates in two distinct phases (Figure 1). In the first phase, the PRP system constructs a statistical model based on prior observations of user actions. During the second phase, the PRP system 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. In our model, we view these shortcomings as sources of uncertainty and user observations are considered noise corrupted. A robot program can be described by the sequence of waypoints that specify the location and orientation of the end-effector. In the PRP domain, the *user observations* are the waypoints of robot programs, which are real-valued vectors sampled at discrete time intervals. 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).

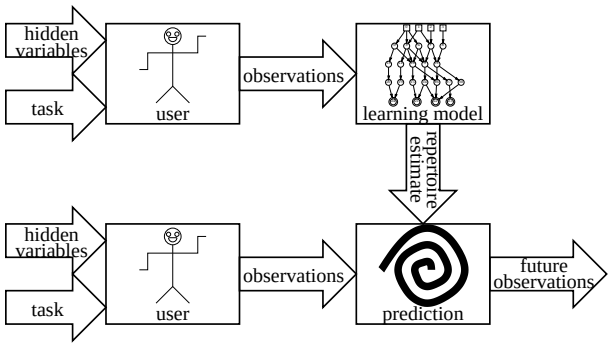

Figure 1: *Conceptual diagram of the operation of the PRP system. First, create a statistical model of user actions from prior observations. Second, use the model to compute predictions of future observations.*

General-purpose six degree-of-freedom (6DOF) manipulators can be described by a Cartesian-space location and orientation of the end-effector. Using this Cartesian-space end-effector description, a waypoint is given by the homogeneous coordinate transform matrix ${}^{G}\mathbf{w}_n$ 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 $\mathbf{W} = \{{}^{G}\mathbf{w}_0, {}^{G}\mathbf{w}_1, \ldots, {}^{G}\mathbf{w}_N\}$ that specify the location and ori-

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
September 30 - October 4, 2002, EPFL, Switzerland

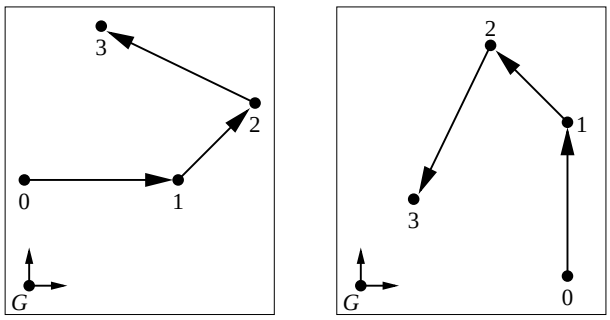

Figure 2: *Though different in a global frame, G, the relative movement of these patterns are the same.*

entation of the end-effector at discrete intervals. The relative change between waypoint m and waypoint n is ${}^{m}\mathbf{w}_n = {}^{m}\mathbf{w}_G {}^{G}\mathbf{w}_n$, where ${}^{G}\mathbf{w}_n$ is premultiplied by the inverse transform ${}^{m}\mathbf{w}_G$. A relative-waypoint program is written as $\widetilde{\mathbf{W}} = \{{}^{0}\mathbf{w}_1, {}^{1}\mathbf{w}_2, \ldots, {}^{N-1}\mathbf{w}_N\}$ and describes the relative change between sequential waypoints. Specifying robot programs by their relative-movement information, instead of absolute positions, has the distinct advantage of yielding rotation and translation independence. For example, the two robot programs shown in Figure 2 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. Homogeneous coordinate transforms are derived and discussed in many references such as [3].

3 Creating the Model

In this section, we describe how observations from disparate tasks are assimilated into a common model representing the repertoire of the user. Essentially, the algorithm operates by first building the maximal-state Directed Acyclic Graph (DAG) suggested by the prior user observations and then repeatedly merges statistically *similar* states so that a *simple* CDHMM results (Figure 3). While not given in this paper, it can be shown that this algorithm produces CDHMMs with an irreducible number of states and is a type of locally minimal graph (a more formal derivation and proof is given in [4] along with the computational complexity of the learning algorithm). Creating simple CDHMMs, such as those produced by this algorithm, are of particular importance in domains such as PRP, where the availability of training data is severely limited. By creating simple models, the algorithm has fewer tunable parameters to assign and can be successfully trained on smaller data sets.

The set of prior observations to the state-merging

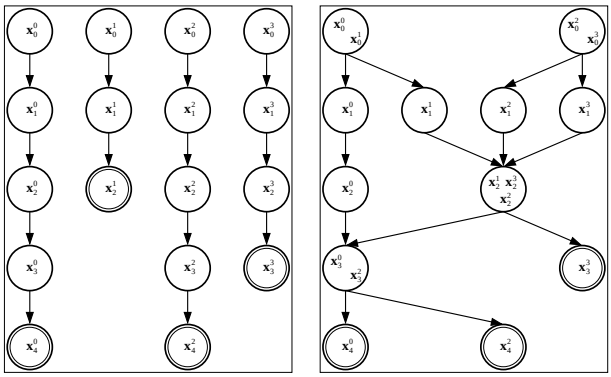

Figure 3: *Hypothetical DAG before assimilating the set of prior observations (left) and after state-merging algorithm (right).*

CDHMM algorithm is the set of robot programs created by users. We make no restrictions that the programs correspond to the same physical task (*e.g.* painting a car or arc welding a boat hull). The programs can describe fundamentally different physical tasks and can have different numbers of waypoints. For convenience, we rewrite each relative waypoint into a stacked column vector, $\mathbf{x}_n = \mathrm{vec}({}^{n-1}\mathbf{w}_n)$. To determine if two states, v_i and v_j, are *similar* we form the hypothesis that the multisets of observations assigned to the states, $\boldsymbol{\mathcal{X}}_{v_i}$ and $\boldsymbol{\mathcal{X}}_{v_j}$, were drawn from the same distribution, in other words created by the same *latent state* in the repertoire of the user. The similarity hypothesis is accepted if

$$\Pr\left\{d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle) \leq \epsilon\right\} > 1 - \delta, \qquad (1)$$

where $\langle \mathcal{A} \rangle$ is the sample mean of the multiset $\mathcal{A}$, $\epsilon \in [0, \infty)$ is a non-negative threshold, and $\delta \in (0, 1]$ is the probability of rejection. In our work, the distance metric from Equation 1 is defined as

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

which is a sum of squared Mahalanobis distances. If we assume that user errors follow a Gaussian distribution, we can compute ϵ as a function of δ from

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

where $Q(y)$ is 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. Therefore, if $d(\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}, \langle \boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j} \rangle) \leq \epsilon$, then the states v_i and v_j are merged, yielding a simpler model and the merged node is assigned the union of the two multisets of observations, $\boldsymbol{\mathcal{X}}_{v_i} \cup \boldsymbol{\mathcal{X}}_{v_j}$. After all similar states are

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
September 30 - October 4, 2002, EPFL, Switzerland

merged, the observation-generation pdf for each state is based on the statistics of the observations assigned to the state. The transition probabilities between states are found by simple frequency counts.

4 Predicting The Future

Once the model is created, the PRP system computes predictions of future waypoints while the user is programming a new task. Since many robot programs contain repeated subsequences, we compute predictions based on a horizon, h, of observations from the current task and discard observations beyond the horizon. Prediction can be based on many criteria, such as Maximum Likelihood (ML) etc., and in this work we compute the Expected Value (EV) of future waypoints. Given a model and observations from the current task, the expectation of the next observation is

$$\widehat{\mathbf{x}}_n = \mathrm{E}_{c_n}\{\mathbf{x}\} = \mathrm{E}_{c_n}\big\{\mathrm{E}_{\mathbf{x}|c_n}\{\mathbf{x}\}\big\}. \tag{3}$$

The prediction, $\widehat{\mathbf{x}}_n$, represents the expected change in position of end-effector from its current position, based on the estimated repertoire of the user. The complete derivation for the computation of Equation 3 is given in [4], but we summarize it here. Using standard HMM notation (*i.e.* [6]), $a_{j|i}$ is the transition probability from state i to state j, $b_j(\mathbf{x})$ is the likelihood of observation $\mathbf{x}$ in state j, and α^j_{n-1} is the conjunctive likelihood of being in state j at time $n-1$ and observing the current task. Using this notation, Equation 3 can be computed as

$$\widehat{\mathbf{x}}_n \doteq \sum_j \underbrace{\frac{\sum_i a_{j|i}\alpha^i_{n-1}}{\sum_k \alpha^k_{n-1}}}_{(\star)} \underbrace{\int_{\mathbf{x}} \mathbf{x} b_j(\mathbf{x}) d\mathbf{x}}_{(\diamond)}. \tag{4}$$

Essentially, Equation 4 states that the expectation of the next observation is the sum of expected values of each state ($\diamond$), weighted by the probability that the state generates the next observation given observations from the current task ($\star$). Computing ($\diamond$) is usually straightforward as many distributions are parameterized by their mean and ($\star$) can be computed at typical dynamic-programming costs.

Regardless of the criterion used (EV, ML, etc.), 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

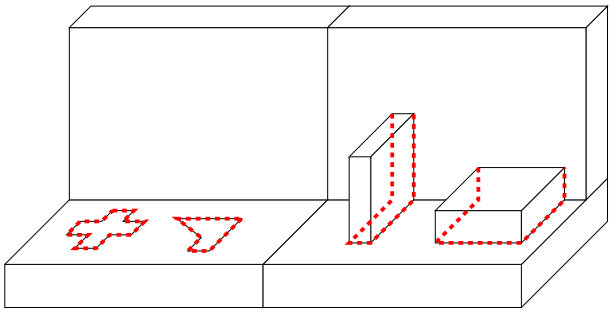

Figure 4: *The four patterns for user demonstrations with* 13, 6, 10, *and* 10 *waypoints respectively.*

Figure 5: *Creating waypoints for the patterns in Figure 4 with an ABB IRB140.*

certainty. The confidence value should also be independent of the number of observations and states in the model. 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

$$\phi_n \triangleq \frac{\mathcal{D}_{\mathrm{KL}}(c_n \parallel \frac{1}{|\mathcal{Q}|})}{\log_2 |\mathcal{Q}|} = 1 - \frac{H(c_n)}{log_2|\mathcal{Q}|}. \tag{5}$$

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.

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
September 30 - October 4, 2002, EPFL, Switzerland

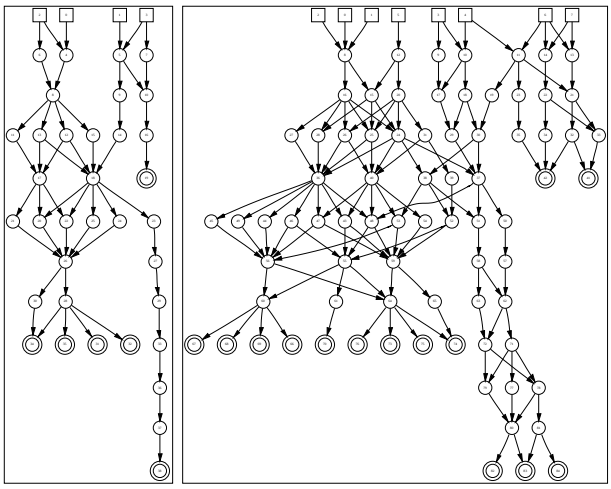

Figure 6: *CDHMMs constructed from small δ (left) and large δ (right).*

Small δ	**mean**	**std**
Prediction Error	4.260 cm	1.580 cm
Percent Error	22.42%	8.316%
States	52.53	2.872
Transitions	89.46	8.646
Large δ	**mean**	**std**
Prediction Error	2.587 cm	1.291 cm
Percent Error	13.61%	6.789%
States	185.7	2.740
Transitions	284.5	4.325

Table 1: *Leave-one-out statistics for CDHMMs with variations on δ.*

Prediction Criterion	**mean** (*sec*)	**std** (*sec*)	**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 4 with no prediction, high-confidence prediction, and low-confidence prediction.*

5 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 4. 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 geometric movements. To create a program, a user moves the robot end-effector with a joystick (Figure 5). When the user feels that the end-effector is sufficiently close to the desired waypoint, he presses a button on the teach pendant. The robot programs were assimilated into a CDHMM and the learning algorithm took well under a second to construct an estimated user repertoire. To give some intuition of the effects of the rejection probability, δ (Equation 2), we show the topology of CDHMMs created using a small rejection probability (39 states, 51 transitions) and a large rejection probability (85 states, 139 transitions) in Figure 6. It is typical that a larger rejection probability, δ, results in more CDHMM states since this allows the algorithm fewer opportunities to merge states (*cf.* Equation 1). As with any machine-learning algorithm, there is a danger of overfitting the data as δ becomes "too large" and does not allow the algorithm to merge truly similar states.

To determine the prediction accuracy of the PRP system, we computed the prediction performance against human-generated data. The leave-one-out statistics of the Cartesian-space prediction error are 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 4 was 19 centimeters, meaning that the average prediction error of the system was roughly 13% of total distance traveled by the end-effector. This low error indicates that the algorithms can compute accurate predictions based on fairly small training sets.

However, a user of the system will be more 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 4, with predictions computed from the estimated repertoire based on the prior observations. If the PRP system computes a valid prediction (*i.e.* confidence from Equation 5 exceeds a threshold) then the user can allow the PRP system to move the end-effector to the predicted waypoint. It is not uncommon for the user to decide that the prediction must be refined using the joystick, and this fine-tuning time was 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 deadman switch on the teach pendant. The first row of Table 2 summarizes the time taken to program the tasks in Figure 4 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 ($h = 3$) of observations from the current program. 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 rel-

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
September 30 - October 4, 2002, EPFL, Switzerland

atively short horizon ($h = 2$). In the cases when the PRP system was permitted to compute predictions, the decrease in programming time was both substantial and statistically significant. It is also worth noting that allowing low-confidence predictions rather than only high-confidence predictions tended to reduce programming time, though the difference between the two prediction criteria was not statistically significant. This suggests that users still find less accurate, low-confidence predictions helpful.

6 Related Work

Though there have been few attempts at decreasing online robot-programming time using machine-learning techniques, the underlying ideas of our PRP system have been explored. The seminal work on HMM applications is [6] and the archetypal HMM-training technique is the forward-backward algorithm. However, this algorithm has several long-standing problems, such as its tendency to leave many superfluous parameters in the model, its reliance on large training sets, and requiring a fixed HMM topology. In many domains, assuming a model topology *a priori* is not possible and training data may be extremely difficult to obtain, as in PRP. Central to the development of the algorithms mentioned in this paper (and derived formally in [4]) are the low availability of training data and an unknown model topology. The "standard tricks" to reduce the number of parameters in an HMM, such as left-right models or assuming diagonal covariance matrices (in the continuous case), do not perform well on the small training sets allowed by PRP. Researchers have developed a variety of techniques to address the short-comings of the forward-backward algorithm, such as parameter extinction [2] and topology discovery [8]. However, these powerful techniques rely on large training sets. There are results that suggest globally optimal HMM training is intractable [1] and attention has shifted to special subclasses of HMMs [7] to derive more optimistic results.

In the man-machine-interface 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 [5], researchers used Dynamic Time Warping (DTW) to decompose observations of human tasks into symbolic, predefined primitives. In [9], researchers applied the standard forward-backward algorithm to synthesize human operation of telerobotic tasks.

7 Conclusions

In order to decrease online robot-programming time, we have developed algorithms for a novel application: predictive robot programming. Our PRP system operates in two phases. The first phase constructs a simple CDHMM by a state-merging algorithm based on waypoints from prior robot programs. This CDHMM is used during the second phase to compute predictions of waypoints in future robot programs. By using a relative-waypoint representation, our PRP system can recognize patterns and compute predictions independent of rotation and translation. We also presented experimental results showing that our PRP system decreased robot-programming time significantly.

Acknowledgments

We would like to thank Mike Seltzer and Chris Paredis for asking tough questions and giving enlightening suggestions and the anonymous IROS 2002 reviewers for their very helpful comments. 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

[1] N. Abe and M. K. Warmuth. On the computational complexity of approximating probability distributions by probabilistic automata. *Machine Learning*, 9, 1992.

[2] M. Brand. An entropic estimator for structure discovery. In *Advances in Neural Information Processing Systems*, volume 11, pages 723–729, 1999.

[3] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley, second edition, 1989.

[4] K. R. Dixon and P. K. Khosla. Unsupervised model-based prediction of user actions. Technical report, Institute for Complex Engineered Systems, Carnegie Mellon University, 2002.

[5] S. B. Kang and K. Ikeuchi. A robot system that observes and replicates grasping tasks. In *IEEE Fifth International Conference on Computer Vision*, pages 1093–1099, 1995.

[6] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[7] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56(2), 1998.

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

[9] J. Yang, Y. Xu, and C. S. Chen. Hidden Markov model approach to skill learning and its application to telerobotics. *IEEE Transactions on Robotics and Automation*, 10(5):621–631, 1994.