# Programming Complex Robot Tasks by Prediction: Experimental Results

Kevin R. Dixon
krd@cs.cmu.edu
Dept of Electrical & Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Pradeep K. Khosla
pkk@ece.cmu.edu
Dept of Electrical & Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213 USA

***Abstract*— One of the main obstacles to automating production is the time needed to program the robot. Decreasing the programming time would increase the appeal of automation in many industries. In this paper we analyze the performance of a Predictive Robot Programming (PRP) system on complex, real-world robotic tasks. The PRP system attempts to decrease programming time by predicting the waypoints of a robot program based on previous examples of user behavior. We show that the PRP system is able to generate a large percentage of useful and highly accurate predictions, resulting in a potentially great amount of time saved.**

## I. INTRODUCTION AND MOTIVATION

Manipulator robots perform a wide variety of industrial tasks. As the capabilities of robots increase, they are being used to perform more sophisticated tasks. Automating production can result in increased productivity and decreased costs. However, one of the main obstacles to robotic automation is the time needed to program the manipulator. Simple tasks can take days to program, while more complex tasks can take weeks or months. Significant programming time means that production may have to be halted temporarily. In many cases, the cost of halting production may be prohibitive. Decreasing the time needed to program a task will increase the appeal of robotic automation in many industries.

A 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 components, robot programmers typically spend the majority of time defining the sequence of positions, called *waypoints*. Despite their complexity, most tasks can usually be decomposed into simpler subtasks. These subtasks may be repeated many times throughout the program directly or in some modified form. Due to the cumbersome nature of current systems, most programmers create these subtasks from scratch each time. If a system could identify these repeated subtasks, instead of discarding previous work, programming time could be reduced by completing the subtask for the user. In this paper, we present experimental results in predicting the waypoints of complex industrial tasks. Specifically, the tasks are offline robotic programs consisting of several thousand waypoints created to automate arc-welding production at several factories. Each program was designed to produce different products, from bed frames to round tables. On all programs, we are able to generate a large percentage of useful and highly accurate predictions.

## II. PREDICTIVE ROBOT PROGRAMMING

The results presented in this work are an experimental validation of our previous work in Predictive Robot Programming (PRP) [3]. In PRP, the system attempts to decrease programming time by predicting the position of future positions based on previous tasks and completing the task for the user. There are two main steps in PRP. The first step is estimating a model of user behavior and the second step is using the model to predict future waypoints. In PRP, the user is modeled as an automaton. From a high-level perspective, the user selects a *task* from her *repertoire*, $R$. At discrete time steps the user generates an observation, or waypoint. A waypoint is the desired location and orientation of the end effector, a continuous vector. The observations are based solely on the current *state* of the task. However, it is impractical to instrument fully any realistic working environment. Consequently there will always be hidden, or latent, causes for user behavior. As a result, the transitions between states in the automaton can be considered stochastic. An automaton with stochastic transitions is a type of Markov chain. Furthermore, humans have poor repeatability and precision, so user observations are considered noise corrupted. Since both the observations and state transitions are stochastic, one natural user model is a Continuous-Density Hidden Markov Model (CDHMM). After estimating a CDHMM from the previous tasks, the model is then used to predict future observations. While the user is performing the current task, the CDHMM predicts the position of the next waypoint. These predicted waypoints are suggested to the user. If the prediction is accepted, the user may allow the PRP system to position the end-effector automatically, instead of the user moving the robot. Compared to the time required to move a robot manually, automatic positioning of a robot is essentially instantaneous. This results in significant savings in robot programming time.

To appear in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems
October 27 - 31, 2003, Las Vegas, Nevada USA

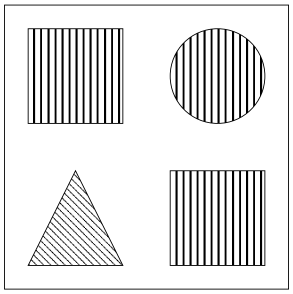

Fig. 1. The concept of similarity is a continuum.

It is important to note that the observations are the result of human activity of significant duration. The tasks analyzed in this work typically take weeks to complete. Because creating such programs is so arduous there is a severe lack of data with which to estimate a user model. As a result, the model should be kept as simple as possible. However, the PRP system has no *a priori* knowledge of the general structure of the repertoire or what tasks may be performed. Therefore we must induce the structure of the repertoire, as well as optimal parameters for the model, from user observations alone. Since we are using a CDHMM, estimating the structure of the model means determining the topology of the underlying graph. Simple graph-based models imply as few nodes and edges as possible. Our PRP algorithm begins by constructing a maximal-node graph by assigning one observation to each node. The algorithm then repeatedly merges *similar* nodes until a *compact* graph remains. This graph is then converted to a CDHMM to estimate future user observations. Let $\boldsymbol{\mathcal{A}} = v_i \cup v_j$ be the union of the observation multisets from node $v_i$ and node $v_j$. Let $\boldsymbol{\mu} = \langle \boldsymbol{\mathcal{A}} \rangle$ be the sample mean of $\boldsymbol{\mathcal{A}}$. Our measure of similarity is

$$d_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mu}) \quad \triangleq \quad \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{A}}} (\boldsymbol{x} - \boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{C} (\boldsymbol{x} - \boldsymbol{\mu}), \qquad (1)$$

where the symmetric PD precision matrix $\boldsymbol{C}$ provides a notion of the *a priori* expectation of node variance. The nodes $v_i$ and $v_j$ are considered similar if $d_{\boldsymbol{C}}(\boldsymbol{\mathcal{A}}, \boldsymbol{\mu}) \leq \epsilon$, where the similarity threshold is $\epsilon \in [0, \infty)$. This parametric concept of similarity accounts for the intuitive notion that similarity is a continuum. For example, consider the objects in Figure 1. One possible set of similar objects is the two squares. If we make the concept of similarity looser then the set of similar objects could be all shapes with a vertical pattern. In PRP, the notion of similarity is also a continuum. The nodes in the maximal graph in Figure 2(a) are merged using a strict and a loose definition of similarity, Figure 2(b) and Figure 2(c) respectively. The resulting graphs are different since the definition of similarity was different.

The parameter $\epsilon$ does not have an obvious real-world analogue that gives the user insight into the operation of

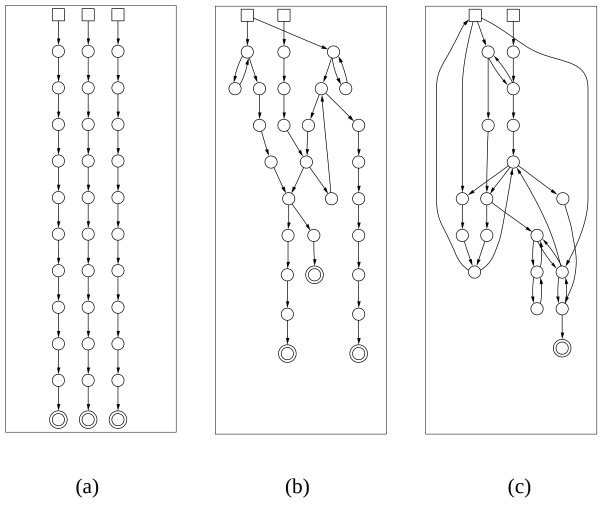

Fig. 2. The maximal-node graph is given in Figure 2(a). Results of state merging using strict and loose similarity definitions are shown in Figure 2(b) and Figure 2(c) respectively.

the PRP algorithm. Assuming that users make zero-mean Gaussian errors about the target waypoint then the Mahalanobis distance follows is chi-squared random variable with $g$ degrees of freedom, where $g$ is the dimension of $\boldsymbol{x}$. In real-world terms, we hypothesize that observations are generated with probability at least $1 - \delta$. Using this formulation, we compute $\epsilon$ from $\delta$ by evaluating the cdf of the chi-square distribution,

$$\epsilon \quad \in \quad \left\{ w \,\middle|\, \delta = 1 - \frac{\gamma(\frac{g}{2}, \frac{w}{2})}{\gamma(\frac{g}{2}, 0)} \right\},$$

where $\gamma(\cdot, \cdot)$ is the incomplete gamma function. The graph is then converted in a CDHMM by estimated probability density functions based on the observations assigned to each node. We denote the current task as $\boldsymbol{X}^{c}_{0:n\text{–}1} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n\text{–}1}\}$. While the user is performing a new task, the CDHMM, $R$, computes the maximum likelihood prediction of the next waypoint,

$$\widehat{\boldsymbol{x}}^{*}_{n} \quad = \quad \arg\max_{\boldsymbol{x}_n} \mathrm{p}\big(\boldsymbol{x}_n | \boldsymbol{X}^{c}_{0:n\text{–}1}, R\big) \qquad (2)$$

However, any prediction criterion should also be accompanied with a *confidence* of the prediction. Otherwise, there is no way to convey the relative quality of the prediction. Using recent observations from the current task, we compute the divergence of the CDHMM from complete internal uncertainty. Let $c_n$ be the random variable denoting the current state of the CDHMM at time $n$. In our PRP system confidence is defined as

$$\phi_n \quad \triangleq \quad \frac{\mathcal{D}_{\mathrm{KL}}(c_n \| \frac{1}{|\mathcal{Q}|})}{\log_2 |\mathcal{Q}|} \quad = \quad 1 - \frac{H(c_n)}{\log_2 |\mathcal{Q}|}, \qquad (3)$$

where $|\mathcal{Q}|$ is the number of states in the CDHMM, $H(\cdot)$ is entropy, and $\mathcal{D}_{\mathrm{KL}}(\cdot \parallel \cdot)$ is the Kullback-Leibler

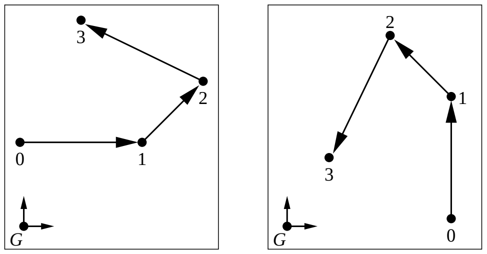

Fig. 3. Though different in a global frame, $G$, the relative movement of these patterns are the same.

divergence. It is easy to verify that the confidence is $\phi_n \in [0, 1]$ for any state distribution and any $|\mathcal{Q}|$. When the confidence $\phi_n$ is greater than some predetermined threshold then the corresponding prediction $\widehat{\boldsymbol{x}}_n^*$ is suggested to the user. Otherwise the prediction is discarded.

In three-dimensional space, 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 [2]. Our use of the prediction estimator (Equation 2) influences our choice of orientation representation. While each representation 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 the current task be represented by $\boldsymbol{W} = \{{}^G\boldsymbol{w}_0, {}^G\boldsymbol{w}_1, \ldots, {}^G\boldsymbol{w}_N\}$, where the waypoint ${}^G\boldsymbol{w}_n$ maps the global frame $G$ to the frame of the $n$th waypoint. It is simple to show that the relative-movement representation $\widetilde{\boldsymbol{W}} = \{{}^0\boldsymbol{w}_1, {}^1\boldsymbol{w}_2, \ldots, {}^{N-1}\boldsymbol{w}_N\}$ is independent of an initial rotation and translation, *cf.* Figure 3. In other words, using the relative-movement representation allows predictions of patterns independent of their location and orientation in the workspace. However, using relative-movement information requires that the orientation of the robot end-effector be extremely accurate. Small errors in orientation can result in large position errors. However, the ability to perform rotation- and translation-independent prediction outweighs the potential loss in accuracy.

## III. EXPERIMENTAL RESULTS

The five tasks analyzed in this paper were created using an offline-programming environment (Figure 4). The programs have between $252$ to $1899$ waypoints with $16$ to $196$ subroutines. Collectively, these five programs took over $70$ work days to complete. Thus even a small reduction in programming time would result in days saved. The programs were created to automate arc-welding production at several factories in Sweden, not for experimenting

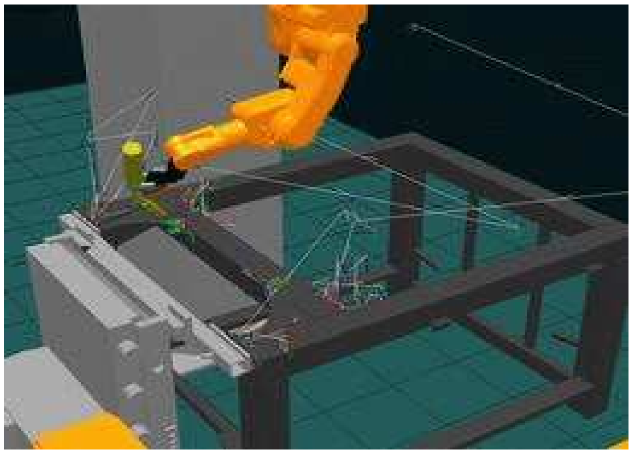

Fig. 4. Screen shot from the offline package showing the workspace and waypoints of a program with $18$ subroutines.

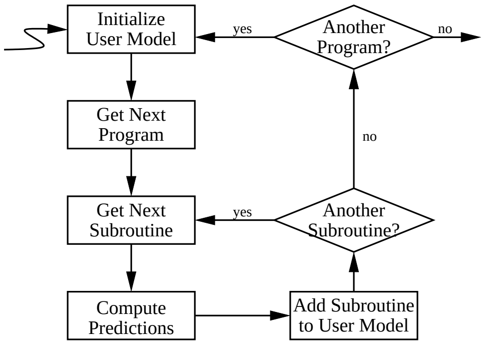

Fig. 5. Flow chart for computing predictions.

with PRP. Each program was designed to weld a different type of product, from round tables to bed frames. The underlying PRP algorithm makes no restriction that the programs have the same number of waypoints or perform the same physical process.

To compute waypoint predictions, we segment the programs along the subroutines contained in each program. We emulate the user programming the task by feeding the waypoints into the PRP system in a serial fashion. The CDHMM computes a prediction of the next waypoint, based on previous waypoints in the subroutine. To determine the accuracy of the prediction, we compute the Euclidean and angular error of the prediction with respect to the next waypoint. At the end of each subroutine we assimilate it into the CDHMM, building an estimate of the user repertoire for that program. After predicting the final waypoint in a program, we initialize the CDHMM *tabula rasa* and start the prediction of the next program. A flow chart of this procedure is given in Figure 5.

The definition of similarity, Equation 1, incorporates a symmetric PD precision matrix, $\boldsymbol{C}$, that represents the variance of human error when defining a waypoint. We

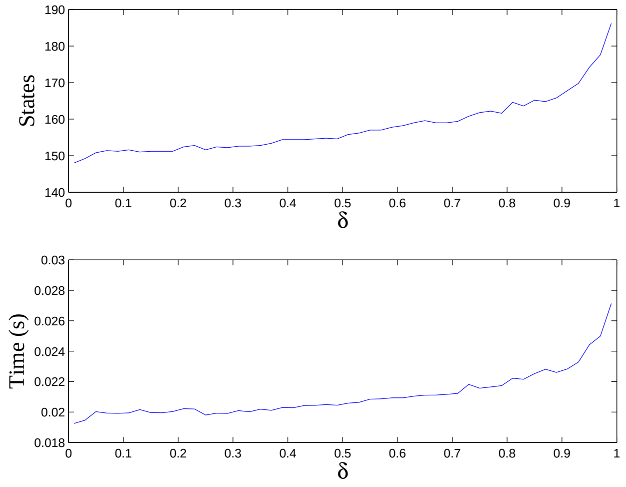

Fig. 6. Model complexity as a function of $\delta$.

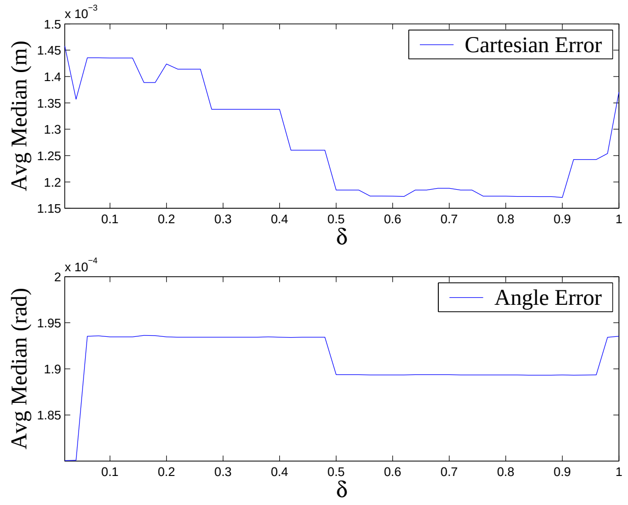

Fig. 7. Median Cartesian error and angle error as a function of $\delta$.

computed the covariance matrix by having users repeatedly move a robot to a point in space and analyzing the residual error.[1] Robotic arc-welding typically requires approximately $1$ millimeter of Cartesian accuracy and about $0.1$ radians of angular accuracy. The results presented in this work will use these physical tolerance constraints as a threshold for determining a "useful prediction". We are more interested in the typical prediction error, which is better conveyed by the median. 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.

### A. *Performance as a Function of Model Complexity*

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 programs. The held-out program is meant as a sort of test set, but since data are so sparse it will be difficult to draw any statistically significant conclusions based on this four-program training set and one-program test set. The parameter central to the complexity of the user model is the implicit probability of rejection, $\delta \in (0, 1]$, which is a function of the similarity threshold $\epsilon \in [0, \infty)$. Increasing the rejection likelihood, $\delta \to 1$, induces a more complex CDHMM since the definition of similarity is more strict ($\epsilon \to 0$). Conversely, decreasing the rejection likelihood, $\delta \to 0$, induces a simpler CDHMM. In Figure 6, we plot two measures of model complexity: number of states in the CDHMM and running time. As expected, the number of states in the CDHMM increases as $\delta$ increases. Also, the running time increases as the model becomes more complex. However, the time shown in Figure 6 is the average time per waypoint taken to estimate a model *and* compute a prediction. This average time, $\sim$25ms, is more than adequate for real-time use in a robot-programming environment.

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

Indirectly, the parameter $\delta$ also determines the accuracy of predictions by controlling the complexity of the user model. In Figure 7, we plot the average median error on the four-program training set as a function of $\delta$. For all values of $\delta$, the angle error is extremely small, less than $200$ micro-radians. This is because the robot tends to change orientation in a fairly routine 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 variable. In PRP, as in many machine-learning algorithms, "everything should be made as simple as possible, but not simpler" and the *correct* value of $\delta$ depends on the task at hand. The Cartesian error bottoms out on the interval $\delta \in (0.5, 0.9)$, implying that a value of $\delta \approx 0.7$ induces the right amount of complexity on the training set.

### B. *Performance as a Function of Confidence Threshold*

Generating a prediction for every waypoint is not desirable. Ideally, the PRP system would only suggest the most accurate predictions to the user. However, the target waypoint is not known at the time of the prediction, so we must find an alternative criterion correlated with accuracy. This is where the confidence threshold comes into play.

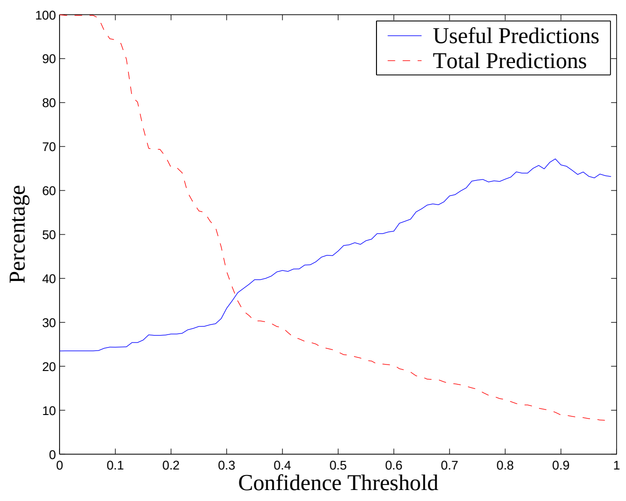

Fig. 8. Percentage of predictions and useful predictions as a function of the confidence threshold.

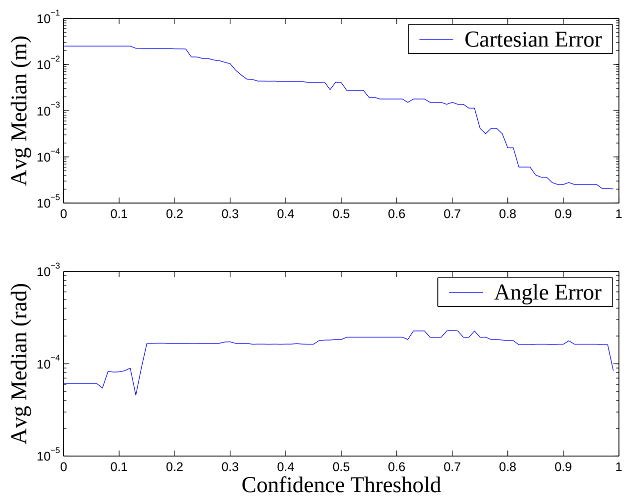

Fig. 9. Median Cartesian error and angle error as a function of the confidence threshold.

If the confidence of the prediction, $\phi_n$ (Equation 3), is below a threshold then the prediction is discarded. The assumption is that higher-confidence predictions are more useful than lower-confidence predictions. However, as the confidence threshold increases, inevitably, there will be useful predictions discarded since their confidence is insufficient. Using the training set, we would like to determine a confidence threshold that strikes a balance between quality and quantity. In Figure 8, we fix the value of $\delta = 0.7$ and plot the percentage of useful predictions as a function of the confidence threshold. 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 9, we fix the value of $\delta = 0.7$ and plot the average median error as a function of the confidence threshold. Once again, for all confidence thresholds, the angle error is extremely small, between $60$ and $200$ micro-radians. However, the Cartesian error improves as the confidence threshold increases, until it bottoms out about $30$ microns around $\phi_n \geq 0.8$. This implies that, on the training set, higher-confidence predictions tend to be more accurate. The average median error on the training set for $\phi_n \geq 0.8$ is well below the physical tolerance required by arc welding.

## C. *Temporal Performance*

In this subsection we use parameters based on results from the training set, $\delta = 0.7$ and $\phi_n \geq 0.7$, to analyze the performance on the hold-out program. The CDHMM is initialized *tabula rasa* and assimilates user behavior incrementally. From this perspective, we expect the prediction error to start high and decrease as more information becomes available. In Figure 10, we plot the median error on the hold-out program as the programmer creates waypoints. After assimilating about $900$ waypoints the median Cartesian error stabilizes around $200$ microns. Likewise, the median angle error stabilizes around $20$ micro-radians after about $300$ waypoints. Both of these values are well below the physical tolerance of $1$mm and $0.1$ radians required by arc welding. This asymptotic-like behavior of the algorithm suggests that the estimate of the user repertoire improves until acquiring sufficient information about the task. The PRP algorithm made predictions for about $25\%$ of the waypoints on the hold-out program. This program originally took over eight work weeks to complete using an offline-programming package. The programming time saved by predicting waypoints could be several days. 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 is a result 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 programming-time savings, making robot-programming environments less cumbersome.

## IV. RELATED WORK

The seminal paper on HMM applications [8] describes training algorithms for fixed-topology HMMs. In PRP the general structure of the model is unknown *a priori* and must be induced by observing user behavior. We therefore consider the *much* more difficult problem of discovering both the structure of the model and its optimal parameters. There are results that suggest optimal training of general HMMs is not tractable [1]. To deliver more optimistic results, researchers have focused on special subclasses of

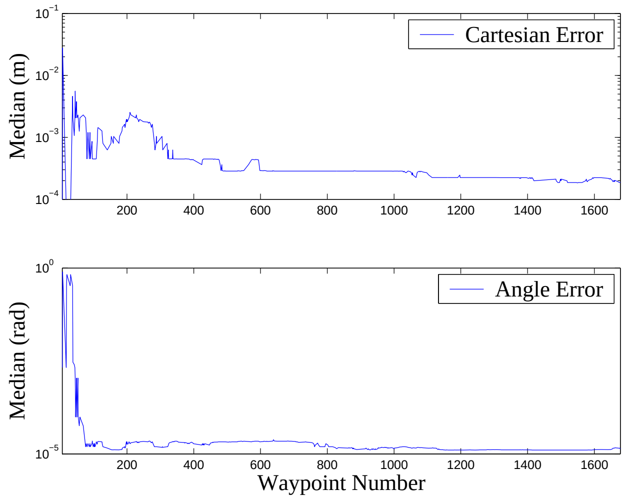

Fig. 10. Median Cartesian error as a function of the number of waypoints assimilated into the CDHMM.

HMMs [9], prior topology distributions [11], and heuristic methods [10].

While few attempts have been made at reducing robot-programming time by prediction, the underpinnings of PRP have been explored extensively. Time-series analysis is the umbrella term for predicting future observations of a stochastic source given prior observations. Applying these ideas to the Human-Computer Interaction (HCI) domain is called Learning By Observation (LBO) and many similar terms. The best-known example of LBO is the neural-network system (ALVINN) that learns to drive a car by observing human drivers [7]. In recent years, LBO research has investigated robotic task decomposition [4], nonlinear trajectory emulation [5], and human-motion synthesis [6].

## V. CONCLUSIONS

The results from Section III show that a *tabula rasa* PRP system can predict the waypoints of complex, real-world programs with great accuracy. On both the training set and hold-out program, the median prediction error was well below the tolerance required by arc welding. For each program tested, the PRP system generated predictions for about $10\%$ to $20\%$ of the possible waypoints, and the vast majority of the predictions were useful. Collectively, the programs analyzed took over $70$ work days to complete using an offline-programming package. By allowing the PRP system to define waypoints for the user, the programming time could be reduced by several days.

In the future, we will experiment with non-blank-slate approaches, especially with versions of a robot program. For example, products are updated regularly. We would like to experiment in determining how helpful the previous product is in determining the waypoints of the current product. Also, we would like to employ semi-supervised learning techniques to improve the PRP system.

## VI. ACKNOWLEDGMENTS

We would like to thank Per-Ulrik Eriksson for providing us with the robot programs and Mike Seltzer for very helpful discussion. We used GraphViz (`dotty`) from AT&T Research to display graph topologies. We would like to thank the Intel Corporation for providing the computing hardware. This work was sponsored by ABB Corporate Research.

## VII. REFERENCES

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

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

[3] K. R. Dixon, M. Strand, and P. K. Khosla. Predictive robot programming. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

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

[5] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning rhythmic movements by demonstration using non-linear oscillators. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[6] O. C. Jenkins and M. J. Matarić. Deriving action and behavior primitives from human motion capture data. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[7] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. PhD thesis, Carnegie Mellon University, 1994.

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

[9] 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.

[10] R. Singh, B. Raj, and R. M. Stern. Automatic generation of sub-word units for speech recognition systems. *IEEE Transactions on Speech and Audio Processing*, 10(2):89–99, 2002.

[11] A. Stolcke and S. Omohundro. Inducing probabilistic grammars by Bayesian model merging. In *International Conference on Grammatical Inference*, 1994.