2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

A HMM-based Approach to Learning Probability Models of Programming Strategies for Industrial Robots

Rebecca Hollmann, Arne Rost, Martin Hägele, and Alexander Verl

Abstract—**The integration of industrial robot systems into the manufacturing environments of small and medium sized enterprises is a key requirement to guarantee competitiveness and productivity. Due to the still complex and time-consuming procedure of robot path definition, novel programming strategies are needed, converting the robotic system into a flexible coworker that actively supports its operator.**

In this paper, a learning-from-demonstration strategy based on Hidden Markov Models is presented, which permits the robot system to adapt to user- as well as process-specific features. To evaluate the suitability of this approach for small-lot production, the learning strategy has been implemented for an arc welding robot and has been evaluated on-site at a medium sized metal-working company.

I. Introduction

Today's industrial robot systems perform at high accuracy and efficiency levels, guaranteeing a constant product quality. In order to stay competitive on their market, more and more *small and medium sized enterprises* (SMEs) consider the use of industrial robots to partly automate their production. In the end, a significant number of SMEs decide against the acquisition of a robot system due to the time-consuming and rather complex process of programming, which is still required in conventional robots [1]. SMEs typically deal with small lot sizes in the range of 50-200 pieces, so reprogramming the robot will be necessary frequently. Furthermore, the workers in producing workshops might be highly experienced process experts, but are usually not trained in operating complex technical systems.

The most suitable robot system fitting into the given context would be instructed similarly to a new apprentice boy: for the first time you explain to him a task, you will have to mention every single detail before you leave him alone with the task. By the next time, he will already remember the special cases seen in the first task. Over time, the apprentice will gain enough experience to operate the task on his own after a short introduction to the workpiece.

As discussed in [2], *Programming by Demonstration* (PbD) is a promising approach to fast and intuitive robot programming. The key idea of PbD is that the operator shows and explains to the robot its next task without the need to have explicit programming knowledge. This technique is well-known and can be defined in a number of different ways (for a full classification see [3]). There exist lots of interesting approaches using different learning strategies to model human skills as in [4], the building of state/action memory maps to model observed actions as in [5] or learning household tasks decomposed into a sequence of actions from human demonstrations as in [6] and [7]. Most of these studies are conducted at a rather theoretical basis, using simulation technologies or laboratory test platforms for evaluation purposes.

Human-robot interaction for industrial robots in manufacturing environments, in contrast, forms an area of research that is only now emerging. The current developments concentrate on the design of interfaces for multi-modal human-robot communication specialized for online programming as in [8] and [9] or on the intuitive definition of workspace constraints to cope with the mismatch between virtual and real worlds for offline path-planning as in [10]. Although these methods are intuitive to learn and significantly faster than conventional methods, the robot systems keep performing at a constant level no matter how many demonstrations they have seen, as there are no suitable interfaces able to extract relevant knowledge in order to generalize over the given task.

In this paper an approach to learn programming strategies as well as process knowledge from human demonstrations will be presented. The approach is based on *Hidden Markov Models* (HMM) and aims at actively supporting the operator in the programming process. As an application example, an arc welding robot equipped with an intuitive programming environment based on PbD is used, which will be shortly introduced in section II. Section III and IV present the general modeling approach as well as the calculation of observable environment parameters, whereas the following section details the concrete procedure used to generate predictions. The learning strategy has been evaluated in an authentic manufacturing environment. The results are summarized in section VI, followed by a conclusion and outlook to future work.

This work has been partly funded by the European Commission`s Sixth Framework Programme under grant no. 011838 as part of the Integrated Project *SMErobot*™

R. Hollmann is with the Fraunhofer Institute Manufacturing Engineering and Automation,70569 Stuttgart, Germany (phone: +49-711- 970-1095; fax: +49-711-970-1008; e-mail: hollmann@ipa.fraunhofer.de).

A. Rost is with the Fraunhofer Institute Manufacturing Engineering and Automation, 70569 Stuttgart, Germany (e-mail: rost@ipa.fraunhofer.de).

M. Hägele is with the Fraunhofer Institute Manufacturing Engineering and Automation, 70569 Stuttgart, Germany (e-mail: haegele@ipa.fraunhofer.de).

A.Verl is with the Fraunhofer Institute Manufacturing Engineering and Automation, 70569 Stuttgart, Germany (e-mail: verl@ipa.fraunhofer.de).

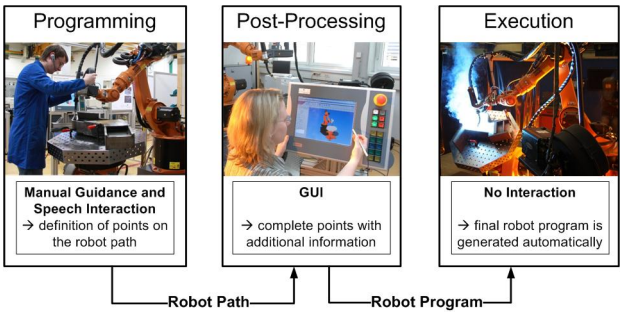

Fig. 1. Programming procedure using the intuitive programming environment for industrial robots developed at Fraunhofer IPA.

II. Programming Environment

A PbD strategy has been implemented by way of example for a KUKA KR16 industrial robot equipped with a JR3 *force-torque sensor* (FTS), a handle, a welding device and a two-axis positioner. The general programming procedure is shown in Fig. 1.

Programming, i.e. the definition of a sequence of points on the robot path is conducted using manual guidance and speech interaction. Manual guidance is realized using admittance algorithms as described in [11]. For the robot, two different movement modes are implemented which differ in the number of *degrees of freedom* (DoF) that are activated. In *3 DoF mode* only translational movements are activated. The *6 DoF mode* allows the user to change the tool orientation as well as to guide the robot arm translationally as in *3 DoF mode*. Additionally, a *table mode* has been implemented to change the orientation of the workpiece fixed on the positioner via the same interface. The speech interface, which is described in detail in [12], is used to activate commands from the set

$$Q_C = \{Move_R, Move_T, Switch_{ROT}, Speed_F, Speed_S\} \quad (1)$$

with the following practical impact:

- $Move_R$: switch to robot movement mode
- $Move_T$: switch to table movement mode
- $Switch_{Rot}$: enable / disable 6DOF robot movement
- $Speed_F$: increase the scaling factor of the FTS-values resulting in faster robot movement
- $Speed_S$: decrease the scaling factor of the FTS-values resulting in slower robot movement.

At the end of the programming step, one point has the form [X Y Z A B C $curr_{Speed}$ $curr_{Dist}$], where X, Y, Z, A, B, C define the position and orientation of the welding torch. The variables $curr_{Speed}$ and $curr_{Dist}$ monitor the speed of the robot while being guided to the current position and the distance between tool and workpiece respectively. An exact definition follows in section IV.

During the post-processing step, the programmed robot path is loaded to a *graphical user interface* (GUI) and visualized in a 3D-view. Here each point of the robot path can be complemented with additional information taking the form [X Y Z A B C $curr_{Speed}$ $curr_{Dist}$ $final_{Speed}$ welding]. The variable $final_{Speed}$ sets the movement speed for the final robot program. The welding variable defines the functional role of the according point for the welding process and takes one of the values from set

$$Q_P = \{ARC_{ON}, ARC_{OFF}, ARC_{Switch}, LIN\} \quad (2)$$

which correspond to the following types of movement commands in the final robot program:

- LIN: linear movement in free space with the welding torch switched off.
- ARC_{ON}: starting point of the welding seam, including necessary preparation information for the turning-on of the welding torch.
- ARC_{OFF}: last point of the welding seam, including all relevant process parameters.
- ARC_{SWITCH}: a point on the welding seam, including changes in the movement/process parameters.

Instead of post-processing a point, it can also be deleted if it has been programmed accidentally and is not needed in the final program.

During the last step the post-processed path is automatically converted to an executable robot program using a template header and footer and translating each point into the according robot command set. Further details about the programming environment can be found in [13].

III. Modeling the Learning Problem

Programming as described in section II contains repetitive and therefore predictable operations: a robot programmer will develop his individual procedural method for path definition. Likewise, a work process like welding requires a defined order of actions to produce high quality results. To model and predict the sequential data sets, i.e. points in a robot path, HMMs are considered a promising methodology [14].

A. Model Training

The proposed model for the learning problem is based on all data seen by the system at usage time. It starts with zero information and is updated every time a new robot path has been programmed and saved. To ensure a high confidence of the model, the learning strategy is fully supervised by comparing the data in the preliminary robot path to the command sets of the final robot program (see dashed arrows in Fig. 2).

To avoid unexpected actions, the system requires the user to accept or decline all predictions generated by a trained model. The information obtained from this user feedback can be used to reinforce the model parameters as indicated by the dotted arrows in Fig. 2.

B. Formal description

Following the HMM definition given in [15] it is necessary to define a set of internal states $X \in Q$ as well as a set of emissions $Y \in E$. In the present case, an internal state describes the functional role of a single point within a programmed robot path. Every internal state produces an emission depending on its current context. An emission in that case describes the collectivity of observations associated

with the individual program point, whereas the context of the internal state means the characteristics of its predecessors and their emissions. The final goal of this approach is the stepwise prediction of the internal states in a robot path from the observations, i.e. the emissions. Precise examples will be given in the next sections.

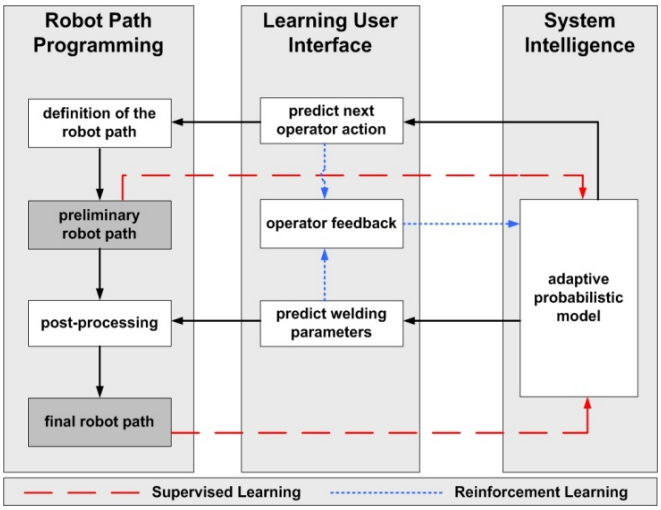

Fig. 2. Overview of the proposed adaptive interface. The dashed arrows mark the influence of supervised learning strategies, the dotted arrows mark the influence of reinforcement learning.

To train the desired model, transition probabilities p_{kl} and emission probabilities $e_k(b)$ have to be calculated. At the creation of a new model, it will contain no valuable information, so

$$p_{kl} = e_k(b) = \text{INIT\_VAL} \; \forall \; k, l \; \in Q, b \; \in E \tag{3}$$

where INIT_VAL represents some small initial value (here set to 0.1) in order to avoid numerical problems. With every robot path seen, the probability matrices will be updated according to the following equations:

$$p_{kl} = P(X(t+1) = l \,|X(t) = k) \; with \; k, l \; \in Q. \tag{4}$$

$$e_k(b) = P(Y(t) = b \,|X(t) = k) \;\; with \; k \in Q, b \; \in E. \tag{5}$$

IV. Observations

The robot system used in the presented approach is equipped with a force torque sensor as well as a realtime-interface, which allows the monitoring of the current set-positions. No further sensors are integrated into the system in order to keep it cost-efficient as well as suitable for rough environment conditions. Based on the available information the following observations are obtained:

A. Velocity calculation

Due to the used control strategy the movement velocity of the robot during manual guidance scales with the applied forces and torques. During every control cycle the forces and torques measured by the sensor are summed up and stored in a data vector. With every added program point, the mean value of all sums obtained since its predecessor-point, will be stored as the associated speed parameter

$$curr_{speed} = \frac{\sum_{i=1}^{n_t} (f_x(i) + f_y(i) + f_z(i) + m_x(i) + m_y(i) + m_z(i))}{n_t} \tag{6}$$

where n_t is the number of control cycles between the predecessor-point at time $t-1$ and the novel point at time t.

B. Distance calculation

Due to the reasons discussed above, the exact position and shape of the workpiece are unknown to the robot system. Nevertheless, a rough estimation about the relative distance between tool and workpiece can be obtained from the movements already performed during the current programming cycle. It can be assumed that the workpiece is fixed on some known kind of attachment (in this case the two-axis positioner), so for every programmed point it is monitored if the tool has moved closer towards the attachment or not:

$$curr_{dist} = dist_{TA}(t) - \; dist_{TA}(t-1) \tag{7}$$

where $dist_{TA}$ describes the distance between the robot tool center point and the center of the attachment at time t.

C. Parameter Scaling

The absolute values obtained in (6) and (7) will vary not only depending on the workpiece and the operator, but also from day to day. To extract relevant and comparable information from the absolute parameters, they are scaled to the range observed during the current programming cycle:

$$curr_{rel}(t) = \frac{curr_{abs}(t) - \min curr_{abs}}{range}. \tag{8}$$

$$\text{with } range = \max curr_{abs} - \min curr_{abs} \tag{9}$$

According to their relative values, the parameters are classified into discrete categories:

$$vel = \begin{cases} 0 \;\; if \;\; curr_{rel} < \; -0.15 \\ 1 \;\; if \;\; -0.15 < curr_{rel} < 0.15 \\ 2 \;\; if \;\; 0.15 < curr_{rel} \end{cases} \tag{10}$$

$$dist = \begin{cases} 0 \;\; if \;\; curr_{rel} < \; -0.15 \\ 1 \;\; if \;\; -0.15 < curr_{rel} < -0.01 \\ 2 \;\; if \;\; -0.01 < curr_{rel} < 0.01 \\ 3 \;\; if \;\; 0.01 < curr_{rel} < 0.15 \\ 4 \;\; if \;\; 0.15 < curr_{rel} \end{cases} \tag{11}$$

The observation space used in the following sections can be summarized as

$$E = \{vel, dist\}. \tag{12}$$

V. Generating Predictions

Two different learning problems are addressed in the presented approach: on the one hand, time-consuming steps concerning the path post-processing are sought to be eliminated. To achieve this, a model describing the work process under consideration is needed. On the other hand, the path definition procedure is assumed to be accelerated by continuously updating a model of the programming strategies of the known operators.

A. Process Model

The process model determines the functional role of every program point in order to predict the process parameters and therefore inherit the task of post-processing step by step from the operator. The internal states for arc welding equal the set Q_P given in (2).

As can be reasoned from the welding example, the functional role of many program points already provides information about the order in which the internal states typically occur. The process model stepwise adapts to the specific relationship between the internal states and how they correspond to the observation parameters. The structure of the process model is shown in Fig. 3:

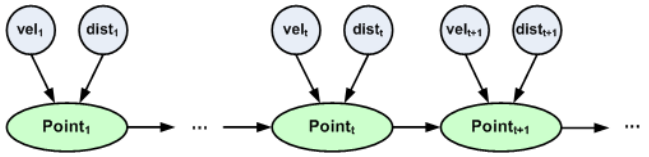

Fig. 3. Overview of the process model architecture.

The model training is conducted according to the formulas given in (3) - (5). As the predictions from the process model are used for post-processing, the decoding can be conducted offline on the complete robot path permitting the application of the viterbi algorithm [16]: for every program point with a given observation $Y_t \in E$, and for every internal state $l \in Q_P$ the viterbi variables are calculated as

$$v_l(t) = e_l(Y(t)) * \max(v_k(t-1) * p_{kl})_{k \in Q_P} \quad (13)$$

with p_{kl} the transition probability according to (4) and $e_k(Y)$ the emission probability according to (5). For every viterbi variable a pointer is set to the most probable predecessor of the associated internal state:

$$ptr_t(l) = \arg\max(v_k(t-1) * p_{kl})_{k \in Q_p} \quad (14)$$

Having stored all viterbi variables and their associated pointers, the most probable path of internal states is determined starting from the highest viterbi value obtained for the last program point in the sequence and successively following the pointers to the most probable predecessors in a backward manner:

$$\hat{X}_{t-1} = ptr_t(\hat{X}_t) \quad (15)$$

with $\hat{X}_t$ the most probable internal state at time t.

B. Command Model

The command model represents the programming strategy of the known users, i.e. the typical sequence of actions (as defined in Q_C, see (1)) carried out during a programming task.

The command model supports the operator by suggesting the next action necessary for an efficient programming procedure. In contrast to the process model, the command model is used online with only the first part of the robot path available at the time of predicting. Another challenge this set-up presents is to find not only the appropriate succeeding action, but also the most suitable time-window for the suggestion. To accommodate this problem specification, a cascade of two individual HMMs is proposed (depicted in Fig. 4).

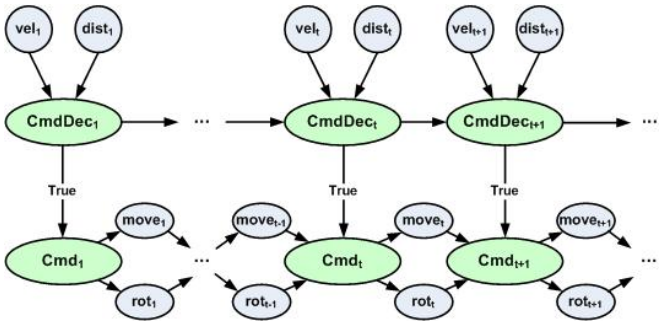

Fig. 4. Overview of the command model architecture.

A decision model is constantly monitoring the observations. Considering the observation as well as the time elapsed since the last command has been set, this model first decides whether it is time for a suggestion at all. Formally speaking

$$Q_{Cdec} \in \{true, false\} \quad (16)$$

The observation space corresponds to the one already defined in (12). Model training is conducted as defined in (3) – (5) with the difference that a higher HMM order is needed to take into account the time elapsed since the last command. Hence (4) is reformulated to

$$p_{klmn} = P(X_{t+1} = n | X_t = m,\ X_{t-1} = l,\ X_{t-2} = k) \quad with\ k, l, m, n \in Q_{Cdec} \quad (17)$$

As the predictions are required in an online manner, the viterbi algorithm is not applicable in this case. Since a binary result is adequate, a simple threshold is defined as

$$thresh = \left(\frac{100}{|Q_{Cdec}|} + \frac{100}{|E|}\right) * 0.5 \quad (18)$$

permitting a binary classification according to

$$\hat{X}_t = \begin{cases} true\ if\ \frac{p_{klmn} + e_k(b)}{2} * rand > thresh \\ false\ if\ \frac{p_{klmn} + e_k(b)}{2} * rand < thresh \end{cases} \quad (19)$$

If $\hat{X}_t$ is *false* the model ignores the current step, staying passive until the next parameter set is available. If, however, the decision is *true* the underlying command model is activated.

Some of the commands defined as internal states in (1) change the overall system status in a manner that makes specific commands useless as long as the state is not changed back. For example, it is useless to switch to robot

movements if the system is already in robot movement mode as well as a two-axis positioner will never be able to perform 6 DOF movements, whereas the movement speed can be increased several succeeding times until a maximum value is reached. To represent all such coherences within the model, it would be necessary to adapt the order of the HMM to the number of points already programmed. To avoid exaggerated computational effort, an additional observation space is introduced for keeping track of relevant changes to the system status:

$$E_C = \{move(robot||table), rot(3DOF||6DOF)\} \quad (20)$$

The model training is generally conducted according to (3)-(5) with the following change in the calculation of the emission probabilities:

$$e_k(b,c) = P(Y(t) = b, Y_C(t) = c \,|X(t) = k) \\ with\ k \in Q_C, b \in E, c \in E_C \quad (21)$$

For the prediction of internal states, the viterbi variables as defined in (13) are calculated using the definition in (21) to represent the emission probability term. As the predictions are needed online, the viterbi variables are used directly to obtain the most probable next state:

$$\hat{X}_t = \arg max(v_k(t) * p_{kl})_{k \in Q_C} \quad (22)$$

The estimated internal state is communicated to the user via the speech interface. The user is asked to accept or decline the suggestion. If he accepts it, the command takes effect immediately. Whereas in the latter case, the model is updated in a reinforcement manner and the command is discarded. A more detailed description of the user interface can be found in [17].

VI. Results

The proposed learning strategy is evaluated using a data set consisting of 15 robot programs defined with the programming environment described in section II. The programs have been recorded by five different operators in a manufacturing workshop on original workpieces. The data hence represents a rather hard benchmarking set. The robot programs will be referred to as trials in the following.

The preferred evaluation method used in this section is a leave-one-out cross validation. Applying cross validation to the whole data set, i.e. using 14 trials for model training and the remaining one for testing, a mean accuracy of 85.02% of internal states correctly predicted is obtained for the process model and 75.97% for the command model respectively. For the performance on the individual trials see Fig. 5.

A. Evaluation of the Process Model

Besides the overall performance of a model trained on the complete data set, another interesting measure is how fast the model learns from scratch, when seeing new data similar to the known training basis or even data containing new features never seen before. As the ARC_{SWITCH} command is only present in 8 out of the 15 recorded trials, a splitting of the data set into two subsets (one with ARC_{SWITCH}, one without it) can be used to visualize this learning process.

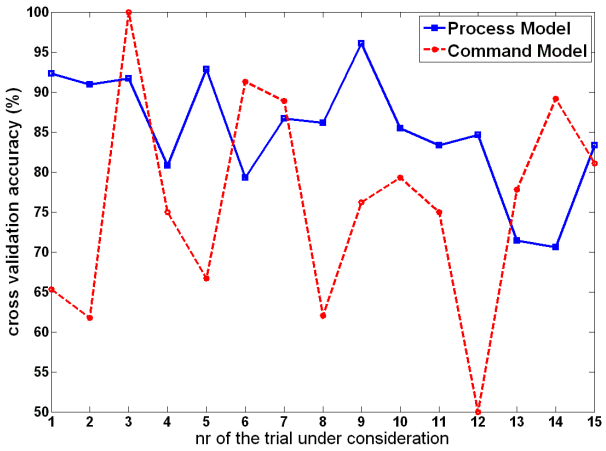

Fig. 5. Performance of the process and the command model on leave-one-out cross validation for the individual trials.

The solid line in Fig. 6 shows the development of a new process model without the use of the ARC_{SWITCH} command. On the X-axis the number of trials used for model training is given. The accuracy drawn for the training data sets of increasing size is calculated as the mean value of the k-fold cross validation on the subset lacking the ARC_{SWITCH} command. The figure describes nicely the learning progress, i.e. the accuracy increasing by about 10% during the first six trials seen. Additionally, the standard deviation over the single evaluation cycles decreases with every additional trial seen, showing that the model is stabilized. The dashed line in Fig. 6 shows the progress of learning a new internal state, when already trained on different data. For training, the whole subset lacking the ARC_{SWITCH} command has been used in each case. The number of new trials containing the unseen internal state that are added one by one to the training data set is given on the X-axis. The evaluation is based on the remaining robot paths containing the ARC_{SWITCH} command that are currently not used for training purposes.

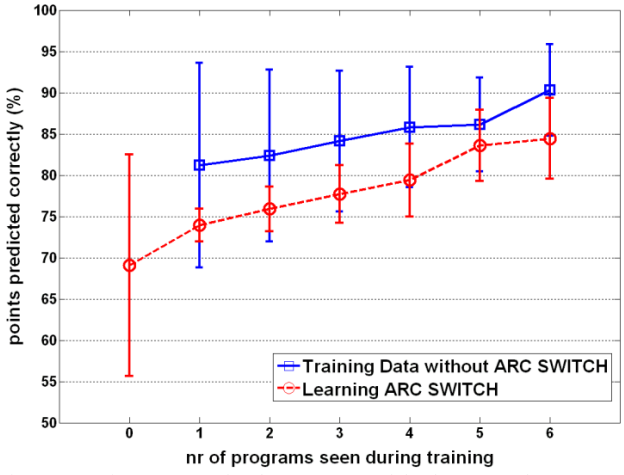

Fig. 6. Learning progress for the process model. Shown are the mean value and standard deviation for training data sets of different sizes.

B. Evaluation of the Command Model

As the command model consists of two cascaded HMMs, i.e. a decision model and the actual command model, the evaluation has to be examined separately for the two models. The command model can be evaluated in a similar way as described for the process model, whereas the decision model performance highly depends on the individual preferences of the operator. Some users appreciate frequent suggestions which can be easily rejected using the speech interface. Others might prefer few suggestions with a high significance that can be complemented with traditional user input. To meet the preferences of the individual operator, the threshold for the decision model as given in (18) can be adapted by a scaling factor.

The command model has been evaluated by leave-one-out cross validation as described for the process model above with at an average of 9% worse performance compared to the process model (see Fig. 5). This difference can be explained by two factors: first, the sequence given by the nature of the welding process is more rigid than a user's programming strategy. Aside from that, the predictions of the command model are needed in an online manner, thus eliminating the backward step of the viterbi algorithm. Nevertheless, the acceptance of incorrect predictions from the command model is relatively high due to the fact that the user is required to react to every prediction anyway. It is expected that the performance of the command model increases significantly if it is trained by only one user, which will be the standard case in the final application.

VII. CONCLUSION

In this study, a systematic approach for building an adaptive user interface to ease the programming effort for industrial robots has been presented using a concrete example, which was set up and evaluated on-site due to the application scenario. The given example can be understood as a proof of concept study as it consists of rather small models implementing a limited number of internal states.

The approach is scalable concerning the number of internal states as well as the observation of the environment. An interesting extension in the post-processing step would be the integration of workpiece data in order to gather more exact distance data and such improve the accuracy of the programmed path. Such data could be provided e.g. by the integration of CAD models of the workpiece or from an optical sensor. The absence of such data in the current programming environment is caused by the clear specifications of the underlying application scenario. In many SMEs CAD data are not available for all kinds of workpieces and optical sensors are either susceptible to the rough environment conditions at the productional workshop or too expensive compared to the rest of the equipment.

As a next step the presented concept will be extended to be applied to a variaty of different working processes, strengthening the application of industrial robots as a multi-purpose tool. To further improve the accuracy of the generated predictions, the usage of operator specific profiles is considered to better model individual preferences of the different workers.

The results presented in the previous section which are based on a rather small first data set already indicate that welding as an example process as well as the programming strategy of different users can be modelled by the proposed method and the model is able to adapt to new data.

REFERENCES

[1] S. Kinkel, "Final report on the market potential of the developed new robotic technologies", *Report ISE.4 of the SMErobot Project*, 2009.

[2] R. Schraft, and C. Meyer, "The need for an intuitive teaching method for small and medium enterprises", *in Proc. ISR / Robotik,* Munich, 2006.

[3] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration", *Robotics and Autonomous Systems 57*, 2009, pp. 469-483.

[4] D. Aarno, and D. Kragic, "Motion intention recognition in robot assisted applications", *Robotics and Autonomous Systems 56*, 2008, pp. 692-705.

[5] J. Saunders, C. Nehaniv, and K. Dautenhahn, „Teaching robots by moulding behavior and scaffolding the environment", *in Proc. 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, Utah, 2006, pp. 118-125.

[6] M. Pardowitz, R. Zöllner, S. Knoop, and R. Dillmann, "Using physical demonstrations, background knowledge and vocal commments for task learning", *in Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, 2006, pp. 322-327.

[7] S. Ekvall, and D. Kragic, "Learning task models from multiple human demonstrations", *in Proc.15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006, pp. 358-363.

[8] B. Akan, B. Cürüklü, and L. Asplund, "Interacting with industrial robots through a multi-modal language and sensor systems", *in Proc. 39th International Symposium on Robotics (ISR)*, Seoul, 2008, pp. 66-69.

[9] N. Pires, G. Veiga, and R. Araujo, "Programming-by-demonstration in the coworker scenario for SMEs", *Industrial Robot: An International Journal*, Volume 36, Issue 1, Emerald Publishing, 2009, pp. 73-83.

[10] Y. Maeda, T. Ushioda, and S. Makita, "Easy robot programming for industrial manipulators by manual volume sweeping", *in Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, USA, 2008, pp. 2234-2239.

[11] A. Albu-Schaefer, G. Hirzinger, "Cartesian impedance control techniques for torque controlled light-weight robots", *in Proc. 2002 IEEE International Conference on Robotics and Automation (ICRA)*, Washington, 2002.

[12] R. Hollmann, and M. Hägele, "The use of voice control for industrial robots in noisy manufacturing environments", in Proc. *39th International Symposium on Robotics (ISR),* Seoul, 2008, pp. 14-18.

[13] A. Breckweg, C. Meyer, K. Drechsler, and O. Rüger, "Fast and intuitive robot programming for mandrel guiding of braiding machines for textile preforming ", *in Proc. 28th SAMPE EUROPE International Conference* , Paris, 2007, pp. 578-584.

[14] T. Dietterich, "Machine learning for sequential data: a review", *in Proc. Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition,* 2002, pp. 15-30.

[15] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition", *in Proc .of the IEEE, Vol. 77, No. 2*,1989, pp. 257-285.

[16] G. Forney, "The viterbi algorithm", *in Proc. of the IEEE, Vol. 61, No. 3*, 1973, pp. 268-278.

[17] R. Hollmann, M. Hägele, and A. Verl, "Learning Probabilistic Models to Enhance the Efficiency of PbD for Industrial Robots", *Joint 41th International Symposium on Robotics and 6th German Conference on Robotics*, Munich, 2010, accepted for publication.