12

Robot Learning from Demonstration Using Predictive Sequence Learning

Erik Billing, Thomas Hellström and Lars-Erik Janlert
Department of Computing Science, Umeå University
Sweden

1. Introduction

In this chapter, the prediction algorithm Predictive Sequence Learning (PSL) is presented and evaluated in a robot *Learning from Demonstration* (LFD) setting. PSL generates hypotheses from a sequence of sensory-motor events. Generated hypotheses can be used as a semi-reactive controller for robots. PSL has previously been used as a method for LFD (Billing et al., 2010; 2011) but suffered from combinatorial explosion when applied to data with many dimensions, such as high dimensional sensor and motor data. A new version of PSL, referred to as *Fuzzy Predictive Sequence Learning* (FPSL), is presented and evaluated in this chapter. FPSL is implemented as a Fuzzy Logic rule base and works on a continuous state space, in contrast to the discrete state space used in the original design of PSL. The evaluation of FPSL shows a significant performance improvement in comparison to the discrete version of the algorithm. Applied to an LFD task in a simulated apartment environment, the robot is able to learn to navigate to a specific location, starting from an unknown position in the apartment.

Learning from Demonstration is a well-established technique for teaching robots new behaviors. One of the greatest challenges in LFD is to implement a learning algorithm that allows the robot pupil to generalize a sequence of actions demonstrated by the teacher such that the robot is able to perform the desired behavior in a dynamic environment. A behavior may be any complex sequence of actions executed in relation to sensor data (Billing & Hellström, 2010).

The LFD problem is often formulated as four questions, *what-to-imitate*, *how-to-imitate*, *when-to-imitate* and *who-to-imitate* which leads up to the larger problem of how to evaluate an imitation (Alissandrakis et al., 2002). Large parts of the literature approach the learning problem by trying to find the common features within a set of demonstrations of the same behavior. A skill is generalized by exploiting statistical regularities among the demonstrations (e.g. Calinon, 2009). This is reflected in the *what-to-imitate* question, originally introduced in a classical work by Nehaniv & Dautenhahn (2000) and is in a longer form described as:

> An action or sequence of actions is a successful component of imitation of a particular action if it achieves the same subgoal as that action. An entire sequence of actions is successful if it successively achieves each of a sequence of abstracted subgoals.

The problem is difficult since a certain behavior can be imitated on many different abstraction levels. Byrne & Russon (1998) identified two levels; the *action-level imitation* copying the surface of the behavior and a *program-level imitation* copying the structure of the behavior. A

third level, the *effect-level imitation*, was introduced by Nehaniv & Dautenhahn (2001) in order to better describe imitation between agents with dissimilar body structures. Demiris & Hayes (1997) proposed three slightly different levels: *1) basic imitation* with strong similarities to the notion of action-level imitation, 2) *functional imitation* that best corresponds to effect-level imitation and 3) *abstract imitation* that represents coordination based on the presumed internal state of the agent rather than the observed behavior. Demiris and Hayes give the example of making a sad face when someone is crying.

The necessity to consider the level of imitation in LFD becomes apparent when considering two demonstrations that look very different considered as sequences of data, but that we as humans still interpret as examples of the same behavior since they achieve similar results on an abstract level. This would correspond to a functional or program-level imitation. In these situations it is very difficult to find similarities between the demonstrations without providing high level knowledge about the behavior, often leading to specialized systems directed to LDF in limited domains.

A related problem is that two demonstrations of the same behavior may not have the same length. If one demonstration takes longer time than another, they can not be directly compared in order to find common features. Researchers have therefore used techniques to determine the temporal alignment of demonstrations. One common technique is *dynamic time warping* (Myers & Rabiner, 1981), that can be used to compensate for temporal differences in the data.

Behaviors can be demonstrated to a robot in many different ways. Argall et al. (2009) outline four types of demonstrations: A direct recording of sensor stimuli, joint angles, etc., is referred to as an *identity record mapping*. In this case, the robot is often used during the demonstration and controlled via teleoperation or by physically moving the robot's limbs (kinestetic teaching). An external observation, e.g. a video recording of the teacher, is called a *non-identity record mapping*. This type of demonstrations poses a difficult sensing problem of detecting how the teacher has moved, but also allows much more flexible demonstration setting. The teacher may have a body identical to that of the pupil (*identity embodiment*) or a body with a different structure (*non-identity embodiment*). In the latter case, the demonstration has to be transformed into corresponding actions using the body of the pupil, a difficult problem known as the *correspondence problem* (Nehaniv & Dautenhahn, 2001). In this work we focus on LFD via teleoperation. Sensor data and motor commands are in this setting recorded while a human teacher demonstrates the desired behavior by tele-operating the robot, producing demonstrations with identity in both record mapping and embodiment.

1.1 Metric of imitation

Successful imitation requires that relevant features of the demonstration are selected at a suitable imitation level and processed into a generalized representation of the behavior. The process is difficult to implement in a robot since it is often far from obvious which imitation level that is optimal in a specific situation, and the relevance of features may consequently vary significantly from one learning situation to another. This problem has been formalized as a *metric of imitation*, defined as a weighted sum over all strategy-dependent metrics on all imitation levels (Billard et al., 2003).

The metric of imitation was originally demonstrated on a manipulation task with a humanoid robot (Billard et al., 2003). With focus on the correspondence problem, Alissandrakis et al. (2005) propose a similar approach to imitation of manipulation tasks. The what-to-imitate problem is approached by maximizing trajectory agreements of manipulated objects, using several different metrics. Some metrics encoded absolute trajectories while other metrics

encoded relative object displacement and the relevant aspects of the behavior were in this way extracted as the common features in the demonstration set. Calinon et al. (2007) developed this approach by encoding the demonstration set using a mixture of Gaussian/Bernoulli distributions. The Gaussian Mixture Model approach is attractive since the behavior is divided into several distributions with different covariance, and different metrics can in this way be selected for different parts of the demonstrated behavior. More recently, similar encoding strategies have been evaluated for learning of a robot navigation task (de Rengervé et al., 2010).

1.2 Behavior primitives as a basis for imitation

Another common approach to LFD is to map the demonstration onto a set of pre-programmed or previously learned primitives controllers (Billing & Hellström, 2010). The approach has strong connections to *behavior-based architectures* (Arkin, 1998; Matarić, 1997; Matarić & Marjanovic, 1993) and earlier reactive approaches (e.g. Brooks, 1986; 1991). When introducing behavior primitives, the LFD process can be divided into three tasks (Billing & Hellström, 2010):

1. *Behavior segmentation* where a demonstration is divided into smaller segments.
2. *Behavior recognition* where each segment is associated with a primitive controller.
3. *Behavior coordination*, referring to identification of rules or switching conditions for how the primitives are to be combined.

Behavior segmentation and recognition can be seen as one way to approach the what-to-imitate problem, whereas behavior coordination is part of how-to-imitate. The approach represents one way of introducing good bias in learning and solve the generalization problem by relying on previous behavioral knowledge. While there are many domain specific solutions to these three subproblems, they appear very difficult to solve in the general case. Specifically, behavior recognition poses the problem of mapping a sequence of observations to a set of controllers to which the input is unknown. Again, the need to introduce a metric of imitation appears.

Part of the problem to find a general solution to these problems may lie in a vague definition of *behavior* (Matarić, 1997). The notion of behavior is strongly connected to the purpose of executed actions and a definition of goal. Nicolescu (2003) identified two major types of goals:

Maintenance goals: A specific condition has to be maintained for a time interval.

Achievement goals: A specific condition has to be reached.

The use of behavior primitives as a basis for imitation has many connections to biology (e.g. Matarić, 2002) and specifically the mirror system (Brass et al., 2000; Gallese et al., 1996; Rizzolatti et al., 1988; Rizzolatti & Craighero, 2004). While the role of the mirror system is still highly debated, several groups of researchers propose computational models where perception and action are tightly interweaved. Among the most prominent examples are the *HAMMER* architecture (Demiris & Hayes, 2002; Demiris, 1999; Demiris & Johnson, 2003) and the *MOSAIC* architecture (Haruno et al., 2001; Wolpert & Kawato, 1998). Both these architectures implement a set of modules, where each module is an inverse model (controller) paired with a forward model (predictor). The inverse and forward models are trained together such that the forward model can predict sensor data in response to the actions produced by the inverse model. The inverse model is tuned to execute a certain behavior when the forward model produces good predictions. The prediction error is used to compute a bottom-up

signal for each module. Based on the bottom-up signal, a top-down responsibility signal or confidence value is computed and propagated to each module. The output of the system is a combination of the actions produced by each inverse model, proportional to their current responsibility. The responsibility signal also controls the learning rate of each module, such that modules are only updated when their responsibility is high. In this way, modules are tuned to a specific behavior or parts of a behavior. Since the prediction error of the forward model is used as a measure of how well the specific module fits present circumstances, it can be seen as a metric of imitation that is learnt together with the controller. The architecture can be composed into a hierarchical system where modules are organized in layers, with the lowest layer interacting with sensors and actuators. The bottom-up signal constitutes sensor input for the layer above and actions produced by higher levels constitutes the top-down responsibility signal.

One motivation for this architecture lies in an efficient division of labor between different parts of the system. Each module can be said to operate with a specific temporal resolution. Modules at the bottom layer are given the highest resolution while modules higher up in the hierarchy have decreasing temporal resolution. State variables that change slowly compared to a specific module's resolution are ignored by that module and are instead handled by modules higher up in the hierarchy. Slowly changing states that lead to high responsibility for the module is referred to as the module's *context*. In a similar fashion, variables that change fast in comparison to the temporal resolution are handled lower in the hierarchy. This allows each module to implement a controller where the behavior depends on relatively recent states. Long temporal dependencies are modeled by switching between modules, which removes the requirement for each model to capture these dependencies. Furthermore, updates of a single behavior or parts of a behavior will only require updates of a few modules and will not propagate changes to other modules. See Billing (2009) for a longer discussion on these aspects of hierarchical architectures.

The HAMMER and MOSAIC architectures make few restrictions on what kind of controllers each module should implement. We argue however, that modules should be *semi-reactive*, meaning that action selection and predictions of sensor events should be based on recent sensor and motor events. Strictly reactive modules are not desirable since each module must be able to model any dependency shorter than the temporal resolution of modules in the layer directly above.

The division of behavior into modules is however also producing a number of drawbacks. The possibility for the system to share knowledge between behaviors is limited. Moreover, the system has to combine actions produced by different modules, which may be difficult in cases when more than one module receives high responsibility.

One architecture with similarities to HAMMER and MOSAIC able to share knowledge between different behaviors is *RNNPB* (Tani et al., 2004). *RNNPB* is a recurrent neural network with parametric bias (PB). Both input and output layer of the network contains sensor and motor nodes as well as nodes with recurrent connections. In addition, the input layer is given a set of extra nodes, representing the PB vector. The network is trained to minimize prediction error, both by back-propagation and by changing the PB vector. The PB vector is however updated slowly, such that it organizes into what could be seen as a context layer for the rest of the network. In addition to giving the network the ability to represent different behaviors that share knowledge, the PB vector can be used for behavior recognition.

Another architecture known as *Brain Emulating Cognition and Control Architecture* (BECCA) (Rohrer & Hulet, 2006) heavily influenced our early work on the PSL algorithm. The focus

of BECCA is to capture the discrete episodic nature of many types of human motor behavior, without introducing a priori knowledge into the system. BECCA was presented as a very general reinforcement learning system, applicable to many types of learning and control problems. One of the core elements of BECCA is the temporal difference (TD) algorithm *Sequence Learning* (SL) (Rohrer, 2007). SL builds sequences of passed events which is used to predict future events, and can in contrast to other TD algorithms base its predictions on many previous states.

Inspired by BECCA and specifically SL, we developed the PSL algorithm as a method for LFD (Billing et al., 2010; 2011). PSL has many interesting properties seen as a learning algorithm for robots. It is model free, meaning that it introduces very few assumptions into learning and does not need any task specific configuration. PSL can be seen as a variable-order Markov model. Starting out from a reactive (first order) model, PSL estimates transition probabilities between discrete sensor and motor states. For states that do not show Markov property, the order is increased and PSL models the transition probability based on several passed events. In this way, PSL will progressively gain memory for parts of the behavior that cannot be modeled in a reactive way. In theory, there is no limitation to the order of the state and hence the length of the memory, but PSL is in practice unable to capture long temporal dependencies due to combinatorial explosion.

PSL has been evaluated both as a controller (Billing et al., 2011) and as a method for behavior recognition (Billing et al., 2010). Even though the evaluation overall generated good results, PSL is subject to combinatorial explosion both when the number of sensors and actuators increase, and when the demonstrated behavior requires modeling of long temporal dependencies. PSL can however efficiently model short temporal dependencies in a semi-reactive way and should thus be a good platform for implementing a hierarchical system similar to the HAMMER and MOSAIC architectures.

In this chapter, we present and evaluate a new version of PSL based on Fuzzy Logic. While keeping the core idea of the original PSL algorithm, the new version can handle continuous and multi dimensional data in a better way. To distinguish between the two, the new fuzzy version of the algorithm is denoted FPSL, whereas the previous discrete version is denoted DPSL. A detailed description of FPSL is given in Section 2. An evaluation with comparisons between the two algorithms is presented in Section 3, followed by a discussion and conclusions in section 4.

2. Predictive Sequence Learning

FPSL builds fuzzy rules, referred to as *hypotheses* h, describing temporal dependencies between a sensory-motor event e_{t+1} and a sequence of passed events $\left(e_{t-|h|+1}, e_{t-|h|+2}, \ldots, e_t\right)$, defined up until current time t.

$$h : \left(\mathrm{Y}_{t-|h|+1} \; is \; E^h_{|h|-1} \wedge \mathrm{Y}_{t-|h|+2} \; is \; E^h_{|h|-2} \wedge \ldots \wedge \mathrm{Y}_t \; is \; E^h_0\right) \overset{C}{\Rightarrow} \mathrm{Y}_{t+1} \; is \; \bar{E}^h \tag{1}$$

Y_i is the event variable and $E^h(e)$ is a fuzzy membership function returning a membership value for a specific e. The right hand side $\bar{E}^h$ is a membership function comprising expected events at time $t+1$. $|h|$ denotes the length of h, i.e., the number of left-hand-side conditions of the rule. Both E and $\bar{E}$ are implemented as standard cone membership functions with base width ε (e.g. Klir & Yuan, 1995).

A set of hypotheses can be used to compute a prediction $\hat{e}_{t+1}$ given a sequence of passed sensory-motor events η, defined up to the current time t:

$$\eta = (e_1, e_2, \ldots, e_t) \tag{2}$$

The process of matching hypothesis to data is described in Section 2.1. The PSL learning process, where hypotheses are generated from a sequence of data, is described in Section 2.2. Finally, a discussion about parameters and configuration is found in Section 2.3.

2.1 Matching hypotheses

Given a sequence of sensory-motor events $\eta = (e_1, e_2, \ldots, e_t)$, a match $\alpha_t(h)$ of the rule is given by:

$$\alpha_t(h) : \bigwedge_{i=0}^{|h|-1} E_i^h(e_{t-i}) \tag{3}$$

where $\wedge$ is implemented as a *min*-function.
Hypotheses are grouped in fuzzy sets C whose membership value $C(h)$ describes the confidence of h at time t:

$$C(h) = \frac{\sum_{k=t^h}^{t} \alpha_k(h)\, \bar{E}^h(e_{k+1})}{\sum_{k=t^h}^{t} \alpha_k(h)} \tag{4}$$

.
t^h is the creation time of h or 1 if h existed prior to training. Each C represents a *context* and can be used to implement a specific behavior or part of a behavior. The *responsibility signal* $\lambda_t(C)$ is used to control which behavior that is active at a specific time. The combined confidence value $\tilde{C}_t(h)$ is a weighted sum over all C:

$$\tilde{C}_t(h) = \frac{\sum_C C(h)\, \lambda_t(C)}{\sum_C \lambda_t(C)} \tag{5}$$

.
$\tilde{C}_t$ can be seen as a fuzzy set representing the active context at time t. Hypotheses contribute to a prediction in proportion to their membership in $\tilde{C}$ and the *match set* $\hat{M}$. $\hat{M}$ is defined in three steps. First, the best matching hypotheses for each $\bar{E}$ is selected:

$$M = \left\{ h \mid \alpha(h) \geq \alpha(h')\ for\ all\ \left\{ h' \mid \bar{E}^{h'} = \bar{E}^h \right\} \right\} \tag{6}$$

.
The longest $h \in M$ for each RHS is selected:

$$\tilde{M} = \left\{ h \mid |h| \geq |h'|\ for\ all\ \left\{ h' \in M \mid \bar{E}^{h'} = \bar{E}^h \right\} \right\} \tag{7}$$

.
Finally, the match set $\hat{M}$ is defined as:

$$\hat{M}(h) = \begin{cases} \alpha(h)\, \tilde{C}(h) & h \in \tilde{M} \\ 0 & otherwise \end{cases} \tag{8}$$

.
The aggregated prediction $\hat{E}(e_{t+1})$ is computed using the Larsen method (e.g. Fullér, 1995):

$$\hat{E}\left(e_{t+1}\right) : \bigvee_{h} \bar{E}_h\left(e_{t+1}\right) \hat{M}\left(h\right) \tag{9}$$

$\hat{E}$ is converted to crisp values using a squared *center of sum* defuzzification:

$$\hat{e} = \frac{\sum_{e} e\hat{E}\left(e\right)^2}{\sum_{e} \hat{E}\left(e\right)^2} \tag{10}$$

The amount of entropy in $\hat{M}$ also bears information about how reliable a specific prediction is, referred to as the *trust* $\hat{c}$:

$$\hat{c}\left(\hat{M}\right) = \begin{cases} 0 & \hat{M} = \emptyset \\ \exp\left[\sum_{h} \hat{M}\left(h\right) \log_2\left(\hat{M}\left(h\right)\right)\right] & otherwise \end{cases} \tag{11}$$

The trust is important since it allows a controller to evaluate when to rely on PSL, and when to choose an alternate control method. The proportion of time steps in η for which $\hat{c} > 0$ and PSL is able to produce a prediction is referred to as the *coverage* $\phi\left(\eta\right)$:

$$\phi\left(\eta\right) = \frac{\sum_{i=1}^{t} \begin{cases} 1 & \hat{c}_i > 0 \\ 0 & otherwise \end{cases}}{t} \tag{12}$$

2.2 Generating hypotheses

Hypotheses can be generated from a sequence of sensory-motor events η. During training, PSL continuously makes predictions and creates new hypotheses when no matching hypothesis produces the correct prediction $\bar{E}$. The exact training procedure is described in Algorithm 0.1.

For example, consider the event sequence $\eta = abccabccabcc$. Let $t = 1$. PSL will search for a hypothesis with a body matching a. Initially, the context set C is empty and consequently PSL will create a new hypothesis $(a) \Rightarrow b$ which is added to C with confidence 1, denoted $C\left(a \Rightarrow b\right) = 1$. The same procedure will be executed at $t = 2$ and $t = 3$ such that $C\left((b) \Rightarrow c\right) = 1$ and $C\left((c) \Rightarrow c\right) = 1$. At $t = 4$, PSL will find a matching hypothesis $(c) \Rightarrow c$ producing the wrong prediction c. Consequently, a new hypothesis $(c) \Rightarrow a$ is created and confidences are updated such that $C\left((c) \Rightarrow c\right) = 0.5$ and $C\left((c) \Rightarrow a\right) = 1$. The new hypothesis receives a higher confidence since confidence values are calculated from the creation time of the hypothesis (Equation 4). The predictions at $t = 5$ and $t = 6$ will be correct and no new hypotheses are created. At $t = 7$, both $(c) \Rightarrow a$ and $(c) \Rightarrow c$ will contribute to the prediction $\hat{E}$. Since the confidence of $(c) \Rightarrow a$ is higher than that of $(c) \Rightarrow c$, $\hat{E}$ will defuzzify towards a, producing the wrong prediction (Equation 10). As a result, PSL creates a new hypothesis $(b, c) \Rightarrow c$. Similarly, $(c, c) \Rightarrow a$ will be created at $t = 8$. PSL is now able to predict all elements in the sequence perfectly and no new hypotheses are created.

Source code from the implementation used in the present work is available online (Billing, 2011).

Algorithm 0.1 Predictive Sequence Learning (PSL)

Require: $\psi = (e_1, e_2, \ldots, e_T)$ where T denotes the length of the training set
Require: $\hat{\alpha}$ as the precision constant, see text

let $t \leftarrow 1$
let $\eta = (e_1, e_2, \ldots, e_t)$
let $C \leftarrow \varnothing$
let $\hat{E}$ as Eq. 9
if $\hat{E}(e_{t+1}) < \hat{\alpha}$ **then**
    let $h_{new} = CreateHypothesis(\eta, C)$ as defined by Algorithm 0.2
    $C(h_{new}) \leftarrow 1$
end if
Update confidences $C(h)$ as defined by Equation 4
set $t = t + 1$
if t<T **then**
    goto 2
end if

Algorithm 0.2 CreateHypothesis

Require: $\eta = (e_1, e_2, \ldots, e_t)$
Require: $C : h \rightarrow [0, 1]$
Require: α as defined by Eq. 3

let $\hat{M}(h)$ as Eq. 8
let $\bar{M} = \left\{ h \mid \bar{E}^h(e_{t+1}) \geq \hat{\alpha} \wedge \hat{M}(h) > 0 \right\}$ where $\hat{\alpha}$ is the precision constant, see Section 2.3
if $\bar{M} = \emptyset$ **then**
    let E^* be a new membership function with center e_t and base ε
    return $h_{new} : (\mathrm{Y}_t \ is \ E^*) \Rightarrow \mathrm{Y}_{t+1} \ is \ \bar{E}$
else
    let $\bar{h} \in \bar{M}$
    if $C(\bar{h}) = 1$ **then**
        return *null*
    else
        let E^* be a new membership function with center $e_{t-|\bar{h}|}$ and base ε
        return $h_{new} : \left(\mathrm{Y}_{t-|\bar{h}|} \ is \ E^*, \mathrm{Y}_{t-|\bar{h}|+1} \ is \ E^{\bar{h}}_{|\bar{h}|-1}, \ldots, \mathrm{Y}_t \ is \ E^{\bar{h}}_0 \right) \Rightarrow \mathrm{Y}_{t+1} \ is \ \bar{E}$
    end if
end if

2.3 Parameters and task specific configuration

A clear description of parameters is important for any learning algorithm. Parameters always introduce the risk that the learning algorithm is tuned towards the evaluated task, producing better results than it would in the general case. We have therefore strived towards limiting the number of parameters of PSL. The original design of PSL was completely parameter free, with the exception that continuous data was discretized using some discretization method. The version of PSL proposed here can be seen as a generalization of the original algorithm (Billing et al., 2011) where the width ε of the membership function E determines the discretization resolution. In addition, a second parameter is introduced, referred to as the *precision constant* $\hat{\alpha}$. $\hat{\alpha}$ is in fuzzy logic terminology an α*-cut*, i.e., thresholds over the fuzzy membership function in the interval $[0, 1]$ (Klir & Yuan, 1995).

ε controls how generously FPSL matches hypotheses. A high ε makes the algorithm crisp but typically increases the precision of predictions when a match is found. Contrary, a low ε reduces the risk that FPSL reaches unobserved states at the cost of a decreased prediction performance. The high value of ε can be compared to a fine resolution data discretization for the previous version of PSL.

$\hat{\alpha}$ is only used during learning, controlling how exact a specific $\bar{E}$ has to be before a new hypothesis with a different $\bar{E}$ is created. A large $\hat{\alpha}$ reduces prediction error but typically results in more hypotheses being created during learning.

Both ε and $\hat{\alpha}$ controls the tolerance to random variations in the data and can be decided based on how exact we desire that FPSL should model the data. Small ε in combination with large $\hat{\alpha}$ will result in a model that closely fits the training data, typically producing small prediction errors but also a low coverage.

3. Evaluation

Two tests were performed to evaluate the performance of FPSL and compare it to the previous version. A simulated Robosoft Kompai robot (Robosoft, 2011) was used in the Microsoft RDS simulation environment (Microsoft, 2011). The 270 degree laser scanner of the Kompai was used as sensor data and the robot was controlled by setting linear and angular speeds.

Demonstrations were performed via tele-operation using a joypad, while sensor and motor data were recorded with a temporal resolution of 20 Hz. The dimensionality of the laser scanner was reduced to 20 dimensions using an average filter. Angular and linear speeds were however fed directly into PSL.

The first test (Section 3.1) was designed to compare FPSL and DPSL as prediction algorithms, using sensor data from the simulated robot. The second test (Section 3.2) demonstrates the use of FPSL as a method for LFD.

3.1 Sensor prediction

The two versions of PSL were compared using a series of tests of prediction performance. Even though DPSL and FPSL are similar in many ways, a comparison is not trivial since DPSL works on discrete data whereas FPSL uses continuous data. Prediction performance of DPSL will hence depend on how the data is discretized while the performance of FPSL depends on the parameters ε and $\hat{\alpha}$.

To capture the prediction performance of the two algorithms using different configurations, a series of tests were designed. 10 discretization levels were chosen, ranging from a fine resolution where DPSL could only produce predictions on a few samples in the test set, to a low resolution where DPSL rarely met unobserved states. Laser readings were discretized

Fig. 1. Simulation Environment (Microsoft, 2011) used for evaluations. Blue stars and yellow dots represent starting positions used for demonstrations and test runs, respectively. The green area marked with a *G* represents the target position. The white area under star 10 is the robot.

over 0.8 m for the finest resolution, up to 8 m for the lowest resolution. Motor data was discretized over 0.06m/s for the finest resolution up to 0.6 m/s for the lowest resolution. Similarly, 10 ε values were chosen, corresponding to a cone base ranging from 0.8 m to 8 m for laser data, and 0.06 m/s up to 0.6 m/s for motor data. $\hat{\alpha}$ was given a constant value of 0.9, corresponding to a error tolerance of 10% of ε.
10 data files were used, each containing a demonstration where the teacher directed the robot from a position in the apartment to the TV, see Figure 1. A rotating comparison was used, where PSL was tested on one demonstration at a time and the other nine demonstrations were used as training data. Prediction performance was measured in meters on laser range data.

3.1.1 Results

The results from the evaluation are illustrated in Figure 2. While the ε value of FPSL cannot directly be compared to the discretization level used for DPSL, the two parameters have similar effect on coverage. Prediction error is only calculated on the proportion of the data for which prediction are produced, and consequently, prediction error increases with coverage.

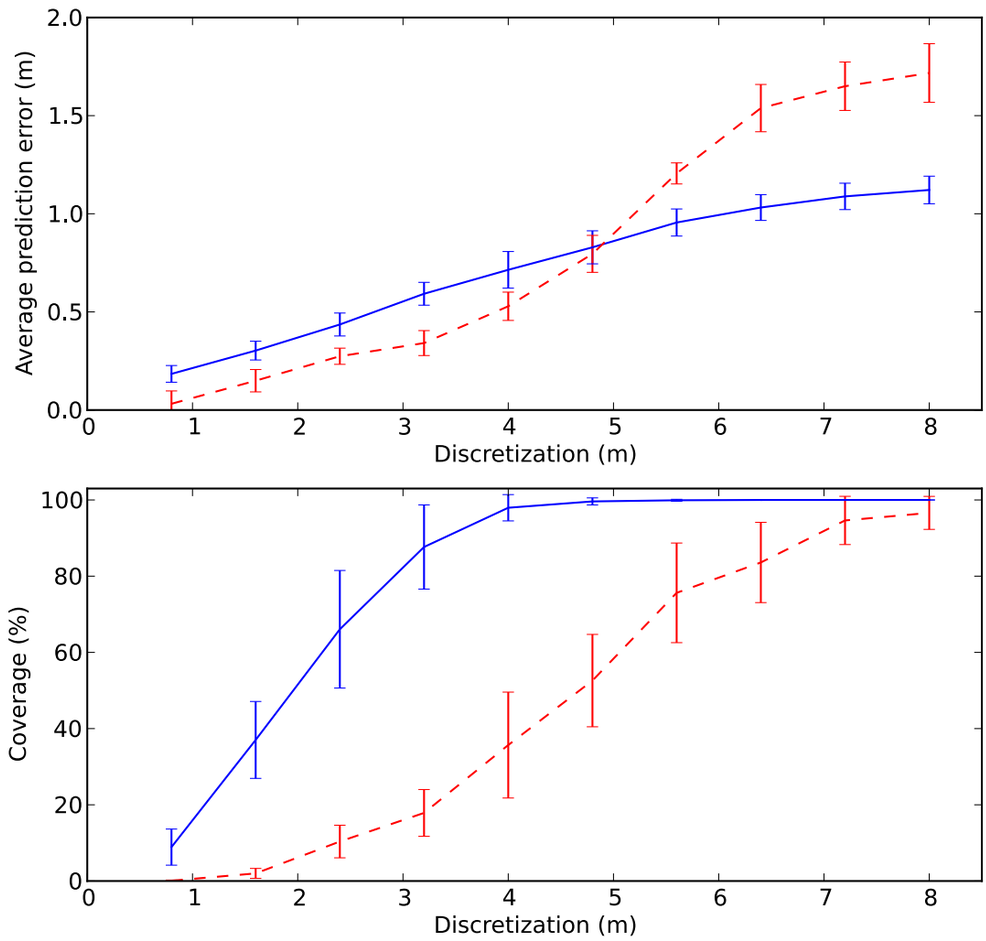


Fig. 2. Results from the prediction evaluation (Section 3.1). Upper plot shows prediction errors for FPSL (solid line) and DPSL (dashed line). Lower plot shows coverage , i.e. the proportion of samples for which the algorithm generated predictions, see Equation 12. Vertical bars represent standard deviation.

3.2 Target reaching

This evaluation can be seen as a continuation of previous tests with DPSL using a Khepera robot (Billing et al., 2011). The evaluation is here performed in a more complex environment, using a robot with much larger sensor dimensionality. Initial tests showed that DPSL has sever problems to handle the increased sensor dimensionality when used as a controller. A discretization resolution of about 2 m appeared necessary in order to produce satisfying discrimination ability. Even with this relatively low resolution, the 20 dimensional data produced a very large state space causing DPSL to frequently reach unrecognized states. DPSL could control the robot after intense training in parts of the test environment, but could

not compete with FPSL in a more realistic setting. We therefore chose to not make a direct controller comparison, but rather show the behavior of FPSL in an applied and reproducible setting.
FPSL was trained on 10 demonstrations showing how to get to the TV from various places in the apartment, see Figure 1. The performance of FPSL as a method for LFD was evaluated by testing how often the robot reached the target position in front of the TV starting out from 10 different positions than the ones used during training. FPSL controlled the robot by continuously predicting the next sensory-motor event based on the sequence of passed events. The motor part of the predicted element was sent to the robot controller. A standard reactive obstacle avoidance controller was used as fallback in cases where FPSL did not find any match with observed data. The task was considered successfully executed if the target position was reached without hitting any walls or obstacles. The experiment was repeated ten times, producing a total of 100 test runs.

3.2.1 Results

FPSL successfully reached the target position in front of the TV (the green area in Figure 1) in 79% of the test runs. In 68 runs, it stopped in front of the TV as demonstrated, but in 11 runs it failed to stop even though it reached the target position. The distribution over the 10 starting positions illustrated in Figure 3.

4. Discussion

Applied as a robot controller, PSL is a semi-reactive generative model that produces both actions and expected observations, based on recent sensory-motor events. We believe that this approach to robot learning has great potential since the behavior can be learnt progressively and previous knowledge contributes to the interpretation of new events. It is also general in the sense that very little domain specific knowledge is introduced. Memories are stored as sequences of sensory-motor events that in principle can represent any behavior. While PSL efficiently can represent behaviors with short temporal dependencies, it is subject to combinatorial explosion when the behavior requires representations over longer time spans. We argue that the gradually extending memory of PSL, from being purely reactive to containing representations over longer time when needed, provides a good bias in learning. It will however make learning of behaviors that do require long temporal dependencies slow. The fuzzy version of PSL presented in this work does not directly provide a solution to this problem, but is one step towards integrating PSL in a hierarchical structure as discussed in Section 1.2.
The seeds to FPSL came from the observation that a lot of training was required in order to cover the state space of DPSL with satisfying resolution. A better tradeoff between high precision in prediction and coverage would make PSL a more competitive alternative for real world LFD scenarios. Without sacrificing the strong attributes of the original PSL algorithm, such as the model free design, few parameters and progressively growing representations, FPSL was designed.
Expressing PSL with Fuzzy Logic is in many ways a natural extension and generalization of the discrete algorithm. By using a discrete uniform membership function E and a *max* operator for defuzzification, FPSL becomes very similar to DPSL. Even though the processing of continuous values does add significant processing requirements in comparison to DPSL, FPSL can still be efficiently implemented as a fuzzy rule controller.

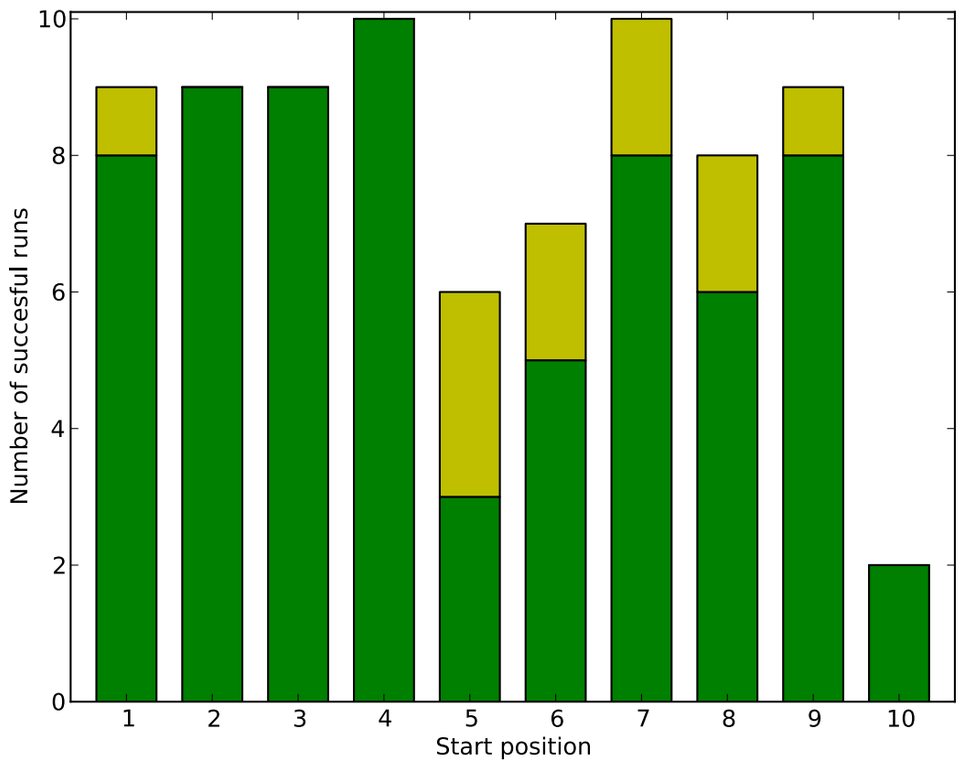


Fig. 3. Results from test runs in simulated environment (Section 3.2). Each bar corresponds to one starting position, see Figure 1. The green part of the bar represents number of successful runs where the robot reached and stopped at the target position in front of the TV. The yellow part represents runs when the robot successfully reached the target, but did not stop. The test was executed 10 times from each starting position.

The evaluation shows that FPSL produces significantly smaller prediction errors in relation to the coverage than DPSL (Section 3.1). This was expected since FPSL can be trained to produce small prediction errors by keeping a high precision constant $\hat{\alpha}$, while the coverage is still kept high by using a large ε. In contrast, when using DPSL, one must choose between a small prediction error with low coverage or a high coverage at the price of an increased prediction error. As can be seen in Figure 2, FPSL is also affected by the precision/coverage tradeoff, but not nearly as much as DPSL. Furthermore, the number of generated hypotheses will increase with $\hat{\alpha}$, which also has a positive effect on coverage for multidimensional data.
While FPSL performs much better than DPSL on large and multidimensional state spaces, it should not be seen as a general solution to the dimensionality problem. The increased number of hypotheses results in increased processing and memory requirements. Furthermore, FPSL is still not able to ignore uncorrelated dimensions in data, making it subject to the curse of dimensionality. One potential solution is to modify the size of membership functions in relation to the information content of the dimension. However, initial tests did not produce satisfying results and further experiments in this direction were postponed to future work.

We found the results from the controller evaluation very promising (Section 3.2). The application environment has been scaled up significantly in comparison to previous work (Billing et al., 2011) and we are now able to perform learning in a fairly realistic setting. When observing these results one should remember that PSL does not solve a spatial task. There is no position sensor or internal map of the environment and the robot is still able to navigate from almost any position in the environment, to a specific target location. The goal is better described as an attractor in a dynamic system, where the robot in interaction with the environment finally reaches a stable state in front of the TV (Figure 1).
An interesting observation is that it is often difficult to predict how well PSL will be able to handle a specific situation. For example, starting position 6 was not passed during any demonstration, but PSL still managed to control the robot such that it reached the target in 7 out of 10 test runs. On the other hand, position 5 and 10 produced worse results than expected. Even though these starting positions were spatially close to several positions passed during the demonstrations, the directions at which the robot reached these positions were different, producing different laser scans, and PSL could consequently not find a suitable match. In some of the cases, inappropriate matches were found and the robot turned in the wrong direction. In other cases, no match at all was found causing the robot to fall back on the reactive controller for a longer period and usually getting stuck in a corner.
The amount of training data used in this evaluation was fairly small. Only one demonstration from each starting position was performed. One reason why FPSL is able to solve the task despite the small amount of training is that all data potentially contribute to every action selection, independently of where in the demonstration it originally took place. Techniques that represent the whole behavior as a sequence with variations, typically require more training since information from the beginning of the demonstration does not contribute to action selection in other parts of the behavior. PSL does not relies on common features within a set of demonstrations and consequently does not require that demonstrations are compared or temporally aligned, see Section 1. In its current design, PSL is of course unable to perform a program-level imitation since it always rely on sensory-motor events, but it does not suffer from a large diversity in the demonstration set as long as the recent sensory-motor events bear necessary information to select a suitable action.

4.1 Conclusions and future work

In this chapter, we show that PSL can be used as a method for LFD, in a fairly realistic setting. The move from a discrete state space used in previous work to the continuous state space appears to have a positive effect on generalization ability and prediction performance, especially on multi-dimensional data. The next step is to conduct experiments with the physical Kompai robot (Robosoft, 2010) in an attempt to verify the results in the real world.
The fuzzy version of PSL proposed here, and specifically the introduction of context sets C, should be seen as one step towards integrating PSL in a hierarchical architecture. The higher level controller may be another instance of PSL working on a lower temporal resolution, or a completely different control system interacting with PSL by changing the responsibility $\lambda_t(C)$ for each context (Equation 5). For an actual interaction to take place, PSL also has to feed information upwards, to higher level controllers. In previous work on behavior recognition (Billing et al., 2010), we have shown that PSL can be used to compute a bottom-up signal providing information about how well each context corresponds to present circumstances. While this has not been the focus of this chapter, we intend to evaluate these aspects of PSL in future work.

5. References

Alissandrakis, A., Nehaniv, C. L. & Dautenhahn, K. (2002). Imitation With ALICE: Learning to Imitate Corresponding Actions Across Dissimilar Embodiments, *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 32: 482–496.

Alissandrakis, A., Nehaniv, C. L., Dautenhahn, K. & Saunders, J. (2005). An Approach for Programming Robots by Demonstration: Generalization Across Different Initial Configurations of Manipulated Objects, *Proceedings of 2005 International Symposium on Computational Intelligence in Robotics and Automation*, Ieee, Espoo, Finland, pp. 61–66.

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. (2009). A survey of robot learning from demonstration, *Robotics and Autonomous Systems* 57(5): 469–483.

Arkin, R. C. (1998). *Behaviour-Based Robotics*, MIT Press.

Billard, A., Epars, Y., Cheng, G. & Schaal, S. (2003). Discovering imitation strategies through categorization of multi-dimensional data, *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Vol. 3, Las Vegas, Nevada, pp. 2398–2403 vol.3.

Billing, E. A. (2009). *Cognition Reversed - Robot Learning from Demonstration*, Lic. thesis, UmeÃ¥ University, Department of Computing Science, UmeÃ¥, Sweden.

Billing, E. A. (2011). www.cognitionreversed.com.

Billing, E. A. & Hellström, T. (2010). A Formalism for Learning from Demonstration, *Paladyn: Journal of Behavioral Robotics* 1(1): 1–13.

Billing, E. A., Hellström, T. & Janlert, L. E. (2010). Behavior Recognition for Learning from Demonstration, *Proceedings of IEEE International Conference on Robotics and Automation*, Anchorage, Alaska.

Billing, E. A., Hellström, T. & Janlert, L. E. (2011). Predictive learning from demonstration, *in* J. Filipe, A. Fred & B. Sharp (eds), *Agents and Artificial Intelligence*, Springer Verlag, Berlin, pp. 186–200.

Brass, M., Bekkering, H., Wohlschläger, A. & Prinz, W. (2000). Compatibility between observed and executed finger movements: comparing symbolic, spatial, and imitative cues., *Brain and cognition* 44(2): 124–43.

Brooks, R. A. (1986). A Robust Layered Control System For A Mobile Robot, *IEEE Journal of Robotics and Automation* 2(1): 14–23.

Brooks, R. A. (1991). New Approaches to Robotics, *Science* 253(13): 1227–1232.

Byrne, R. W. & Russon, A. E. (1998). Learning by Imitation: a Hierarchical Approach, *The Journal of Behavioral and Brain Sciences* 16(3).

Calinon, S. (2009). *Robot Programming by Demonstration - A Probabilistic Approach*, EFPL Press.

Calinon, S., Guenter, F. & Billard, A. (2007). On Learning, Representing and Generalizing a Task in a Humanoid Robot, *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation* 37(2): 286–298.

de Rengervé, A., D'halluin, F., Andry, P., Gaussier, P. & Billard, A. (2010). A study of two complementary encoding strategies based on learning by demonstration for autonomous navigation task, *Proceedings of the Tenth International Conference on Epigenetic Robotics*, Lund, Sweden.

Demiris, J. & Hayes, G. (1997). Do robots ape?, *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents*, pp. 28–31.

Demiris, J. & Hayes, G. R. (2002). *Imitation as a dual-route process featuring predictive and learning components: a biologically plausible computational model*, MIT Press, pp. 327–361.

Demiris, Y. (1999). *Movement Imitation Mechanisms in Robots and Humans*, PhD thesis, University of Edinburgh.

Demiris, Y. & Johnson, M. (2003). Distributed, predictive perception of actions: a biologically inspired robotics architecture for imitation and learning, *Connection Science* 15(4): 231–243.

Fullér, R. (1995). *Neural Fuzzy Systems*, Abo Akademi University.

Gallese, V., Fadiga, L., Fogassi, L. & Rizzolatti, G. (1996). Action recognition in the premotor cortex, *Brain* 119(2): 593–609.

Haruno, M., Wolpert, D. M. & Kawato, M. M. (2001). MOSAIC Model for Sensorimotor Learning and Control, *Neural Comput.* 13(10): 2201–2220.

Klir, G. J. & Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall.

Matarić, M. J. (1997). Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior, *Journal of Experimental and Theoretical Artificial Intelligence* 9(2-3): 323–336.

Matarić, M. J. (2002). *Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics*, MIT Press, pp. 391–422.

Matarić, M. J. & Marjanovic, M. J. (1993). Synthesizing Complex Behaviors by Composing Simple Primitives, *Proceedings of the European Conference on Artificial Life*, Vol. 2, Brussels, Belgium, pp. 698–707.

Microsoft (2011). Microsoft Robotic Developer Studio.
URL: *http://www.microsoft.com/robotics/*

Myers, B. C. S. & Rabiner, L. R. (1981). A Comparative Study of Several Dynamic Time-Warping, *The Bell System Technical Journal* 60(7): 1389–1409.

Nehaniv, C. L. & Dautenhahn, K. (2000). *Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications*, Vol. 24, World Scientific Press, pp. 136–161.

Nehaniv, C. L. & Dautenhahn, K. (2001). Like Me? - Measures of Correspondence and Imitation, *Cybernetics and Systems* 32: 11–51.

Nicolescu, M. (2003). *A Framework for Learning from Demonstration, Generalization and Practice in Human-Robot Domains*, PhD thesis, University of Southern California.

Rizzolatti, G., Camarda, R., Fogassi, L., Gentilucci, M., Luppino, G. & Matelli, M. (1988). Functional organization of inferior area 6 in the macaque monkey. II. Area F5 and the control of distal movements., *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale* 71(3): 491–507.

Rizzolatti, G. & Craighero, L. (2004). The Mirror-Neuron System, *Annual Review of Neuroscience* 27: 169–192.

Robosoft (2010). www.robosoft.com.

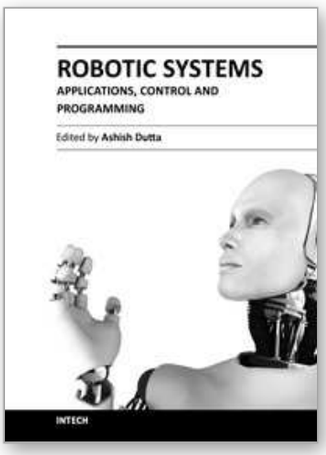
Robosoft (2011). Kompai Robot.

Rohrer, B. (2007). S-Learning: A Biomimetic Algorithm for Learning, Memory, and Control in Robots, Kohala Coast, Hawaii, pp. 148 – 151.

Rohrer, B. & Hulet, S. (2006). BECCA - A Brain Emulating Cognition and Control Architecture, *Technical report*, Cybernetic Systems Integration Department, Univeristy of Sandria National Laboratories, Alberquerque, NM, USA.

Tani, J., Ito, M. & Sugita, Y. (2004). Self-Organization of Distributedly Represented Multiple Behavior Schemata in a Mirror System : Reviews of Robot Experiments Using RNNPB, *Neural Networks* 17: 1273–1289.

Wolpert, D. M. & Kawato, M. (1998). Multiple paired forward and inverse models for motor control.



Robotic Systems - Applications, Control and Programming
Edited by Dr. Ashish Dutta

ISBN 978-953-307-941-7
Hard cover, 628 pages
Publisher InTech
Published online 03, February, 2012
Published in print edition February, 2012

This book brings together some of the latest research in robot applications, control, modeling, sensors and algorithms. Consisting of three main sections, the first section of the book has a focus on robotic surgery, rehabilitation, self-assembly, while the second section offers an insight into the area of control with discussions on exoskeleton control and robot learning among others. The third section is on vision and ultrasonic sensors which is followed by a series of chapters which include a focus on the programming of intelligent service robots and systems adaptations.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen