A Framework for On-Line Learning of Plant Models and Control Policies for Restructurable Control

Spiridon A. Reveliotis, *Member, IEEE,* and Mieczyslaw M. Kokar, *Member, IEEE*

Abstract— **In this paper a learning framework to deal with restructurable control of a single-output dynamic plant is proposed. The central concept used to represent the restructurable behavior of the plant, and subsequently for the design of the framework, is the *behavioral graph*. The nodes of this graph correspond to possible local behaviors of the system while its edges model the switching scheme of the plant among its local behaviors. In the definition of this concept, General Dynamical System theory is used. The framework is able to learn the dynamics (models) of a reconfigurable system, select appropriate models, and ultimately control the plant according to given specifications. The framework design borrows concepts and techniques from the active fields of adaptive and learning control. The underlying ideas and the software prototype implementing the framework design are tested through a series of simulated experiments. The simulations demonstrate the feasibility of the approach for controlling plants with unexpectedly and structurally changing behaviors in moderately noisy environments. They also identify a number of constraints that have to be satisfied for successful operation of the framework. This paper also discusses further validation of the approach, real-time application issues, and potential enhancements of the framework's functionality.**

Keywords— **learning control, restructurable control, machine learning.**

I. Introduction

Restructurable control is a relatively new paradigm in the design and implementation of control systems [1], [2], [3]. Two main forces have been driving its development. (1) The need for controlling plants that change their dynamics *structurally* in an unpredictable fashion, meaning that at different points in time, the dynamic model of the plant has to be described by equations having different variables and different mathematical operators. (2). The need for dealing with failures in the controller itself, meaning that a *restructurable (reconfigurable) controller* that changes its parameters as well as its structure in order to compensate for a structural change (e.g., a failure) in the plant or the control system itself [4] needs to be used.

The general guideline for dealing with the issue of restructurable control is to include some *redundancy* in the design of a controller. Redundancy can be either *parallel* or *analytical*. Parallel redundancy is the duplication of the controller hardware (sensors, actuators, digital circuitry) in order to decrease the overall probability of system failure. Analytical redundancy covers *failure detection*, *failure identification*, and *control system reconfiguration* (collectively referred to as *FDIR*).

There are several control paradigms that are used to deal with the issues of structural changes and failures: *robust control* [5], *gain scheduling* [6], *expert control* [7] and *learning control* [8], [9], [10]. Although each of these approaches has made significant progress in the recent years, they all make explicit presumptions about the expectation of changes in the behavior of the plant and the controller. Our goal is to ease the need for such presumptions. Machine learning is the technique that can be used in achieving such a goal.

Learning in control can be applied in different ways and in different forms [4], [9]. *Connectionist learning* (neural networks) can be used to learn nonlinear models of plant dynamics (*forward plant dynamics*), control policies (*inverse plant dynamics*), and decision policies that assign observed patterns (measurement vectors) to plant operational modes. *Symbolic (inductive) learning* [11] can be employed for the extraction and modification of reconfiguration rules. All these methods are characterized by a high computational complexity and therefore are very slow. Nevertheless, we believe that the combination of progress in machine learning algorithms and speed of computation shows promise of the applicability of these methods in future control systems.

In this paper we propose a framework to be used as a *learning controller for restructurable control*. The design does not address the issue of failure of the controller; in some cases it is possible to incorporate the dynamics of the most error-prone controller interface components, i.e., sensors and actuators, in the plant model. The non-failing controller assumption leaves out design issues related to the hardware configuration, for example the incorporation of parallel redundancy, and reduces the design problem to the matter of *concurrent learning of plant models and control policies for every possible operational mode of the plant, as well as a selection policy for selecting one particular model/control policy every time a control action has to be determined.* The selection policy corresponds to the failure detection/identification functions required by the analytical redundancy concept of restructurable control, while the plant model/control policy corresponds to the system reconfiguration.

In the next section we present the restructurable control

Manuscript received ...; revised ... . This work was performed at Northeastern University and was partially funded by the NSF grant IRI-8915057.

S.A. Reveliotis is with the Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign.

M.M. Kokar is with the Department of Industrial Engineering and Information Systems, Northeastern University, Boston.

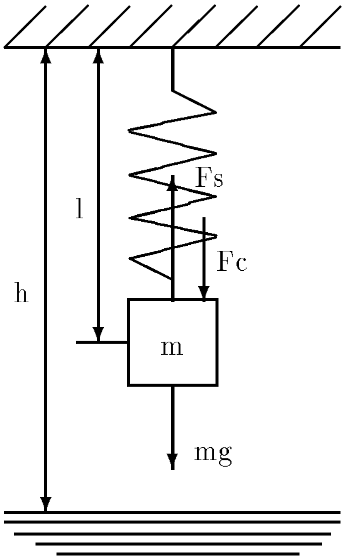

Fig. 1. A restructurable control scenario.

scenario to be used in this paper. Some explicit assumptions incorporated in our design are also stated there. Section III proceeds to the formal statement and the analysis of the problem undertaken; the underlying concepts and problem characteristics are introduced through the study of the scenario outlined in Section II. The proposed framework design is developed in Section IV. The main algorithms implemented in the framework, including model selection, model learning, and model managing, are described in Section V. Section VI presents a series of results obtained by studying the performance of our design through simulation of the scenario of Section II. Finally, in Section VII, we draw conclusions and highlight directions for future research.

II. A Restructurable Control Scenario

Consider the system [12] presented in Figure 1. This system consists of a mass m pending from a spring fixed at its upper end with spring constant k and length l_0. Below the mass there is some fluid of viscosity c, the surface of which lies at a distance h from the fixed point of the spring. If no other external force is applied to the system, the forces exerted upon the mass m are the gravitational force $W = mg$, and the reactive force of the spring F_s. The latter is given by Hooke's law: $F_s = k(l - l_0)$. Thus, without any other external force, the mass remains at rest at a distance $l = mg/k + l_0$ from the fixed point of the spring.

We want to control this system by exerting on the mass a control force F_c, so that it follows the following trajectory. The mass starts from a predefined initial state, i.e., position y_0 and velocity $\dot{y}_0$, and then follows a trajectory between two points, p_1 and p_4, as indicated by the arrows in Figure 2. The point p_1 lies above the liquid, while p_4 is in the liquid. Thus, during its motion, the mass dynamics are described by two different differential equations:

$$m\ddot{y} + ky = F_c + mg, \quad y \le y_s, \ (y_s = h - l_0), \tag{1}$$

for the part of the motion outside of the liquid, and

$$m\ddot{y} + c\dot{y} + ky = F_c + mg \ , \quad y \ge y_s, \tag{2}$$

for the part of the motion inside the liquid. The values of the parameters introduced in the above description that were used in our simulation are given in Table I.

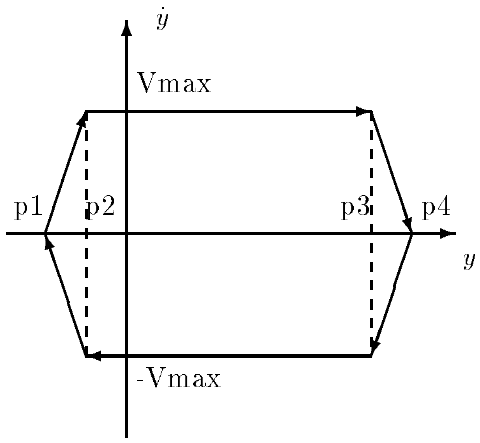

Fig. 2. The control scenario (state-space description).

TABLE I
The parameters of the simulation scenario.

Parameter	Value	Parameter	Value
m	10.0	p_1	-5.0
g	10.0	p_2	-3.0
k	100.0	p_3	8.0
l_0	10.0	p_4	10.0
c	60.0	v_{max}	10.0
h	15.0	T_s	0.05

In addition to following the trajectory described above, the controller's goal may be to halt the mass in the position at which it was when the `STOP` command was sent to the controller, and to resume the motion upon initiation of the `RESUME` command. This feature intends to capture the impact of the human interaction with the controller on the learning performance of the controller. Furthermore, to accommodate some additional restructurable control aspects, we assume that certain failures can occur in the plant during its operation. These are the breaking of the spring, which eliminates the term $k \cdot y$ from the previous differential equations, and the addition/removal of some amount of liquid, which varies the range of applicability of the plant models. The important aspects that we wanted to capture in this scenario are summarized in the following points:

1. **Local time-invariant continuous models.** The plant can be described by a number of time-invariant

continuous models, which are *local* in the sense that they are not applicable over the entire state space of the system. The transition from one model to another entails an abrupt change (discontinuity) in the plant behavior (*hard nonlinearity*.)

2. **Overlapping models.** These models may overlap in the system state space. E.g., if the spring breaks, a new pair of models describing plant behavior apply to the same region of the system state space.
3. **The lack of analytic models.** Only system variables and parameters are known, but not the relationships among them. The controller has to derive empirical plant models and control policies using measurements. This establishes the need for learning.
4. **Sustained modes of operation.** The system stays in one mode of operation (one model) for a finite, but significant, period of time, and switching times are much shorter than these sustained modes of operation. This assumption is important for the learning system, because it has enough time to learn a new model.
5. **Measurable states and permanent validation of models.** The controller is working in discrete time and has access to measurements of the entire system state.
6. **Access to a knowledge base.** The controller has access to a knowledge base that contains some a priori information about the plant useful for constructing/updating the models and for controlling the plant. This implements the *bias* [13] of our learning algorithms.
7. **The existence of a universal stabilizing conventional controller.** In order to ensure the survivability of the system during autonomous operation, we assume the existence of a conventional controller, capable of stabilizing the system over its entire operational range. Although this assumption is a relatively strong one, and it is probable that its satisfaction is not always possible, it seems an acceptable price for the presumed almost complete initial ignorance of the controller about plant dynamics.

III. Representation of Restructurable Behaviors

Following is a formal representation of the behavior of the restructurable control system described in the previous section. In order to make both the terminology and the results applicable to every restructurable control system, we express all our claims in terms of General Dynamical System Theory [14].

A. Basic Formalism

Consider a single-output time-invariant dynamical system with n states and m inputs, sampled with a sampling period T_s. Furthermore, suppose that the system is kept under constant excitation during a sampling period. The observed sampled output of such a system is a function of the system's initial state and input [14]:

$$\varphi : \Re^n \times \Re^m \rightarrow \Re,\ \ y = \varphi(x_0, u).$$

We will refer to the space $\Re^n \times \Re^m$, representing vectors of states and inputs, as *generalized system space*. The space $\Re^n \times \Re^m \times \Re$ will be called the *behavioral space* of the system. The function φ will be called the *d-response function* (discretized response function).

The d-response function is a sufficient representation of the observed behavior of a discretized single-output, time-invariant system [14]. It defines a hypersurface in $\Re^{n+m+1}$ which will be called the system *behavioral hypersurface*. The points of the behavioral hypersurface are real-valued $(n+m+1)$-dimensional vectors. In this framework, learning a model for the behavior of a dynamical system through sampling is equivalent to learning an approximation of the d-response function.

B. Spatial Restructurability

For the mass-spring system described in Section II, define the system state as $x_1 = y$, $x_2 = \dot{y}$. Furthermore, consider the following system *read-out* function

$$y(t) = r(\mathbf{x}(t)) = x_1(t). \tag{3}$$

For both regions ($y \le y_s$ and $y \ge y_s$), the system behavior is described by differential equations with fixed parameters in time. Therefore, according to General System Theory [14], when discrete control is applied, the observed behavior of the considered system in every single area of operation is time-invariant. Accordingly, every local behavior can be described by a d-response function in the system's behavioral space. For this system a closed-form of the d-response functions can be obtained by integration of the differential equations for some initial conditions $y(0) = y_0, \dot{y}(0) = \dot{y}_0$, and replacement of t by T_s in the solution. The resulting forms for y_1 are given below.

- Undamped model (outside liquid):

$$y_1 = \varphi_1(y_0, \dot{y}_0, F_c) = y_0 \cos(\omega_1 T_s) + \dot{y}_0 \frac{\sin(\omega_1 T_s)}{\omega_1} + F_c \frac{1-\cos(\omega_1 T_s)}{k} + mg\frac{1-cos(\omega_1 T_s)}{k}, \tag{4}$$

with $\omega_1 = \sqrt{k/m}$.

- Underdamped model (inside liquid, $4km > c^2$):

$$\begin{aligned} y_1 = \varphi_2(y_0, \dot{y}_0, F_c) = \\ y_0 e^{\zeta T_s}[\cos(\omega_2 T_s) - \frac{\zeta}{\omega_2}\sin(\omega_2 T_s)] \\ +\dot{y}_0 \frac{e^{\zeta T_s}\sin(\omega_2 T_s)}{\omega_2} \\ +F_c \frac{1}{k}\{e^{\zeta T_s}[\frac{\zeta}{\omega_2}\sin(\omega_2 T_s) - \cos(\omega_2 T_s)] + 1\} \\ +mg\frac{1}{k}\{e^{\zeta T_s}[\frac{\zeta}{\omega_2}\sin(\omega_2 T_s) - \cos(\omega_2 T_s)] + 1\}, \end{aligned} \tag{5}$$

The applicability of the equations (4) and (5) is limited by the critical value y_s of the liquid-surface position, i.e., the point where the mass either enters or leaves the liquid. This critical value determines a hypersurface in the generalized system space that partitions the space into regions of *qualitatively different behaviors* [15]. The system behavior in each of the regions is time-invariant. However, different models apply in each of the regions, and the controller may need to restructure its control law when the system switches from one region to another. We call this kind of restructurability *spatial restructurability.*

For our mass-spring system these regions are described by the following constraints:

- undamped model

$$(y_0 \le y_s) \ \wedge \ (y_1 \le y_s), \tag{6}$$

- underdamped model

$$(y_0 \ge y_s) \ \wedge \ (y_1 \ge y_s). \tag{7}$$

Models 4 and 5 cannot be used for predicting the system's next state if the system crosses the liquid surface during the sampling period. The regions of the generalized system space where these crossings take place are delineated by the following constraints, which are complementary to the set of constraints described above:

$$(y_0 \le y_s) \ \wedge \ (y_1 \ge y_s), \tag{8}$$

$$(y_0 \ge y_s) \ \wedge \ (y_1 \le y_s). \tag{9}$$

These constraints express the conditions for transition from one local behavioral model to the other. Equation (8) expresses the transition from the undamped to the underdamped model, while equation (9) expresses the transition from the underdamped to the undamped model.

In the system's behavioral space, the constraints (6, 7, 8, 9) have the effect of splitting the system's behavioral hypersurface into a number of *patches.* Each patch belongs to one particular behavioral mode and spans over the region of applicability of that mode in the generalized system space.

C. Temporal Restructurability

A behavioral space extended by the specification of all system parameters, models, and constraints on the applicability of particular models will be called a *configuration.* Spatial restructurability involves changes of the model within the same configuration. There is another type of restructurability that results from the variation of the system structure over time, which we will call *temporal restructurability.* As an example of the latter, consider the same mass-spring system, but presume that at some point in time the spring breaks. In such a case, the system behavior has to be described by a new set of differential equations:

$$\begin{aligned} m\ddot{y} &= F_c + mg \ , \quad y \le y_s, \\ m\ddot{y} + c\dot{y} &= F_c + mg \ , \quad y \ge y_s. \end{aligned} \tag{10}$$

The solution of these differential equations results in the following d-response models:

- no friction (outside liquid):

$$y_1 = y_0 + \dot{y}_0 T_s + F_c \frac{T_s^2}{2m} + mg\frac{T_s^2}{2m}, \tag{11}$$

- viscous friction (inside liquid):

$$\begin{aligned} y_1 = y_0 + \dot{y}_0 \frac{e^{\zeta' T_s} - 1}{\zeta'} + F_c \frac{1 - e^{\zeta' T_s} + \zeta' T_s}{\zeta' c} \\ + mg\frac{1 - e^{\zeta' T_s} + \zeta' T_s}{\zeta' c}, \end{aligned} \tag{12}$$

with $\zeta' = -c/m$.

The constraints expressing the locality of these models are identical to those used in the unbroken-spring case. Model equations (11, 12), together with the constraint set (6, 7, 8, 9), establish a new system configuration.

Temporal restructurability of the system can be visualized as switching among a number of system configurations. There are many ways in which this switching can occur: from fast random switches like in the case of component failures, to slow and predictable, as in the case of parameter drifting. The major implication of these kinds of temporal restructurability is that the behavior of the system over a longer period of time cannot be described by a single d-response hypersurface (function) composed of a number of non-overlapping patches. Instead, the behavior is described by a number of (overlapping) hypersurfaces over the same generalized system space. In a sense, the system loses its *functional* property, since different functions describe its behavior at different times over the same points of the generalized system space.

D. Behavioral Graphs

In order to provide a visual representation for the structure of the d-response of a locally time-invariant restructurable control system, we introduce the concept of the *behavioral graph.* An example of a behavioral graph is given in Figure 3. It is a 3D-graph, layered in a number of planar subgraphs. Every planar subgraph corresponds to a possible configuration of the system. The nodes in one plane are associated with the local models of a system configuration. We define the *node content* to be the functional description of its corresponding local model. The edges of the graph express the possible transitions among the local models. We define the *edge content* as the set of points in the system behavioral space that correspond to the transition expressed by the edge (transition patch). The topology of a planar subgraph expresses the spatial restructurability of the system, while the interconnections of the planar subgraphs express the system's temporal restructurability.

As we show in the rest of this paper, the behavioral graph structure is very useful for designing a restructurable control architecture. The topology of the graph can be utilized to search for new models when a change of behavior is detected. This is due to the fact that more often than not, switching to a new model is caused by the change in only one constraint (spatial) or only one parameter (temporal) at a time. Such changes can be traced by following the transition edges of the graph, rather than by employing an exhaustive blind search of all possible models. In addition, the behavioral graph can be easily changed to represent either coarser or finer models of the system. In other words, behavioral graphs support model abstraction.

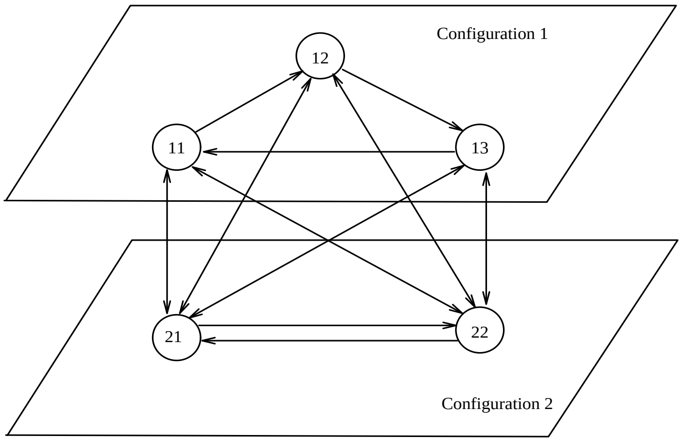

Fig. 3. A behavioral graph.

IV. Framework Design and Implementation

The behavioral graph is the central structure around which we designed our framework for restructurable control. Since we presume we do not have the knowledge of the models of possible future behaviors of the plant, both the node and the edge contents of the behavioral graph have to be learned on-line. A complete design solution must provide: (1) a representation of the behavioral graph (node and edge contents), (2) algorithms for learning the node contents of the behavioral graph (models), (3) algorithms for learning the edge contents (model switching policy), (4) algorithms for deciding whether to use one of the existing models or learn a new one, (5) algorithms for the maintenance of the behavioral graph, (6) algorithms for controlling the plant, (7) algorithms for selecting control goals, (8) an architecture that integrates all of these algorithms.

Although the framework presented in this paper addresses most of the above requirements, automatic learning of the node contents and the edge contents is our primary focus. Learning the node content is a function approximation problem [8]. Learning the edge content reduces to partitioning the observed behavioral vectors into a number of sets (clusters). Although many algorithms exist for the solution of each one of these problems, the combined problem is an *ill-conditioned* one. This is because both the node and the edge contents are learned at the same time, and the feedback information used for this learning is not sufficient for determining whether the node content, the edge content, or both, have to be adjusted and if so, to what extent. In machine learning this problem is referred to as the *structural credit assignment* problem.

One way to overcome this problem is to add more structure, or constraints, to the learning process. It is more advisable to find a set of *mild* assumptions that are likely to be satisfied in most of the situations arising in practice. The assumptions that we make in this research have been listed in Section II. From the learning system's point of view, the assumptions of local models, and of sustained modes of operation are the most important. The critical goal of our investigations is to validate these assumptions using the implemented framework.

A. Representation of the Node Content

In Section III we saw that the models describing the local behaviors of the plant are approximations of the d-response functions that represent the dependency of the plant's output on its initial state and control input. In our example, the output was the position y_1, the initial state was the initial position y_0 and the initial velocity $\dot{y}_0$, and the control input was force F_c. To represent these functions we use polynomials of the following form:

$$y_1 = \sum_{i=1}^{n_k} \theta_{ik} y_0^{a_{ik}} \dot{y}_0^{b_{ik}} F_c^{c_{ik}},$$

where a_{ik}, b_{ik}, c_{ik} are predefined integer exponents, θ_{ik} are real parameters, k is the model index, and n_k is the number of polynomial terms. The number of terms, n_k, and the exponents a_{ik}, b_{ik}, and c_{ik} are selected by the controller designer; they are treated as *design bias* (of the learning program, cf. [13]). A number of options is stored in the system's data base. We call them *model patterns.* The system, in the process of learning models, selects some of the patterns and learns appropriate coefficients θ_{ik}.

The selection of polynomials as approximators of our models was arbitrary. Other candidates that we considered were neural networks and Associative Content Addressable Memories (ACAMs) [16]. We believe that polynomials have several advantages over neural nets and ACAMs. Polynomials are easier to interpret and manipulate than ACAMs and neural nets. We use one set of polynomials for both modeling and for deriving control policies. An extra set of neural nets that learn the inverted plant dynamics would have to be used, if the connectionist representation were selected. ACAMs are able to provide the plant inverse dynamics, but they can do this at increased computational and memory cost. Furthermore, ACAMs cannot provide the insight gained by analytical modeling [17]. Polynomial approximations, being linear in the adjustable parameters, allow for employment of some adaptive control estimation algorithms, which are much faster than those used in learning [8]. Another aspect is that this representation can easily accommodate a priori information about the local models by appropriate selection of the functions implementing the polynomial terms. Any of the functions (not only monomials) of the observed variables can be used, as long as they are linear in the adjustable parameters. Finally, when the plant's behavior is sufficiently smooth, using pattern sets that are linear in the d-response independent variables can lead to the learning of the linearized plant dynamics at a number of points, which are selected automatically by the model selection algorithm. In this way, a kind of on-line gain scheduling may be achieved.

B. Representation of the Edge Content

The model-transition patches (see Section III) are strongly nonlinear small parts of the behavioral hypersurface. For that reason, they can be modeled with ACAMs [16], [18]. In our implementation, we store values of measurements normalized over the operational range; they are represented as 4-tuples $< \bar{y}_0, \bar{\dot{y}}_0, \bar{F}_c, \bar{y}_1 >$. Therefore, the normalized behavioral vectors take their values in the $[-1, 1]^4$ hypercube. The normalization is necessary to make the reasoning mechanisms, which are based on the ACAM contents, independent of the plant parameter ranges.

The ACAM representation scheme includes also point storage and retrieval mechanisms as well as a generalization mechanism, which allow it to function as a general-purpose functional approximator. The generalization mechanism is implemented through *spheres of influence*. A sphere of influence defines the area in the normalized behavioral space for which the stored point should be considered an adequate prototype for the plant behavior. The radius of the sphere of influence is user-dependent and should be determined on the basis of the prior knowledge about the plant dynamics and the required accuracy of modeling. In our simulations it was set to 0.05.

Since ACAMs are to represent the edge contents of the behavioral graph, each of the ACAM elements has to be connected to both an outgoing model (the tail) and the incoming model (the arrowhead). To implement the former, each of the elements is attached to a local model. To implement the latter, a pointer to an incoming local model is associated with each ACAM element.

C. Local Models

A *local model* is a data structure whose basic components are a polynomial representing a time-invariant local behavior, and a collection of ACAM elements representing the transition patches associated with this model. In addition to this, the local model data structure holds auxiliary data needed by either model learning or model managing algorithms. The entire set of local models constitutes the *local model database*, which provides a sufficient representation for the entire behavioral graph.

D. The Framework Architecture

The architecture of the framework (Figure 4) is is a modification of the COPER/IC architecture for learning control, which was proposed in [9]. The architecture consists of a number of modules (Unix processes) communicating over sockets.

Goal-Reasoner specifies the reference input(s) for Controller, every time a control action has to be taken. For the mass-spring system it gives the desired plant position $y_d(t)$ for the current time, as well as the desired plant position $y_d(t + T_s)$ for the next time instance.

Controller generates control input at each step (in our example it is the control force F_c). In this implementation the controller has two operational modes. (1) A discretized PID feedback control stabilizes the closed-loop system during the periods that a new plant behavior is learned by Model Learner. (2) When Model Learner provides a model of the current behavior, $y_1 = \phi(y_0, \dot{y}_0, F_c)$, Controller determines the appropriate control input for a given initial state and a desired next position by solving this equation for F_c using Newton's method [19]. In our experiments, we

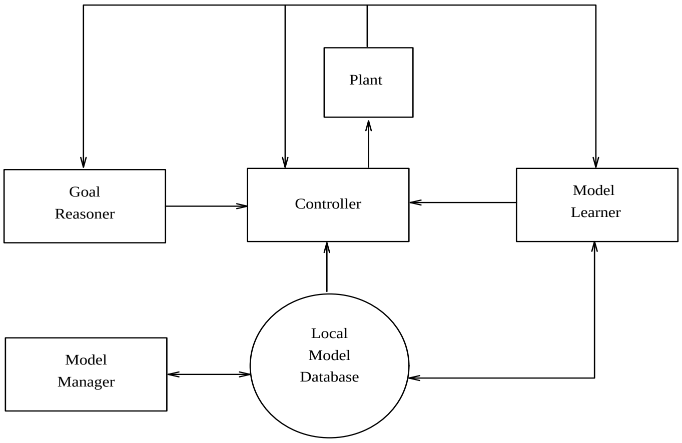

Fig. 4. Restructurable learning control architecture.

terminate the iterations of Newton's method after an error smaller than 0.1 is achieved.

Plant simulator accepts the controller input and computes the plant's next state by integrating the system state equations, using the 4th-order Runge-Kutta algorithm with $n = 10$ steps [19]. Gaussian noise $N(0, \sigma^2)$, with controlled variance σ^2, can be added to the plant output. Various random events, like the breaking of the spring or the relocation of the liquid surface, can be read in from a setup file.

Model Learner is responsible for learning the plant's local models and selecting from the Local Model Database a local model to be used by the Controller module. Since the design of algorithms for the operation of this module is at the core of this paper, its functionality is described in the next section.

Model Manager is responsible for the maintenance of the Local Model database. It removes and merges local models. This module is also briefly described in the next section.

Local Model Database contains models learned by Model Learner. The structure of this database was described in Sections IV-A, IV-B and IV-C. The contents of Local Model Database are stored and used in subsequent runs of the simulation.

Within the main feedback loop, Controller reads outputs from Plant and reference signals from Goal Reasoner, and generates control inputs for Plant. Model Learner works concurrently. At every time-step, it reads inputs and outputs from Plant (in our simulation this is a 4-tuple $< y_0, \dot{y}_0, F_c, y_1 >$) and tries to match the current Plant behavior to one of the learned models. The model prediction error is used as the matching criterion in the process of selecting a particular model. If a model is found, a pointer to this model is sent to Controller. Otherwise, the process of model learning is invoked. In that case, the controller is switched to unmodeled operation, i.e., the conventional feedback controller is used.

The design of this architecture is based on the idea of extending the adaptive controller by adding to it only the elements that are necessary to implement the functionality stated in the design requirements [9]. The loop consisting of Controller and Plant corresponds to the conventional feedback control loop, while Model Learner, Model Manager and Goal Reasoner play the role of the expert that learns, selects and maintains plant models and goals to be used by Controller.

V. Learning of the Behavioral Graph

A. Model Selection

Model Learner performs two functions: model selection and model learning. The basic scheme of the model selection function is implemented as the *nearest-neighbor* clustering algorithm [20]. Clusters are represented by the approximating polynomials $\hat{y}_{1k} = \sum_{i=1}^{n_k} \theta_{ik} y_o^{a_{ik}} \dot{y}_0^{b_{ik}} F_c^{c_{ik}}$, where k is the cluster's index. An *average prediction error*, $\bar{\epsilon}_k$, is associated with each cluster (cluster quality). It is updated by the model learning algorithm.

The observed behavioral vectors of the plant, $\mathbf{x} = (y_o, \dot{y}_0, F_c, y_1)$, are classified as members of the k-th cluster if either of the following two conditions is fulfilled: (1) the *absolute prediction error* $\epsilon_k = |y_1 - \hat{y}_{1k}|$ is less than the average prediction error $\bar{\epsilon}_k$, or (2) $(\epsilon_k < t_l \bar{\epsilon}_k) \wedge ((|\Delta\epsilon_k|/\bar{\epsilon}_k)^2 < t_s)$. t_l and t_s are design parameters. In our simulation they were equal to 10.0. $(|\Delta\epsilon_k|/\bar{\epsilon}_k)$ is a kind of "derivative" of the absolute error signal; its large value indicates that the model might have changed. This technique falls into a broader class of techniques used in exploratory data analysis – *detection-of-anomaly* techniques [21].

If the above conditions are violated, a search for a new model is initiated. In such a case, the currently observed (normalized) behavioral vector is matched against the vectors stored in the working model's ACAM. If a match is found, the pointer to the incoming model in the matching vector is used to determine the next working model. Otherwise, the working model remains the same. The Model Learner, however, is set in an *alert* status. This makes the model selection scheme more robust against shot noise effects. If, in the next cycle, an increased error is also detected, then a search for a new model in the Local Model Database is performed. If no valid model is found, the model learning process is initiated.

B. Model Learning

Learning models involves learning polynomials representing local models, updating average prediction errors associated with particular polynomials, and updating the ACAMs representing transitions among models.

B.1 Learning Local-Model Polynomials

As mentioned earlier, with each learned local model, i.e., approximating polynomial, we associate the average prediction error $\bar{\epsilon}_k$. This average is updated at every cycle that an observed behavioral vector is assigned to cluster k according to the rule: $\bar{\epsilon}_k = \alpha\bar{\epsilon}_k + \beta\epsilon_k$, where α, β are predefined weights. In our simulations we used $\alpha = 0.8$, $\beta = 0.2$ during the learning phase of a model, and $\alpha = 0.5$, $\beta = 0.5$ during the operational phase.

Learning a polynomial involves two kinds of decisions: selecting one of the predefined patterns that are stored in the Local Model Database, and training a polynomial by adjusting its parameters θ_{ik}, $i = 1, \ldots, n$. For the training of the polynomials *Recursive Least Squares* (RLS) algorithm [22] is used. Selection of a model pattern is done through *competitive learning*. Every time the need to learn a new model arises, the system trains concurrently all patterns in the Local Model database until one of them converges. The convergence criterion is that the model's average prediction error becomes smaller than a threshold value t_c ($\bar{\epsilon}_k < t_c$), which in our simulations was $t_c = 0.1$.

Once a local model has converged, it enters the Local Model Database and it can be selected as the valid model to be used for control. Every time the model is selected, the model parameters are also updated (by using RLS), provided that the model's average prediction error is not smaller than a threshold δ_ϵ. This thresholding is introduced in order to guard against the effects of noise (numerical or in measurements) that could destabilize the algorithm. In our simulations we set $\delta_\epsilon = 0.001$.

If none of the existing polynomial patterns is able to sufficiently model the plant behavior, RLS will fail to converge. The Model Learner recognizes this failure observing a high value of the average prediction error over an extended training time period. The implementation of the training algorithm includes an upper bound u_s on the number of training steps before convergence is achieved by one of the model patterns. The value of u_s should be selected on the basis of the most complex of the training patterns (worst-case selection). An aid in the selection of the appropriate u_s value is the remark (cf. [22]) that the RLS algorithm converges in the mean square in about $2m$ iterations, where m is the number of the regression variables. In general, this value should be larger than $2m$. On the other hand, a large value of u_s can be used only if the plant does not switch its behavior (its models) too often. Too frequent switching of models would result in training the same polynomial pattern on vectors belonging to different clusters, and would, therefore, invalidate learning. To avoid this kind of situation, in Section II we explicitly introduced the assumption of "suspended modes of operation", which guarantees that the time the plant remains in one kind of behavior (one model) is large enough, i.e., sufficient for the convergence of the learning process. Another way to deal with the convergence problem is by increasing the sampling rate T_s. The drawback of this solution is a high computational demand. In our simulations a value of $u_s = 40$ was used, although even $u_s = 20$ was found to be satisfactory.

The RLS training algorithm used for the learning of models has the *forgetting factor* parameter $\lambda < 1$. In our simulations we set it to $\lambda = 0.95$. This gives the RLS algorithm a *tracking* capability (following slow variations in the model parameters). For example, a linear model can model the behavior corresponding to a sufficiently smooth nonlinear surface by tangently gliding its plane along the surface. However, to achieve this tracking effect of the model parameters, a higher value of the proximity threshold t_l must be used.

B.2 Learning the Plant's Switching Scheme

In Section V-A, we specified the conditions under which Model Learner is set in an *alert* state. In this case, the currently observed behavioral vector is kept as a candidate pattern to be stored in the current model's ACAM. But at this point the information on the next model is lacking. This information is sought in the next cycle, provided that an increased error is detected again. In such a case a search for a new working model in the Local Model Database is performed. If a valid model is found, the kept behavioral vector is stored in the previous model's ACAM, with its pointer indicating the new working model. The behavioral vector is stored only if it cannot be predicted as a generalization of the vectors which are already stored in the model's ACAM, i.e., if it either does not belong in the sphere of influence of one of the previously stored vectors or if it points to another model.

B.3 The Model Selection Algorithm

In Table II we summarize the previously described algorithm for model selection. On each pass through the loop the algorithm reads the input vector and selects one out of five clauses, depending on the outcome of the previous pass. The system starts with either an empty Local Model Database or with a previously learned set of local models.

TABLE II
THE MODEL SELECTION ALGORITHM.

```
procedure model − selection();
begin{procedure}
unmodeled:=false;
if Local − Models = empty
   learning := true; alert := false
else
   learning := false; alert := true;
repeat-until exit − signal
input(current − vector);
update(variable − ranges, current − vector);
if unmodeled = true
   store(current − vector, Unmodeled − Transitions)
else if learning = true
   RLS(cur − vect, modl − patrns, best − modl, ē_k, iters);
   if ē_k < t_c
      store(best − model, Local − Models);
      working − model := best − model;
      learning := false
   else if iterations ≥ u_s
      learning := false; unmodeled := true;
else if valid(working − model)
   alert := false;
   if ē_k > δ_ε
      RLS(current − vector, working − model, ē_k);
else if alert = true
   alert := false;
   search(Loc − Modls, work − modl, cur − vect, ē_k, match);
   if not match
      learning := true;
      RLS(cur − vect, modl − patrns, best − modl, ē_k, iters)
   else if working − model ≠ previous − model
      store(prev − vect, work − modl, ACAM(prev − modl));
else
   previous − model := working − model;
   search(ACAM(work − modl), best − modl, cur − vect, ε_k, match);
   if match
      working − model := best − model
   else
      previous − vector := current − vector;
   alert := true;
send(working − model, Controller);
end{procedure}.
```

C. Model Manager

Since the clustering algorithm is based on a heuristically determined thresholding operation, it is subject to error. If the threshold values t_l, t_s are too large, the algorithm will fail to recognize some switches in the plant's behavior, which will have a deteriorating effect on the quality of the modeling and thus, on the quality of control. On the other hand, very small values for the thresholds, make the algorithm very sensitive to the intrinsic variation of the prediction error of the RLS, or to noise (numerical or external). As a result, the Local Model Database contains a greater number of models than is necessary. Some of them are duplications of others (initially perceived as new ones but after some training they converged to the same values), while some others are spurious models that were started due to a wrongly perceived switching. In general, it is expected that the redundancy resulting from too small thresholding values is preferable to the deterioration of performance due to too large thresholds, since the spurious models can be recognized and *pruned* from the Local Model Database, and the duplicated models can be *merged*. These two tasks are performed in our framework by Model Manager.

Model Manager is a background process that is invoked periodically to perform the model pruning and model merging tasks. In our simulations it runs every 2000 sampling periods. A model is pruned from the Local Model Database if it has not been used for a long time and if it was not used too often in the past. Two models are merged if their predictions $\hat{y}_{11}$ and $\hat{y}_{12}$ over the plant's operational range differ by less than the modeling tolerance δ_ϵ allowed during the model training. For a more extensive discussion of the topic refer to [23].

VI. Experimental Results

In our experiments we focused on the following issues. (1) Comparison of the framework performance in a noiseless environment against the performance of a conventional control loop. (2) Performance of the framework in a noisy environment, which is the most realistic situation. (3) Investigation of fast switching among local plant behaviors to determine limits of the learning algorithm's convergence, and to evaluate the effects of increasing the sampling rate.

The scenario file used in these experiments is given in Table III. The mass-spring system starts from its resting condition, $y = 1.0$, $\dot{y} = 0$, and then it follows a trajectory which is determined by the Goal Reasoner's logic (cf. Section II), and the `STOP/RESUME` command sequence sent to the Controller as presented in Table III. This is a typical scenario for the considered control task. The graphical results presented in this chapter were produced by mon-

TABLE III
THE EXPERIMENTAL SCENARIO

starting condition:	$y = 1.0\ \dot{y} = 0.0$
Command/Event	Time
`START`	0.0
`STOP`	45.0
`RESUME`	48.0
SPRING BREAKS	50.0
`STOP`	110.0
`RESUME`	120.0
SPRING FIXED	150.0
`STOP`	165.0
`RESUME`	170.0
(SAMPLE L-MODEL DTBASE	180.0)
`EXIT`	500.0

itoring the system's performance during the first 3 *min* of simulated time. To test the hypothesis that the correct model would be picked every time, two model patterns were preloaded in the Local Model Database, *Model-Pattern 0* and Model-Pattern 1:

$$y_1 = \theta_1 y_0 + \theta_2 \dot{y}_0 + \theta_3 F_c + \theta_4,$$

$$y_1 = \theta_1 y_0^2 + \theta_2 \dot{y}_0^2 + \theta_3 F_c^2 + \theta_4 y_0 \dot{y}_0 + \theta_5.$$

A. *Comparison of the Learning Framework with a Conventional Controller in a Noiseless Environment*

In this experiment, the performance of the framework was tested against the performance of the conventional (PID) controller. Under PID control, the system exhibits increased oscillatory behavior outside the liquid, which is the region that has a viscosity value smaller than the nominal value used in the PID controller design. The oscillations are smaller inside the liquid, although the system is slower in the sense that it cannot follow the reference input at the extreme point of its oscillation.

The performance of the developed learning framework in a noiseless environment is presented in Figure 5. This figure presents the mass-spring position (actual output), its desired position (reference input), the difference between the two (control error) over time, and the Model Learner's decision about the valid model at every time instance. Models are identified by their index in the Local Model database (table). It can be seen that the system undergoes two learning periods, starting at times $t = 0.0$, when the systems starts without any knowledge of the models, and $t = 50.0$, when the spring breaks. For both of the operational modes of the plant, it successfully learns a set of local models which it uses for control (model-based control phase). During the model-based control phase, the control error is practically zero. Furthermore, when the spring is fixed ($t = 150.0$), and the plant switches back to its initial operational mode, no further learning is necessary since the relevant knowledge is already stored in the Local Model database. Note that the local models describing the plant's local behaviors are now models 14 and 4, instead of the initial set of 1 and 4. Model 1 was "corrupted" during the learning of the models corresponding to the broken-spring configuration. Note also that at time $t = 500$ only six models remained in the Local Model Database, even though a total of 16 were developed during the entire time span.

Figure 6 gives the force F_c, applied by the Learning Controller on the mass, versus time. It can be seen that this force may take extremely large values. This is the result of the Controller's switching between the conventional and model-based mode. Specifically, when Controller is switched back to the conventional mode during the Model Learner's learning phase, the initial values in the recursive formula implementing the PID control are arbitrarily reset to zero. This may incur an increased control error, and thus, a very high value for the control force. The practical consequences of this effect are that (1) the application of such high values of the control force may be practically infeasible, and (2) during that time the system may oscillate over a region that covers more than one local behavior, preventing the convergence of the learning algorithm. The design of a switching scheme that eliminates these problems is an issue for further investigation.

B. *Experimental Study of the Framework's Performance in a Noisy Environment*

The fact that an "anomaly", i.e., increase in the model's prediction error, is used as a signal for model switching poses a problem for the application of the algorithm in noisy environments. Generally, it is expected that the algorithm's performance degrades severely in the presence of noise. This may happen because it is not possible to distinguish an anomaly in the prediction error caused by noise from an anomaly caused by a switching of behavior. If the noise statistics are known, a possible way to treat the problem is by ignoring variations in the prediction error which are smaller than a number of standard deviations of the noise distribution. In this section, we discuss how the framework's performance is affected by the presence of noise in the measurement of the plant state. The degradation of the framework's performance is examined in two cases: low-level noise and high-level noise.

B.1 Low-level Noise

In this experiment, the measurements were corrupted by adding noise, which was generated according to normal distribution $N(0, \sigma_l^2)$, with $\sigma_l = 3 \cdot 10^{-4}$. This value of σ_l was selected in such a way that it was three times smaller than the admissible prediction error, i.e., the accuracy with which Model Learner recognizes new models. As was mentioned before, this accuracy is represented by δ_ϵ, which in our experiments was set to 0.001. Since the major part of the normal distribution is contained in a region spanning $\pm 3\sigma_l$ from its mean, most of the time the added noise has a value which is below δ_ϵ. For that reason we consider this noise *low*.

It was found that the system learned more local models

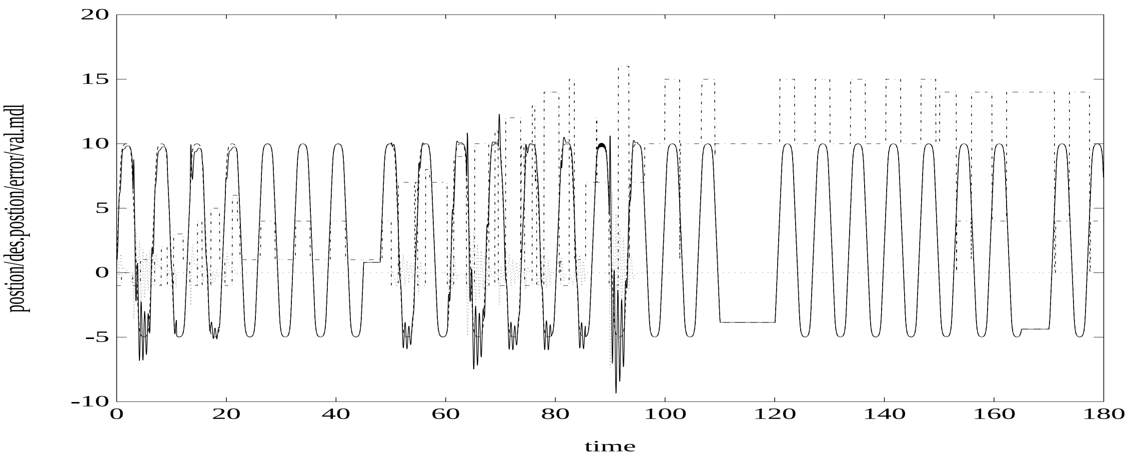

Fig. 5. Learning framework performance (no noise).

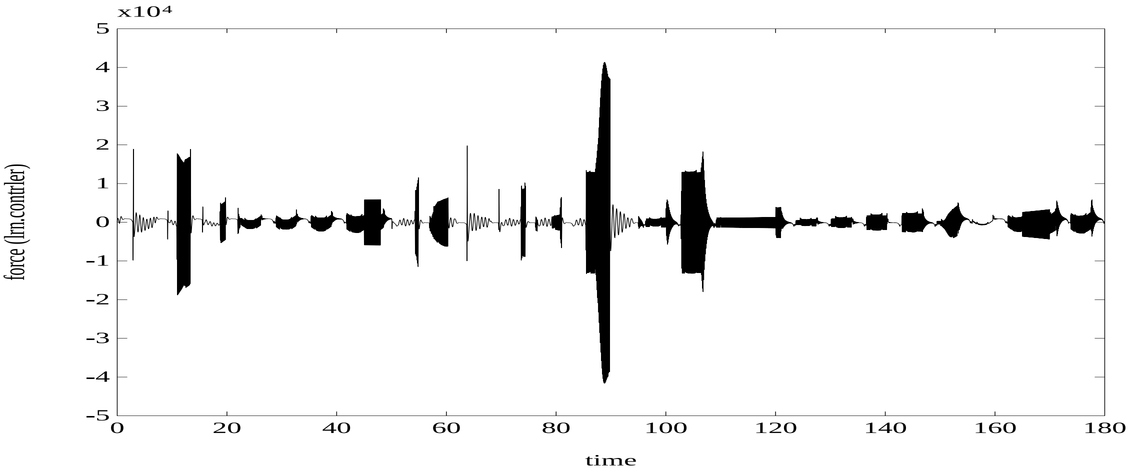

Fig. 6. Control force during the framework operation (no noise).

than necessary. As a result, the structure of the Local Models database was more complex, because one local behavior was represented by more than one local model. Nevertheless, the learned collection of models was sufficient for both predicting plant's outputs within localities, and generating control actions. In other words, in the presence of low-level noise, the framework was able to learn models representing structurally different behaviors and use these learned models to successfully control the plant.

B.2 High-level Noise

In this experiment, the added noise followed the $N(0, \sigma_h^2)$ distribution, with $\sigma_h = 3 \cdot 10^{-3}$, which was 10 times higher than in the previous case, giving noise values up to 0.01. Since this noise was higher than the algorithm's threshold δ_ϵ, it had a more severe effect on the algorithm's performance. Local models were continually trained, failing to converge to δ_ϵ. The scattering of the model parameter vectors was much broader than in the low-level noise case. Model switching was more erratic. It is highly probable that in the case of high-level noise the learning process will not converge and the system will therefore fail to model and control a structurally changing plant. We observed this kind of behavior in experiments in which we attempted to run the system for the period of 500 *sec*.

C. Fast Switching Among Local Behaviors

In this experiment, we investigated another possible limitation of the applicability of our learning framework, i.e., the rate of switching among the local plant behaviors. In our simulation scenario, the plant parameters were fixed (Table III), and therefore the switching rate was determined by the maximum velocity v_{max} of the control specifications.

When the value of v_{max} was doubled to $v_{max} = 20.0$, we obtained satisfactory performance for the established set of framework parameters. However, when v_{max} was increased to $v_{max} = 50.0$, the model selection/learning algorithm failed. In such a case, as we were able to see in our experiments, models were trained with samples from both of the local behaviors of the plant, and as a result, they failed to converge to the desired limit $\delta_\epsilon = 0.001$. Thus, they were never used by Controller. Even when we relaxed the threshold to the value of $\delta_\epsilon = 0.01$, the learning process did not converge.

As we suggested in Section V-B, a possible solution to this problem might be to decrease the sampling period T_s, so that more data points could be processed before a switch takes place. Indeed, decreasing the sampling period to $T_s = 0.025$ led to successful operation of the framework for $v_{max} = 50.0$. However, decreasing the sampling period results in smaller differences between the predictions of each of the local models. This can be seen from equations (4, 5, 11, 12), which show the dependence of the plant's output on time. Since the main principle of operation of the model selection algorithm is based upon the differences of predictions, the decrease of the sampling period T_s could lead to the loss of discriminatory power of this algorithm. This phenomenon does not allow us to use a too low value for the learning threshold δ_ϵ. In our experiments, the algorithm was able to converge for $\delta_\epsilon = 0.01$, but if failed to converge for $\delta_\epsilon = 0.001$. Evidently, the sampling period T_s and the learning threshold δ_ϵ are dependent parameters. They must be carefully selected for an application of the framework to a particular process. A more general methodology for the selection of these parameters is one of the topics for our future research. Another constraint on the plausible values of T_s is imposed by the fact that the algorithm has to operate in real time. In order to avoid high computational costs, T_s cannot be too small.

VII. Conclusions and Further Research Issues

The main objective of the research presented in this paper was to provide a proof-of-concept for an on-line machine learning approach to restructurable control. Specifically, we analyzed the problem of restructurable control of a general dynamic system and we formulated conditions that define a class of restructurable control scenarios. Then we designed and implemented a learning controller; we refer to it also as a learning framework. In addition, we designed and implemented a simulation of a plant that changes its behaviors in a structural way. We tested our learning framework on this simulated plant.

The results of our experiments with the learning framework were presented in the previous section. Our experimental objectives were twofold: (1) to show that our learning framework is capable of learning local models, recognizing changes of behaviors using the learned models, and controlling the plant, and (2) to investigate some of the limits of applicability of the framework. Towards the first aim, we were able to show that (1) in a noiseless environment, the performance of our learning framework was superior to the performance of a PID controller which was designed with the full knowledge of all the models of structurally different behaviors, and (2) in low-level noise, our learning framework was able to perform at required level of accuracy. In the second set of experiments, we identified two conditions that determine the limits of applicability of our framework. The first limit is determined by noise: the learning framework can tolerate noise of up to $\pm\sigma_l < \delta_\epsilon/3$; but when subjected to higher noise ($\sigma_l > \delta_\epsilon/3$), the framework fails. The second limit is determined by the rate of switching of plant behaviors.

The most important aspect of our learning framework is that in a potential application the knowledge of plant models is not necessary; the framework is capable of learning empirical models on-line. It should be pointed out here that since this learning framework served as a proof-of-concept, the optimization of the implementation was not undertaken in this project. We believe that it is possible to apply more efficient, streamlined and robust techniques, achieving both better performance and wider range of applicability than those obtained in this project. The evaluation of the relative merits of a series of implementational variations of the framework is on the list of our future research topics.

As presented in this paper, a number of parameters must be tuned for proper operation of the framework with a particular plant. While this fact constitutes a significant obstacle for the applicability of the learning framework for completely unknown plants, we believe that future research will bring a more generic methodology for selecting and tuning of these parameters.

The work presented in this paper can be continued in a number of directions. One of the most important goals is to develop a set of performance criteria by which this kind of learning controllers can be evaluated and compared. We believe that any effort to develop such a performance criteria set should borrow concepts and methodologies from both of the relevant fields: control theory and machine learning.

Another task that has to be undertaken is the performance of a number of experiments that will further test the framework capabilities in relevant domains. High-performance avionics, which was one of the fields to provide the motivation for restructurable control (cf. [4]), is a good testing scenario for the framework validation. Other possible areas are machine diagnostics, high-precision robotics, process control, and sensor/data fusion.

Since potential applications of this framework involve real-time processes, the development of the hardware platform able to support the real-time operation of the designed system is another issue that has to be investigated. Possible directions should include distributed parallel processing, special-purpose hardware, like VLSI systolic arrays [22], that implement adaptive filters, *k-d trees* [24] implementation of ACAMs, and others.

The application of Statistical Process Control [25] and the employment of more rigorous extrapolative forecasting techniques [26] for detection of model shifting through prediction-error analysis are also open research questions. The convergence of the learning process during the switching of the control modes must also be further investigated.

The capabilities of the framework should be extended in order to make it applicable to a wider selection of scenarios. One of these directions is to modify the framework to make it applicable to plants with more than one controlled variable, interacting through plant dynamics. Another question is how to enhance the framework's capabilities of dealing with varying goals. Meeting this requirement brings the framework's functionality closer to that of intelligent controllers [27] and hybrid systems [28]. This would involve adding some symbolic reasoning and planning capabilities to Goal Reasoner and possibly some algebraic-reasoning capabilities to Model Manager.

Acknowledgements

The authors would like to thank the anonymous reviewers for their remarks, comments and suggestions on how to improve the readability of this paper.

References

[1] R. J. Montoya, W. E. Howell, W. T. Bundick, A. J. Ostroff, R. M. Hueschen, and C. M. Belcastro, "Restructurable control", Tech. Rep. NASA CP-2277, NASA Langley Research Center, VA, 1982.

[2] P. J. Antsaklis, K. M. Passino, and S. J. Wang, "An introduction to autonomous control systems", *IEEE Control Systems*, vol. 11, pp. 5–13, 1991.

[3] J. S. Litt, "An expert system to perform on-line controller restructuring for abrupt model changes", Tech. Rep. 90-C-018, NASA, Lewis Research Center, 1990.

[4] R. F. Stengel, "Intelligent failure-tolerant control", *IEEE Control Systems*, vol. 11, pp. 14–23, 1991.

[5] M. Morari and E. Zafiriou, *Robust Process Control*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[6] W. J. Rugh, "Analytical framework for gain scheduling", *IEEE Control Systems Magazine*, vol. 11, 1, pp. 79–84, 1991.

[7] K. J. Åström and B. Wittenmark, *Adaptive Control*, Addison-Wesley, 1989.

[8] J. Farrell and W. Baker, "Learning control systems", in *Introduction to Intelligent and Autonomous Cotrol*, K. M. Passino and P. J. Antsaklis, Eds. Kluwer Academic Publishers, 1993.

[9] M. M. Kokar, "Learning control: Methods, needs and architectures", in *Introduction to Intelligent and Autonomous Cotrol*, K. M. Passino and P. J. Antsaklis, Eds., pp. 263–282. Kluwer Academic Publishers, 1993.

[10] S. Arimoto, S. Kawamura, F. Miazaki, and S. Tamaki, "Learning control theory for dynamical systems", in *Proceedings of the 24-th Conference on Decision and Control*, 1985, pp. 1375–1380.

[11] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach, vols. 1-3*, Morgan Kaufmann Publishers, Inc., 1983.

[12] M. M. Kokar and S. A. Reveliotis, "Integrating qualitative and quantitative methods for model validation and monitoring", in *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, 1991, pp. 286–291.

[13] P. Benjamin, R. Holte, and L. Rendell, *Change of Representation and Inductive Bias*, Kluwer Academic Publishers, Boston-Dodrecht-London, 1989.

[14] C. Desoer, *Notes for a Second Course on Linear Systems*, Van Nostrand, New York, 1970.

[15] M. M. Kokar, "On consistent symbolic representations of general dynamic systems", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 8, 1995.

[16] C. G. Atkeson and D. J. Reinkensmeyer, "Using associative content-addressable memories to control robots", in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., pp. 255–285. 1990.

[17] C. G. Atkeson, E. W. Aboaf, J. McIntyre, and D. J. Reinkensmeyer, "Model-based robot learning", in *Robotics Research: The Fourth International Symposium*, 1988, pp. 103–110.

[18] A. W. Moore, "Acquisition of dynamic control knowledge for a robotic manipulator", in *Proceedings of the 7th International Conference on Machine Learning*, 1990, pp. 244–252.

[19] L. W. Johnson and R. D. Riess, *Numerical Analysis*, Addison Wesley, 1982.

[20] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[21] W. R. Green, *Computer-aided data analysis: a practical guide*, Wiley, New York, 1985.

[22] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[23] S. A. Reveliotis, "A framework for on-line learning of plant models and control policies for restructurable control", Master's thesis, Northeastern University, Department of Industrial Engineering and Information Systems, Boston, MA, 1992.

[24] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time", *ACM Transactions on Mathematical Software*, vol. 3, pp. 209–226, 1977.

[25] R. E. DeVor, T. Chang, and J. W. Sutherland, *Statistical Quality Design and Control: Contemporary Concepts and Methods*, Macmillan Publ. Co., New York, NY, 1992.

[26] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, McGraw Hill, 1990.

[27] Chair Antsaklis, P. J., "Defining intelligent control: Report of the task force on intelligent control", *IEEE Control Systems*, vol. 14, no. 3, pp. 4–5, 58–66, 1994.

[28] M. D. Lemmon, J. A. Stiver, and P. J. Antsaklis, "Learning to coordinate control policies of hybrid systems", in *Proceedings of the American Control Conference*, 1993, pp. 31–35.

Spiridon A. Reveliotis received his B.S. in Electrical Engineering from the National Technical University of Athens (NTUA), Greece, in 1989, and his M.S. in Computer Systems Engineering / Engineering Software Design from Northeastern University, Boston, in 1992. During the last year of his undergraduate studies, he spent four months at the Computation Dept. of UMIST, Manchester, U.K., working on his graduation project, under an EEC ERASMUS fellowship.

Currently, he is with the Mechanical & Industrial Engineering Department of the University of Illinois at Urbana-Champaign, pursuing a Ph.D. in the area of the supervisory control of discrete event systems and its application to Flexible Manufacturing. His academic interests are related to the systems modeling and control by means of both classical and modern control methodologies, and the development of functional frameworks that can accommodate these methodologies, on the basis of the particular application features.

Mr. Reveliotis is a member of the IEEE, SME, and Phi-Kappa-Phi societies.

Mieczyslaw M. Kokar received his M.S. and Ph.D. degrees in Computer Systems Engineering from the Technical University of Wroclaw, Poland, in 1969 and 1973, respectively. His research interests include machine learning of physical laws, learning control, qualitative dynamics and hybrid control systems, sensor/data fusion, and software engineering. Dr. Kokar worked at the Technical University of Wroclaw (1969 - 1981) and Millersville University of Pennsylvania (1982 - 1984). Since 1984 he has been with the Department of Industrial Engineering and Information Systems, Northeastern University in Boston.

Dr. Kokar served as a committee member of the IEEE International Symposium on Intelligent Control (1989 - 1994) and was the Publications Chair of this conference (1991). He was also on the committee of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, and Intelligent Robotic Systems (IRS). Dr. Kokar is a member of the IEEE, AAAI and ACM.

List of Figures

List of Tables