Higher Level Application of ADP: A Next Phase for the Control Field?

George G. Lendaris, Life Fellow, IEEE

Abstract—Two distinguishing features of humanlike control vis-à-vis current technological control are the ability to make use of experience while selecting a control policy for distinct situations and the ability to do so faster and faster as more experience is gained (in contrast to current technological implementations that slow down as more knowledge is stored). The notions of context and context discernment are important to understanding this human ability. Whereas methods known as adaptive control and learning control focus on modifying the design of a controller as changes in context occur, experience-based (EB) control entails selecting a previously designed controller that is appropriate to the current situation. Developing the EB approach entails a shift of the technologist's focus “up a level” away from designing individual (optimal) controllers to that of developing online algorithms that efficiently and effectively select designs from a repository of existing controller solutions. A key component of the notions presented here is that of higher level learning algorithm. This is a new application of reinforcement learning and, in particular, approximate dynamic programming, with its focus shifted to the posited higher level, and is employed, with very promising results. The author's hope for this paper is to inspire and guide future work in this promising area.

Index Terms—Approximate dynamic programming (ADP), artificial intelligence (AI), context, context discernment, experience-based identification and control (EBIC), neural networks (NNs), optimal control, reinforcement learning (RL), system identification (SID).

I. INTRODUCTION

The purpose of this paper by one of the coeditors is to bring to the attention of readers of this special issue (SI) some hopefully seminal ideas that will inspire and guide an even greater application of approximate dynamic programming (ADP) methods, building on the cumulative wealth of foundational ADP work reported in papers of this SI and other recent work, e.g., [41] and [53]. The quest of many researchers in our field(s) is to have our technology achieve more humanlike capabilities for identification and control. Human performance levels for such tasks clearly depend on effective and efficient use of experiential knowledge. While the control field has indeed accumulated remarkable achievements, even exceeding human control capabilities for some applications, still, substantial additional progress is needed toward building into machines the ability to employ experiential knowledge (hereafter called experience) when performing system identification (SID) and when coming up with a good controller for a given situation and, importantly, to do so effectively and efficiently.

This paper puts forth a notion here called experience-based identification and control (EBIC), provides a definition and some requirements for fulfilling this notion, and gives an overview of a novel concept for applying reinforcement learning (RL)/ADP at a “higher level” to accomplish EBIC.

The EB ideas of this paper are motivated via two key observations of human abilities.

1) After a human learns a set of related identification and/or control tasks, when presented with a novel task of the same genre, the human generates reasonably optimal performance on the new task (i.e., effective selection from experience).
2) The more knowledge a human attains, the speed and efficiency of performing tasks are improved [in contrast, for AI systems thus far developed, the more knowledge acquired (typically stored as “rules”), the slower the decision/action processing].

Basic components for what is meant here by human experience related to a class of identification/control tasks and posited here as fundamental for achieving the above-noted effectiveness and efficiency include the following:

1) a collection of models (of plants or controllers, depending on whether doing SID or control) that is appropriate to a given engineering application;
2) a characterization of this set of models in a form that facilitates indexing and accessing the models;
3) an Agent with an algorithm that effectively and efficiently selects a sequence of (good) models from this set as context changes occur within the application.

Implicit in these requirements is a memory property for the Agent.

To craft a definition of experience, we take note of another fundamental notion—context. Humans intuitively understand that as context changes, so do the decision rules and/or control policies used to function within the given context.

The notion of context is here formulated to comprise three components (for a control setting): 1) plant; 2) environment; and 3) objectives plus associated performance criteria [labeled as criterion function (CF), which is sometimes defined as a cost function] (see Fig. 1). Specification of all three yields a specific context; a change in any results in a different context.

Manuscript received December 7, 2007. This work was supported in part by the U.S. National Science Foundation under Grant ECS-0301022. This paper was recommended by Guest Editor F. L. Lewis.

The author is with the Systems Science Graduate Program, NW Computational Intelligence Laboratory, Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207 USA (e-mail: lendaris@sycs.pdx.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2008.918073
In this formulation, to each specific context, there corresponds a particular control law.

As an intuitive entrée to this use of the term context, think of driving on a clear afternoon on a dry or icy pavement. The driving skills in both scenarios are the same, but selected adjustments are needed to your control law and/or decision logic. If, instead of a change in the environment (road conditions), there is a change in your car’s attributes (e.g., a slightly flat tire), new adjustments are needed to the car to perform your desired maneuvers; driving your friend’s car that day rather than your own is another example. A third consideration involves performance criteria (CF), e.g., in a road race, a criterion is to minimize time, but for an elderly relative on an excursion, maximizing comfort is more likely. Each of the example conditions above may be represented via a triplet of lines from the context in Fig. 1, pointing to a corresponding control law in the repository.

The term experience as used here entails a collection of designs that have already been developed for a set of contexts from a common application domain and also entails a memory about the collection, which is here called an experience repository.

Clearly, the context discernment and controller selection processes are directly impacted by the representations crafted for context and repository and, in particular, by the indexing schemas defined for the various sets. A candidate tool for the latter is the formalism of manifolds from geometric topology, where the manifold comprises a set and a coordinate system; the manifold’s set is to comprise the experience repository, and the manifold’s coordinate system is to be a searchable indexing mechanism with useful “nearness” properties (more details in Section IV). In the author’s preliminary work, such a formulation provided the framework in which a novel concept for applying RL (called higher level learning algorithm (HLLA)) was developed for evolving the nascent EB ideas presented here. An instantiation of the HLLA to SID has been applied with success, which is called contextual RL (CRL) [13], [38]. The key idea for HLLA is to repurpose the RL method (at a “higher level”); therefore, instead of performing the usual task of designing an optimal controller for a given context—the “level” at which the RL methods are typically applied—a collection of such designs for a variety of related contexts is provided (as repository or experience), and the new design task for the RL is to develop a strategy for optimally selecting an existing solution from the repository (the focus for the RL is thus “one level up”—hence, the HLLA). The selection process is to be triggered by the Agent becoming aware that a change in context may have occurred. This is followed by the Agent seeking information about what has changed—a process here called context discernment; the latter process typically entails a form of SID, which is also enhanced via experience. Examples of successful application of the HLLA approach to SID have been accomplished (see Section VI).

A variety of existing methods of the control field may be used to generate components for the repository; those methods are here assumed available within the EB process as a means for growth of the repository.

In summary, it is posited that the following four aspects are fundamental to the EB notion:

1) context;
2) discerning current context;
3) selecting appropriate solution for the discerned context from an experience repository;
4) doing the latter two in an effective and efficient manner.

Looking ahead, notions of context and context discernment will also be fundamental for deciding what task(s) to perform in a given situation, e.g., in a football game, do I throw the ball, kick it, or run it? Clearly, optimization and hierarchy will be fundamental to such considerations; (future) development of a concept that might be called context space hierarchy would powerfully assist in this endeavor.

While some existing methods of adaptive control and learning control may be said to incorporate the idea of context, much of the process is prespecified by the control engineer (cf. Section II). The suggested EB control approach seeks to go beyond the limitations that are implicit in prespecification or, at least, to reduce their number.

Biological intelligent systems do not require predefinition of context; they learn how to discern context on their own, which is likely based on appropriate representation(s) of context. An important attribute to build into an EB Agent is the ability to learn (indeed, to develop) its own representation of context and, in addition, its own algorithm for discerning context—yielding an Agent that is context discerning, in contrast to existing methods, whose Agent may be described as (only) context dependent. A distinction here is that the context discerning Agent is to require little or no predefinition of context or how to detect it. To reemphasize, achievement of high efficiency, particularly as the experience repository grows larger and larger, will require tight coupling between the representation crafted for context, the discernment that is to be accomplished, and the controller selection process.

There is an approach in the literature that may be said to perform context discernment; this approach applies recurrent neural networks (NNs) to SID and control, primarily under the rubric fixed-weight NNs (FWNNs) [8], [10], [11], [35]. However, while the operation of these networks is rather remarkable, no principled explanation had existed for what is here called their context discerning capabilities. The analysis of such early work [10], [11], [35] at the author’s Lab yielded a principled explanation of the FWNN’s context discerning capabilities and yielded an approach that reproduces such capabilities with reduced computation [38].

The following are the main components of this paper.

1) Define EBIC, including a component notion called the HLLA, and some requirements for their realization—it
is suggested here that EBIC is to be a significantly new development phase, indeed a vision, for the field of controls.

2) As a basis for the latter assertion, give a historical overview of the control field via a perspective of how the notion of context has (or has not) been explicitly dealt with to date (Section II).

3) Provide an overview description of the method here called CRL [38] as a candidate HLLA that has a promise for (partially) fulfilling the new vision.

As an aid for the remainder of the presentation, a set of definitions is given in Table I.

II. HISTORICAL OVERVIEW OF CONTROL FIELD VIS-À-VIS EXPLICIT ROLE OF CONTEXT

Since the notion of context is fundamental to the EB approach, the author performed a historical overview of the control field vis-à-vis the explicit role that context has (or has not) played in the various formulations and approaches. This overview was also motivated by the author’s belief that adding the capability to employ experience in the controller design/selection process will usher in a qualitatively new phase in the evolution of the control field. From this perspective, the control field’s evolution is here characterized by the following four phases: 1) design based on intuition and invention; 2) design based on mathematical tools; 3) design for accommodating context variations (context dependence); and 4) design for EB processes, including autonomous context discernment and model selection.

These four phases in the evolution of the control field that is perceived from the perspective of how the notion of context has been explicitly dealt with to date are as follows.

1) Phase 1: Design Based on Intuition and Invention: Various histories of controls (see, e.g., [4]) note the existence of control devices dating back to antiquity; a relatively recent device is the well-known flyball governor invented by James Watt in 1788 [44]. It appears that the designs of these control devices were the product of intuition and inventive genius, with little support from mathematically based tools and with no explicit notion of context per se.

2) Phase 2: Design Based on Mathematical Tools: Mathematics has played a fundamental role in the development of the control field, beginning with Maxwell’s use of differential equations to analyze the flyball governor’s dynamics (see, e.g., [25, Ch. 1]), ca. 1870, progressing through Fourier and Laplace transforms, state-space methods, stochastic methods, Hilbert space methods and, more recently, algebraic and geometric topological methods. The advent of modern computers with their fantastic evolution the past few decades has also been significant, not only from the implementation point of view but also as a driver and motivator for various mathematical and algorithmic developments as well.

The distinguishing feature adopted here for this phase is that the controller, however designed, is placed in service with no associated mechanism for modifying its design in response to context changes, be they in the plant or its environment. The design is done offline and, de facto, based on a single point in the context space or, at most, a small neighborhood of points. However, in practice, the designs in this category are often crafted to have “low sensitivity” (“robustness”) to selected changes in plant or environment parameter values. While these controllers
accommodate certain context changes, this is accomplished by virtue of “margins” in controller design rather than by online changes in the design itself.

The method known as model predictive control may be placed here, as, even though it is an online optimization method that determines set point(s) for each control interval, it is based on a fixed model of the plant/process being controlled.

3) Phase 3: Design for Context Dependence: In a number of applications, the context changes so much during operation that the fixed controller designs resulting from the Phase-2 methods are not sufficient. A design path emerged that accommodates context variations via online instantiation of different controller designs based on the observed variations. However, there are important distinctions between how the knowledge of a changed context is attained and how the different controller designs and/or instantiations occur in these methods versus those to be described for Phase 4.

a) Partitioning Methods: A large segment of Phase-3 methods may be labeled as partitioning as they partition a nonlinear operating region into approximately linear regions and develop a linear controller for each. These methods may be said to focus on CONTEXT: Environment (cf. Fig. 1 and/or Definition 3). The various methods have different means of “knowing” which context is the current one. In general, once the specific current context is known, a previously designated controller or controller design process is then instantiated.

Partitioning methods have appeared in a variety of technology sectors, e.g., control theory, artificial intelligence, NNs, fuzzy logic, statistics, etc. The associated methods have understandably appeared under a variety of labels—e.g., multiple models (see, e.g., [49]–[51]), piecewise models, mixture of experts, fuzzy models, local regression, etc.

What distinguishes most of these methods from those that appear in Phase 4 is that the adjustments to controller design are (typically) prespecified by the human designers.

b) Adaptive Control and Learning Control: Adaptive [3], [15], [27], [37], [43] and learning control methods both operate over a specified pool of controllers, and they select a controller from this set based on a sequence of state (environment) observations and/or performance evaluations. In the adaptive control case, the engineer specifies a set of available controllers (via parameterized models) and an algorithm to select from this set based on observations. In the learning control case, the engineer specifies a parameterized controller structure and a corresponding algorithm to incrementally adjust the parameters as new situations are encountered. Of significance here is that these algorithms do not retain memory of solutions as they are achieved. The primary difference between the two methods lies in the amount of a priori information embedded in their respective pool of controllers and the offline versus online aspects. Apropos the latter, the adaptive control methods normally provide a guarantee that switching among the policies in the set can be done safely, for example, related to stability, in an online manner [3], [29], [37]. Historically, such guarantees were not available with RL methods (cf. several chapters in [41]), but recent extensions demonstrate selection methods that permit online operation as well (see, e.g., [2], [14], [22], [23], [30], [31], and [52]).

The author concedes that the essence of the EB ideas is, to some extent, incorporated in the aforementioned partitioning methods. Ensemble methods may also fall here since they too employ multiple models; however, a voting or merging scheme is used there, not selection. Fuzzy control approaches are also difficult to classify in the present scheme.

Nevertheless, there are sufficient distinctions among the respective guiding principles that explication and pursuit of the EB ideas are still warranted, as some of the distinctions seem crucial to the scalability issues (e.g., speed of selection versus size of the model repository) to be faced when attempting to develop human-level performance in the Agents. Whereas multiple-model methods have historically used linear models (except more recent NN ones [49]), no such constraint is involved in the proposed EB method.

4) Phase 4 (New): Design for EB Processes, Including Autonomous Context Discernment and Model Selection: The following are stipulated for this phase.

1) Agent is able to use experience for model selection (controller or plant, depending on Agent’s task).
2) Agent is able to do so efficiently and effectively.

Phase 4 is defined and developed based on the four aspects listed in Section I, plus any collateral requirements.

By the standards of today’s technology, one stands in awe at the apparent effortless context discernment and seemingly simultaneous selection of “good” actions that humans perform on a routine basis. Equally impressive is the human central nervous system’s (CNS) adjustment of its information processing based on cues available in the environment, and the appearance there is little (if any) explicit direction needed for the development of this capability. This suggests a path for achieving EB capability by our Agents, namely, to emulate the human ability to learn to discern contextual cues without explicit a priori “engineering”—as we are presently obliged to do when designing robots, such as was done in crafting the context variables for the car-steering and flight controllers to be described in Section III-B.

Another CNS capability that bears emulating is that of figuring out what the “problem” is and when to solve that problem. To date, much of the work in computational intelligence has focused on, given a problem, how to solve it. The what/when tasks have typically been left for the engineers or computer scientists to solve explicitly (cf. the gain scheduling method [28]—where an explicitly stipulated set of measurements/criteria is “scheduled” for determining the sequence of contexts). A future objective put forth for the HLLA approach is to address the what and when as well as the how questions, likely involving the application of HLLA at different hierarchical levels.

Phase 4 is here summarized via anthropomorphism—a design approach that develops an (autonomous) Agent whose operation fills the experienced designer role in the following scenario: plant, environment, and control objectives are provided to a control system designer, who is to design/select a controller. If the designer is experienced, has “seen” the situation before, and remembers the solution, after context data are gathered, he is able to pull a “good” design out of the archives and apply it to the current situation—perhaps with a little
LENDARIS: HIGHER LEVEL APPLICATION OF ADP: A NEXT PHASE FOR THE CONTROL FIELD?

III. EB Control Preliminaries

A next developmental step toward humanlike control is to come up with learning methods to achieve the following.

1) Accumulate a set of “relevant” control policies for an engineering task (as is done for adaptive control).
2) Create a representation schema that facilitates accessing the policies.
3) Create an algorithm that efficiently and effectively selects among these policies during online operation of the system in response to changes in its context.

This process entails the HLLA mentioned above, where the HLLA selects an optimal controller from the repository in an optimal way, in contrast to (simply) adjusting the existing controller’s design. Note that the two uses of the word “optimal” here refer to two distinctly different CFs: the first is for the control task, and the second is for selecting the corresponding optimal controller.

Figs. 2 and 3 provide the reader with a bird’s-eye view of the key new components entailed in the EB notions.

Fig. 2 shows a generic structure of adaptive and learning methods. The algorithm part of the adaptive and learning control methods takes observations and uses them to adjust the parameter settings of the controller (policy), some of which may be done online, with adjustments by learning control performed over a larger range than allowed in adaptive control. Historically, the learning control methods were employed for offline operation, but recent developments also allow online performance.

1) Learning Control Example: In the dual heuristic programming (DHP) method of ADP, it is common to implement the controller by a multilayer perceptron (MLP), a substantially looser specification of available policies than is done in the adaptive control method. The weights and biases of the MLP form the space of parameters for the controller/policy where the MLP’s initial weight values represent a starting policy, and each subsequent observation is the basis for adjusting these parameters using the ADP procedures (gradient descent, etc.). See [24] for details of DHP operation.

To describe some distinctions of the HLLA notion from the adaptive and learning control methods, focus on the two main aspects of the HLLA process, namely, SID (employed for context discernment vis-à-vis the plant subspace) and controller selection. Look first at Fig. 3(a), and see that the box in Fig. 2 labeled as “Adaptive Algorithm/Learning Algorithm” has been replaced with two boxes, one labeled EBSID (EB Systems Identifier) and the other HLLA. The HLLA works with a set of plant models appropriate to the given SID task, creates a representation for the (sub-) set of these models relevant to the given engineering task, and trains the algorithm embedded in the EBSID; this algorithm is given the generic name EB-Algorithm. The EBSID employs the EB-Algorithm; thus, it learned to effect the SID. In addition to the box replacement between Figs. 2 and 3(a), connection to the controller box in Fig. 1 is also removed for Fig. 3(a), reflecting the view that the system identification task does not normally feed into the controller, at least not directly.

Proceeding next to Fig. 3(b), note that the same box in Fig. 2 has again been replaced with two boxes, one labeled EBC and the other HLLA. This time, the (trained) EBSID is included as a component of the EBC. The HLLA works with a set of policies available for a given engineering task (for comments about where these come from, see Section VII), creates a representation for the (sub-) set of these policies relevant to the given engineering task, and trains the algorithm embedded in the EBC; this algorithm is again given the generic name EB-Algorithm. The EBC employs both the EBSID subprocess (as part of the context discernment activity), and the EB-Algorithm thus learned to effect the EB control—i.e., to efficiently and effectively switch between the policies in its experience repository as the context changes (this is in distinction to directly manipulating the existing policy as is done in Fig. 2). The phrase “efficiently and effectively” becomes instantiated as an optimal search trajectory in the coordinate space (see Section IV) used to index the set of (relevant) controllers.

Note that the HLLA process appears as an “offline” method that designs the EB-Algorithms used subsequently to perform EBC. More generally, HLLA could continue to perform online
and, when doing so, ought to employ a means for implementing the notion of “safe-fail,” which is often associated with nature’s ecological processes (e.g., forests have built-in mechanisms for regrowth following severe forest fires) and clearly observed in animal systems—in contrast to the “fail-safe” aspirations of many engineering systems.

Note that the controller/policy box could be disconnected from the plant and that the plant fed, instead, from a special test input generator or some combination thereof.

It follows from the aforementioned description that the performances of the EBC and the EBSID are both dependent on the EB-Algorithm developed for them, respectively, by HLLA. The term Agent may be used interchangeably with the labels EBC and EBSID in those cases where the Agent’s role is clear.

A. Conceptual Configuration of EB Control Process

Fig. 4 shows a conceptual layout of the EB control idea as it might be performed when the various aspects of the EB method are worked out. The reader is directed to the upper left corner of Fig. 4 at the place labeled “Starting Condition”; this is intended to represent the situation where a controller/plant configuration is functioning as expected in some operating environment. Moving to the right, the Agent monitors the situation to become aware of the changes that may occur. When a change is noted, the Agent goes into a context discernment mode to figure out what changes have occurred; in Fig. 4, this action is labeled Perform SID (hence, specializes the figure to a control setting with changing plant parameters). This stage yields an updated plant model, and the Agent proceeds to the EB controller selection task. Following this, the Agent runs a simulation with the new controller and plant models and assesses performance according to the CF (as a side note, the author has been informed by colleagues that humans appear to perform “rehearsing” in such situations; this motivates the “simulation” stage). If all is OK, the new controller design is uploaded to the controller box in the upper left corner of the figure; if not, the context discernment stage is entered again.

Section III-B describes two early experiments related to the context-dependent ideas where much of the above is rolled into a single NN controller; in those cases, the NN was directly trained with specialized information about the context. DHP was used for training, and the controller learned to employ the “context variable” input values to change the mapping between its “regular” controller inputs (values of plant state variables) and the control outputs. This worked very well for both scenarios [18], [21]. It is believed that substantial additional “mileage” can be obtained by using that approach (and I encourage others to pursue such studies), but the author has chosen to, instead, delve deeper into the inner workings of what the NN accomplished as a means of developing the context-dependent ideas as presented here.

The multiple-model methods mentioned in Phase 3 require care when switching from one controller to another; the same care must, per force, be considered in the present case as well (cf. “bumpless control transfer” in that literature). In the layout in Fig. 4, this comment relates mostly to the activities performed in the two boxes labeled CFA, in particular, before the “Install” action is performed.

B. Precursor Context-Dependent Controller Experiments

Two early versions of Phase-4-type systems have been demonstrated: 1) steering a car on a dry road and then encountering an ice patch [20], [21] and 2) flying a hypersonic aircraft and then encountering a sudden shift in the center of gravity (c.g.) [18], [23]. In both cases, it was reasoned that an experienced human operator, upon noticing a changed vehicle behavior (context), would invoke a “higher level” process to acquire information, for example, by sensing the vehicle’s response to small perturbations to selected control inputs to update the operator’s knowledge of environment and/or plant parameter values (an SID process), to assist in deciding what to do next. The latter could include modifying the CF being employed (e.g., emphasize safety versus timely arrival) and then selecting appropriate control actions. Guided by this reasoning, we (the human designers) developed a proxy measurement that provided data about the car/road interaction (tire slip angle) in one case (environment data) and a proxy measurement for location of c.g. in the other case (plant dynamics data)—i.e., “context variables.” What is important here is that during the adaptive-critic (AC) design process [24], [36], [42], [45], [46], the context variable was included as an auxiliary input to the controller in addition to the usual plant state variables (cf. bottom half of Fig. 5) (note that this is distinct from the hidden-state estimation notion used in the control field). In both applications, the controller learned to use a change in the value of the context variable to select a different controller instantiation—i.e., the context discernment was (externally) provided to the NN controller as a value of the context variable, and the NN controller, in essence, developed its own local repository of controllers selectable via the context input (cf. Fig. 5). What distinguishes
these precursor results from that desired for Phase-4 systems, however, is the fact that the context variables (and their respective sensors) were crafted by the control engineer. In Phase 4, a key intent is to take the human out of that part of the design loop. While crafting the context variables employed in the aforementioned examples, we consciously took into account how a human operator of each type of vehicle might acquire the additional context data. We did this “manually” as, at that time, it had not occurred to us to define a “next level up” task for the ADP method to perform a process we now know entails crafting a new and different CF; one appropriate for the new level. The aforementioned demonstrations that a context variable could be used as a controller input during RL and the controller learn to use it as a policy-selection mechanism were important precursors for the ideas presented here and, in themselves, provide a continuing arena of future research.

IV. Manifolds

An indexing scheme for the repository must be both searchable and have an operational notion of “nearness.” The mathematical construct of manifolds as defined in the geometric topology branch of mathematics provides a useful formalism, particularly for issues of representation. While the mathematical concept of manifolds entails deep mathematical properties (cf. [1]), for the present purposes, it suffices to think of a manifold as comprising the following: 1) a set of elements $S$ and 2) a coordinate system—a one-to-one mapping from $S$ to $\mathbb{R}^n$ that specifies each element in $S$ via a vector of $n$ real numbers, a.k.a., the coordinates of the element. The terms “index” and “coordinates” thus become synonymous here. The experience repository is represented by the set portion of a manifold, and the manifold’s coordinate space serves as a searchable indexing vehicle for the repository; since the coordinate space is $\mathbb{R}^n$, the Euclidean distance is a natural metric for “nearness.” As a demonstration example, let the set comprise a collection of NNs generated via an NN whose neuron type and structure (feedforward) have been specified, and its adjustable parameters (weights and biases) are made to take on all possible combinations of their respective values. Each such combination yields a distinct member of the set, and the parameter values may serve as the coordinates. Per Definition 4 in Section I, an indexed element in the manifold’s set is referred to as a point. The label neural manifold is employed in [38] and [39] for this case; an important caveat when employing a neural manifold is that care is needed relative to two aspects: 1) the fact that many such points, while corresponding to distinct NN instantiations, nevertheless, all perform the same mapping from the NN’s input domain to its output range, and 2) the set of (distinct) mappings that can be performed by this set of NNs is typically just a subset of all possible mappings on the NN’s input domain to its output range (called the NN’s performance subset [17], [19]).

1) FWNN: The FWNN mentioned in Section I may be said to have learned a parameterized representation of quadratic mappings. It is no surprise to think of quadratic mappings in terms of a parameterization (cf. coefficients of a quadratic function), and the ranges of those coefficients define the family. A manifold for the quadratic mappings would comprise the collection of all such mappings, with coordinates provided by specific combinations of the coefficients. The parameterization developed in the FWNN of [35] was of a substantially different nature than the quadratic-equation coefficients, but the functionality discerned in the trained FWNN is amenable to being described via the neural manifold construct [38]. It was intriguing to notice that in the representation learned, activation levels of the recurrent connections operated like virtual parameters. They adjusted the bias levels of selected neural elements—thus, can be ascribed the role of manifold coordinates to provide the indexing mechanism for selecting the current mapping [38]. During operation, when a change in context occurred, these biases changed their value and held constant until the next change in context occurred.

These and related observations motivated a special construction to assist in “deciphering” the functionality of the trained FWNN of [35]: define an NN such that the weights and biases of the NN are continuous; hold the weights static and allow the biases to be set dynamically. Once the weights are fixed (static), each setting of the biases (dynamical) defines a unique NN, thus an indexing mechanism. This appears to be what the FWNN learned to do. The next challenge is to learn how the FWNN chose the “correct” NN from the manifold’s set; once understood, we may then employ such insights for the larger objectives associated with HLLA.

V. Context Discernment

The definitions of context discernment given in Section I are Agent centric, with specifics dependent on whether the Agent is performing EBSID and/or EBC. Current knowledge about plant, its environment, and the CF is required to perform context discernment and selection.

In the adaptive control methods called model-reference adaptive control (MRAC) and self-tuning control (STC) [3], [29], the MRAC involves an online assessment of a CF in terms of a reference model, and the STC involves an online identification to estimate the plant’s parameters. In both cases, the controller’s parameters are then adjusted based on the newly acquired measurements, i.e., they perform an act of context discernment: the STC in the plant portion of context space and the MRAC in the CF portion. Whereas the information gained by these two methods is used to incrementally adjust a controller’s design (by adjusting parameter values), the EB methods are to use this knowledge to quickly select a model from the respective controller or plant repository.

To achieve the desired efficiency attribute, the proposed process of selecting a controller (or model) from the experience repository must be tightly intertwined with the process of context discernment, and what is particularly critical is that the representation schemas for the two processes must be tightly related and potentially identical. The aforementioned definition for neural manifolds refers generically to a specific type of NN but may be refined to reflect the use to which the NNs are put, i.e., if the NNs represent controller models, assign the name policy (controller) manifold, or if the NNs represent plant models, assign the name plant manifold. So far, an underlying assumption has been made that both manifolds are constructed
An intuitive example of efficiency, nearness, and mappings would be a store that rents movie DVDs and has the DVDs shelved alphabetically by movie title; if the customer knows the title, there is a straightforward (for a human) search strategy for finding the desired DVD. If the store has the DVDs shelved according to content type, a customer with content type in mind but who does not know a title is able to browse the various nearby DVDs and make a selection based on information provided on the DVD covers. The notion of nearness for the first case relates to the spelling of the movie title, and for the second case, to the content of the movie. Which is more efficient depends on the customer’s knowledge and needs. A store that has all its DVDs shelved alphabetically could provide various cross-reference listings, e.g., by genre, artist, studio, decade, etc. Each such list is recognized here as a mapping from one form of representation to another.

Since the coordinate space for a neural manifold is defined via the NN weights, it follows that the weights provide indexing for both the plant and policy manifolds. So far so good. However, how does one go about crafting a mapping between, for example, the coordinate space for the plant to that of the policy manifold? Such a mapping will be required for the Agent to select a policy based on the information about the plant. More generally, how do we craft a good mapping from the full context space to the coordinate system of the policy manifold or of the plant manifold?

The task of answering these questions is assigned to the HLLA introduced earlier—i.e., the answers are to be learned.

The mapping from context space to the policy manifold may, in general, be many-to-one, i.e., changes in plant dynamics or the environment do not necessarily imply a needed change in the control policy. The ultimate efficiency may be for the two spaces being so coordinated that the mapping turns out being one-to-one.

An important aspect of crafting the mapping between the context space and the involved manifold relates to the efficiency issue. For example, when the EBC discerns a change in the context space that requires a different controller, the EBC’s selection task is to pick the “best” controller from the repository (the policy manifold’s set). While it is conceivable that such a selection can be made in one step, it may be that the EBC will select some intermediate policy, let it operate for one or more steps, gather observations that provide sufficient new information, and then make the next selection. It may take a sequence of selections before arriving at the optimal policy for the given point in context space. The HLLA’s task is thus even more complex: it is to develop an EB-Algorithm (for EBC) in such a way that the “trajectory” in policy coordinate space is optimal—where this optimality is assessed via a CF. Given an observation that calls for an experience existing in the repository and given an EB-Algorithm with the capability to generate optimal trajectories in the repository’s coordinate space, a great stride is thereby made in solving the efficiency problem in the face of a huge repository. An equivalent argument based on these ideas was made in [40] for the longstanding problem in AI known as the frame problem.

VI. TRAINING BY HLLA TO DEVELOP EB-ALGORITHM FOR SYSTEM IDENTIFICATION

The list of tasks generated so far for HLLA is recapitulated in Table II.

The major purpose of the HLLA is to train the EB-Algorithm such that the attributes listed in the second and third items in Table II are manifested. This begs the question of what be directly impacted by the trajectory taken in policy coordinate space, and this aspect will, no doubt, be affected by the mapping adopted.

Also related to the efficiency issue is that if the reasonable assumption is made for a given application that the portion of context space to be experienced by the Agent is bounded, as more and more experience is accumulated (resulting in the manifold’s set becoming more and more populated), the likelihood increases that a new observation is associated with a previous experience.

Given an observation that calls for an experience existing in the repository and given an EB-Algorithm with the capability to generate optimal trajectories in the repository’s coordinate space, a great stride is thereby made in solving the efficiency problem in the face of a huge repository. An equivalent argument based on these ideas was made in [40] for the long-standing problem in AI known as the frame problem.
training methodology to use. No general answer is offered here, but the author’s approach has been to employ the DHP AC method to solve this higher level task, and note here that in the AC methods, the sole vehicle for providing information to the method about the objectives of the application is the AC’s CF. Furthermore, any application that can be formulated to have components that can be conceptualized as filling the roles of “controller” and “plant” (where “controller” and “plant” need not be literal) is amenable to the application of the AC method.

The key to applying the AC method to achieve the aforementioned tasks is to craft an appropriate CF—this becomes the main role to be filled by the human designer in setting up the HLLA to accomplish the task of training the EB-Algorithm. The main advice to such a designer is to pay close attention to the requirements embedded in Table II and, in addition, to some of the issues to be addressed in Section VII.

The following three examples exemplify selected portions of the HLLA notions, relative to the SID genre of tasks.

1) **EB SID Example 1: Quadratic Mapping**: Consider a family of quadratic mappings, and apply an input to a “black box” whose input–output transfer is determined by a specific one of these mappings. Assume that, occasionally, the mapping instantiated inside the black box changes to another member of the family. To apply the HLLA ideas, populate a neural manifold’s set with NNs, each of which performs a single one of these mappings, and the NN weights serve as the coordinate space. Call this particular neural manifold the model manifold.

The HLLA is to develop an EB-Algorithm for the EBSID box in Fig. 3(a). This EB-Algorithm is to observe a sequence of data emanating from the black box and select the NN from the model manifold that corresponds to the mapping inside the black box that is generating the data (each candidate NN receives the same input stream as does the black box). An appropriate CF is specified to assess the quality of fit of the data emanating from the candidate NN; assume that the CF is to minimize the squared error between the data observed from the black box and the data generated by the candidate NN (note that this is not the “higher level” CF).

From the SID perspective, the context variables are as follows: 1) parameters of the quadratic mapping and 2) parameters adopted for the aforementioned CF (equivalent to the A and C boxes in Fig. 1). The point in the environment subspace is assumed fixed for this example. In this way, with a specific quadratic mapping instantiated (generates the observed data) and with a stipulated CF, a specific point in the context space is defined. The EBSID’s task is to select the NN in the model manifold’s set that corresponds to this point in the context space using the given CF as the evaluator.

It is HLLA’s job to design the underlying EB-Algorithm for the EBSID to accomplish the aforementioned task in some optimal way (the “higher level” CF is defined for this optimality). In the present case, a DHP AC approach was employed. Once the EBSID is designed (via training), it is then employed to accomplish the selection task (for SID).

To set up the training process, first adopt the perspective of a different Agent, one whose task is to train the EBSID (thus invoking a two-level hierarchy). In turn, define a new context space for this perspective. The two key subspaces of this new context space are as follows: 1) the family of quadratic mappings (the role of plant here) and 2) the new higher level training CF. This CF’s definition is to capture the requirement for the resulting EB-Algorithm to have the ability to sequentially specify an optimal trajectory through the neural manifold’s coordinate space for selecting the “correct” NN.

During training, if an incorrect NN is selected, a set of observations/measurements dictated by the training CF is taken and then evaluated. This value provides information for where to go next in the model manifold’s coordinate space—i.e., which NN to instantiate next. Note that each observation provides only partial information; therefore, it is unlikely that one step will suffice. After each NN instantiation, new data are collected via observation, and the aspiration is for the EBSID to attain the capability to generate a sequence of NN choices that leads to the “correct” solution (in this example, the NN that corresponds to the instantiated quadratic mapping) and to do so in a way that is, in some sense, optimal (as defined by the training CF). Examples for criteria include the following: 1) Minimize the accumulated squared error during the trajectory, or 2) minimize the number of steps in the trajectory. For controller selection, stability requirements could be embedded in the CF.

Once the design of the EB-Algorithm is completed, when the EBSID discerns that a new quadratic mapping has been instantiated in the black box (a change in context occurred), the EB-Algorithm commences to guide a new trajectory in the manifold’s coordinate space to select the NN corresponding to the new quadratic mapping.

Successful early experiments of the aforementioned design have been carried out (see [38]). Many improvements, embellishments, and extensions remain to be made.

2) **EB SID Example 2: Pole-Cart Plant**: As another partial example of this paper’s notions, consider the often-used benchmark pole-cart plant. This comprises a wheeled cart on a straight track that may be pushed back and forth via a controlled force that is parallel to the track; the cart has a hinge that holds an inverted pendulum of a specified length and mass. The objective is to apply a sequence of control actions (discrete time is assumed here) to make the pendulum stand vertically following a disturbance in the pendulum’s angle relative to the vertical (maximum angles are $+90^\circ$ from vertical). This type of plant is useful due to its conceptual simplicity and, at the same time, characterized by nontrivial nonlinearities. The plant is characterized via an analytic model whose form is such that the length and mass of the pendulum are among the model’s parameters. This makes the construction of a plant manifold easier, implicitly populating its set with various instantiations of the analytic model and defining the coordinate space via the model’s parameters. The task is to discern changes in context when they occur; the context variables of interest are the pole length and mass. The result of this SID process could be used, for example, as the prerequisite step to controller selection (cf. Fig. 4).

Again, assign to the HLLA the task of generating an EB-Algorithm for the EBSID box in Fig. 3(a). This EB-Algorithm is to observe a sequence of data emanating from the system and select the model in plant manifold that corresponds to the

Authorized licensed use limited to: University of Illinois. Downloaded on December 21, 2008 at 20:14 from IEEE Xplore. Restrictions apply.
pole-cart system generating the data (each candidate model receives the same input stream as does the pole-cart system). An appropriate CF is specified to assess the fit quality of the data emanating from the candidate models (e.g., to minimize the squared error between the data observed from the pole-cart and data generated by the candidate model). Due to space limitations here, refer to [13] for details of this example.

3) EB-SID Example 3: NN as Plant: Consider an NN of the type described at the end of Section IV. Generate a set of such NNs by varying the bias values; let the biases be the coordinates, thereby constructing a neural manifold. An equivalent procedure may be used here as for the previous example. For this construction, the task of the context discerner is to determine which of the NNs in the repository corresponds to the one currently instantiated. The procedure gave excellent results (not yet published).

An observation to be made here relates to a comment in Section IV that many points in the set, while corresponding to distinct NN instantiations, nevertheless, all perform the same mapping from the NN’s input domain to its output range. In the present example, we “looked under the hood” after training and noted that, while the error was indeed small, the weight combinations for the selected model were not always identical to the NN instantiated in the plant box. It appears that the EB-Algorithm learned to select candidate models that were functionally equivalent to the plant rather than select only the one that had the identical configuration of weights. This result is entirely consistent with the fact that the CF only considered the outputs of the plant and candidate models. This brings to the fore another earlier comment that the CF is the sole vehicle for providing information to the method about the objectives of the application.

VII. DISCUSSION

A. Experience Repository

The experience repository, which is formalized as the set portion of a manifold, is the collection of controllers available to the EBC for a given control scenario or the collection of plant models for the EB-SID to perform a SID task. So far in this paper, the issue of how the experience repository gets populated in the first place has not been addressed. In principle, when the repository is empty, “experience” does not yet exist. In practice, the repository will likely be built up piece by piece, employing whatever tools are available to develop the respective components. For example, in a control setting, one could employ any of the methods of Section II to generate a set of controllers for a given engineering task and for selected points in context space, which itself has to be characterized/parameterized, and collect them into a repository. In addition, one would then be obliged to craft a list of controller attributes (parameters) that can serve as coordinates. A control manifold could then be defined. The big hiccup here is the significant technical difficulty related to developing the “list of controller attributes” just mentioned that can serve as coordinates for the experience repository to yield a useful manifold and, in addition, coming up with an appropriate parameterization of the context appropriate to the task being addressed.

Thus, the big question that emerges is how to characterize the collection of components in the plant model repository for the SID task and the collection of components in the controller repository for the control selection task and, at the same time, to characterize the context space in a way that useful mappings between it and the two repositories can be crafted. The success of the entire EB enterprise suggested in this paper will rest with success in these endeavors, i.e., in crafting coordinate spaces of the repositories, crafting the representation(s) of context, and crafting the mappings between the context space and the coordinate space of the manifold appropriate to the Agent’s task.

While it has been stipulated that one of the tasks of the HLLA is to, in fact, develop such representations and mappings and, indeed, to do so in a tightly coordinated fashion, in general, this is not likely to come easily. It may be that in the sequence of developments in this new phase of the control field, continued assistance will be required of the human designer for inventing the schemata employed for indexing the controllers or plants residing in their respective libraries/repositories. As a minimum, the human designer can contribute significantly to the HLLA’s potential success by being very mindful of the aforementioned issues when initializing the repositories the HLLA is to work with. As the field matures, the HLLA is to take over more and more of this task. Because of the tight coordination requirement mentioned above, it may be that the HLLA (or one at a yet higher level) may have to guide the creation process of the controller designs that are to populate the repository for specified portions of context space so they can be crafted in a way that permits the mentioned coordination. In principle, this process could use any of the Phase-2 and/or Phase-3 methods to create these designs, but a metalevel guiding principle will have to be developed and employed.

For the examples described in Section VI, we (the human designers) filled the so-called meta-metalevel role in crafting the manifolds. For research purposes, a “synthetic” method was employed: A model, analytic equation, or NN was used, and their respective parameters served as built-in indexing mechanisms. If the structure of the repository models is a direct analog of the system in the Agent’s focus, the mapping aspect could be made simpler or even eliminated. To obtain a close identification fit for a SID task, the model structure has to at least subsume the plant structure being identified. Efficiency is enhanced by having the structure closely match the plant structure; this also requires the most a priori information.

For another aspect of mappings, consider performing SID on a linear plant, and assume that the transfer functions in the plant manifold are factored polynomials, but the CF requirements are given in terms of an expanded polynomial representation. While the two representations are equivalent, the Agent would need a mapping between the two to accomplish the controller selection (e.g., via factoring the polynomial). In a second-order case, the notion of nearness in the CF subspace might be in terms of the damping coefficient, whereas in the corresponding plant-manifold coordinate space, nearness would be in terms of $S$-plane pole locations.

For the quadratic equation example in Section VI, a metalevel role was invoked by knowing (at another level up) the
principle we wanted to demonstrate, and that role crafted the situation accordingly. Similarly, for the pole-cart SID example, the model repository for the pole-cart system was defined via an analytic equation, the set portion of the manifold was generated via the analytic equation, and again, the coordinate space was constructed via the equation’s coefficients, with a prior stipulation specifying which of the context variables (pole length and mass) were to be focused on for the context discernment task. In the third example, the plant was defined to be an NN of a specified structure and element type, and the repository comprised various instantiations generated via the weights taking on different values. Thus, in all three examples, the repository comprised models of exactly the “correct” form, and the task was to select the one with the correct parameter values. This was a useful starting point to demonstrate that given such manifolds and corresponding mappings, an RL process could be configured to implement an HLLA to develop a context discerner with effective and efficient selection capability, i.e., demonstrated that an RL process could be employed to train an Agent to learn Component B of Definition 7 in Section I, the particular HLLA employed called CRL [13], [38], [39]. Demonstrations provided earlier are for the SID part of the task, but early work to develop demonstrations of the controller selection task has been sufficiently promising for the author to encourage others in the field to take up the mantle of such developments.

B. Model Refinement/Generalization

The phrase “with possible refinement” was invoked earlier when describing the controller or plant model selection process. Behind this lies the notion used in NNS called generalization: Provide an output for an input not seen during training and, in particular, “good” outputs, with the obvious connotation. In the aforementioned examples, the manifolds were crafted; therefore, the interpolation/generalization/refinement came rather easily. For example, by virtue of defining the set elements via an analytical equation, there is an automatic mechanism in place for the “interpolation” process that takes place in the corresponding coordinate space. If the “refinement” cannot be successfully accomplished via interpolation, then perhaps the HLLA would need to invoke an online redesign method, such as one of the learning control methods mentioned in the Phase-3 discussion.

Various aspects of the repository impact the quality of generalization and the efficiency/effectiveness attributes and are useful to consider during construction. For example, larger population count in the repository impacts both generalization and selection response time: generalization is improved via more and more constraint (of the good kind) being present. For NNs, this added constraint is known to increase chances of better generalization. Also, an increased “richness” of the available solutions in the repository will likely result in fewer steps during selection. Any a priori knowledge available that is relative to the application tasks may be employed to build in such constraints when parameterizing the context and controllers and while populating the repository.

For a neural manifold as defined earlier, one way that the HLLA itself could embed more and more constraint as learning progresses is as follows. Start out with all the weights of the NNs being dynamic, and as more and more information about the application domain is learned, start changing the status of the selected weights to static. In essence, this is what we inferred the FWNNs described in Sections I and IV have accomplished.

C. Prespecification

A comment was made in Section I related to limitations implicit in the prespecifications required in the adaptive and learning methods. That comment is now revised as follows. The specifications to be provided by the human designer are to be at a “higher level” than those entailed in the adaptive and learning control methods. This higher level prespecification is to be employed to tailor the definition of the coordinate spaces and the various parameterizations to predispose successful context discernment and controller selection/refinement. Progress in the field will depend on the success in this aspect.

VIII. CONCLUSION

There remain many issues to confront as the idealizations in the notions advanced in this paper are relaxed little by little. In the examples provided in Section VI, the repositories were populated with models of the precise form of the plant being identified. The same is true for the preliminary work related to controller selection. What is to happen as the plant and controller models are close approximations but not exact? What is to happen when noise is allowed into the process? The notion of “levels” is invoked in the EB notions. While these concepts are more typically used in the systems literature (cf. [16]), it will be useful to more deeply explore their application to the EB control notions presented here.

The theory development and the experiments carried out so far are but a starting point along an anticipated long development path wherein HLLAs are crafted and instantiated to attain both EB controller and EB SID capability for increasingly general application settings. One need only look at the biological exemplars to envision the possible applications as the underlying theory reaches maturity: indeed, a vision for the control field.

ACKNOWLEDGMENT

The author would like to thank the creative role filled by the various graduate students in his Laboratory over the years. These include, but not limited to, the following: M. Carroll, L. Holmstrom, S. Hutse, B. Johnson, G. Lankena, J. Lotz, K. Mathia, S. Matzner, J. McCarthy, C. Paintz, A. Rogers, A. Rustan, L. Schultz, S. Shervais, A. Toland, and S. Vasupongayya. The author would also like to thank, in particular, the Lab Managers: T. Shannon and R. Santiago. The pole-cart experiments in Section VI were performed by L. Holmstrom, and the quadratic mapping and NN-as-plant experiments were performed by R. Santiago.
REFERENCES


George G. Lendaris (S’55–M’58–SM’74–F’83–LF’97) received the B.S., M.S., and Ph.D. degrees from the University of California, Berkeley, in 1957, 1958, and 1961, respectively.

He is currently the Director of the NW Computational Intelligence Laboratory and the Director of the Systems Science Graduate Program, Portland State University (PSU), Portland, OR.