Experience-Based Control and Context Discernment: 
A Next Phase for the Controls Field?

George G. Lendaris
Systems Science Ph.D. Program and NW Computational Intelligence Laboratory (NWCIL) 
Portland State University, Portland, OR

e-mail: lendaris@sysc.pdx.edu

Key Words: Experience, Context, Context Discernment, System Identification, Neural Networks, Reinforcement Learning, Approximate Dynamic Programming, Optimal Control.

ABSTRACT

Two distinguishing features of human-like control vis-à-vis current technological control is 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. The notions of context and context discernment are important to understanding this ability. The approaches embedded in the methods known as Adaptive Control and Learning Control focus on modifying the design of a controller in use as changes in context occur. The key to attainment of experience-based control is posited to entail a shift of the technologist’s focus “up a level”, away from individual controllers to that of algorithms that efficiently and effectively, on line, select designs from a repository of existing controller solutions for selected regions of context space. The mathematical construct of manifolds of geometric topology is a useful formalism for such inquiry. A key component of the notions presented here is that of Higher Level Learning Algorithm. This is based on Reinforcement Learning, with its focus shifted to the posited higher level, and is employed, with very promising results. General ideas and definitions are given, along with examples and considerations for further developments.

I. INTRODUCTION

The quest of many researchers in our field is to achieve more human-like capability for identification and control with our technology. The ideas and work reported in this paper rest on the belief that effective and efficient employment of knowledge that is attained via experience is a major factor underlying human-like performance levels for such tasks. While the control systems field has indeed accumulated remarkable achievements, and for some applications exceeding human control capabilities, nevertheless, substantial additional progress is needed toward building into machines the ability to employ experiential knowledge (hereafter called experience) when performing system identification and when coming up with a good controller for a given situation, and importantly, to do so effectively and efficiently.

This paper puts forth the notion here called Experience-Based Identification and Control (EBIC); provides a general definition and some requirements for fulfilling this notion; and gives an overview description of a novel concept for applying Reinforcement Learning at a “higher level” to accomplish the Experience-Based (EB) ideas.

1 This work was partially supported by NSF Grant ECS-0301022
The Experience-Based 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 is able to quickly generate reasonably optimal performance on the new task, based on the previously learned skills. Examples include athletic performance under varying conditions, grasping and moving objects of similar size/shape, solving a class of mazes, etc.

2. The more knowledge a human attains, the speed and efficiency of performing tasks are improved (in a relevant environment). On the other hand, in Artificial Intelligence systems thus far developed, the more knowledge acquired (typically stored as “rules”) the slower the decision/action processing.

We posit that implementing the equivalent of experience will be the key to achievement by computational Agents the effectiveness and efficiency noted above. The Experience-Based (EB) approach presented here entails:

1) a collection of models (of plants or controllers, depending on whether doing system identification or control) appropriate to a given engineering application,

2) a characterization of this set of models in a form that facilitates accessing the models, and

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.

We take these components as being fundamental to what is meant when humans are said to have attained experience related to a class of identification/control tasks. We note that implicit in these requirements is a memory property for the Agent.

In terms of a control engineering setting, consider the following: a plant, environment, and control objective are provided to a designer, who is to design and implement a controller. If the designer is experienced and has “seen” the situation before, after obtaining context data (defined in the next Section), he/she pulls the appropriate design out of the archives and applies it to the current situation – perhaps with a little tailoring. The more experienced the control engineer is, the process goes faster and with better results.

To aid us in the task of crafting a definition of experience for the present purposes, we again appeal to human performance and take note that another notion is fundamental here as well – that of context. Humans intuitively understand that as context changes, so do the decision rules and or control policies we use to function within the given context.

We formulate context as comprising three components: 1) plant, 2) environment, and 3) objectives plus associated performance criteria (labeled CF). See Fig. 1. Specification of all three yields a specific context; a change in any of the components results in a different context. In this formulation, to each specific context there corresponds a particular control law.

For an intuitive entrée to this use of the term context, consider driving your car down a nice country road on a clear afternoon on 1) dry pavement, or, 2) icy pavement. You use the same set of driving skills in both cases; however, depending on which of the two situations obtains, you find it necessary to make adjustments to your “control law” and/or “decision logic.” If instead of changes in the environment (road conditions) there is a change in your car’s attributes (e.g., a tire becomes slightly flat), adjustments are again needed for the car to perform your desired
maneuvers; driving your friend’s car that day rather than your own would be another example. A third type of consideration for selecting your control law involves performance criteria; for example, if you are in a road race a criterion might be to minimize time, but if your are taking an elderly relative on an excursion, the criterion might be to maximize comfort. Each of the conditions described in this example 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 (tasks of the same genre), and also, entails a memory about the collection. The collection is here called an experience repository for that domain.

A tight coupling will be required between the representation crafted for context, the discernment to be accomplished, and the decision/controller selection process, particularly for the issue of efficiency; the coupling entails distinct indexing schemas for the various sets, designed to facilitate efficient transition to a new design as needed. Our approach uses the formalism of manifolds from geometric topology, where such manifolds comprise 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 are provided in Section IV). This formulation provided the framework in which we developed a novel concept for applying Reinforcement Learning (called HLLA – Higher Level Learning Algorithm) for evolving the experience-based ideas presented herein. An instantiation of the HLLA to system identification, called Contextual Reinforcement Learning (CRL), is being applied with success in important early steps of the larger objective [13][38]. The key idea for HLLA is to re-purpose the Reinforcement Learning (RL) method (at a “higher level”) so 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 an experience repository), 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 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 changed – a process here called context discernment; the latter process typically entails a form of system identification (SID), especially for the ‘plant’ portion of Context. The SID process is also enhanced when performed using experience. Examples of successful application of the HLLA approach to the SID portion of the process are included in Section VI.

We note that various existing methods of the controls field may be employed to generate components for the repository; those methods are here assumed available within the experience-based process as a means for growth of the repository.

In summary, it is posited that the following four aspects are fundamental to the Experience-Based (EB) notion: 1) context, 2) discerning current context, 3) selecting appropriate solution for the discerned context from an experience repository, and 4) doing the latter two in an effective and timely manner. We further posit that context discernment is fundamental not only for the selecting aspect, but also 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? Notions of hierarchy and optimization are fundamental to such considerations; a Context Space Hierarchy is currently being conceptualized to assist in this endeavor.
While some of the existing Adaptive and Learning Control approaches may be said to incorporate the idea of context, we comment that these methods in general depend on an external designer to predefine the associated contexts and how to recognize them. Biological intelligent systems, on the other hand, do not require that context be predefined; instead, they learn how to discern context on their own. It is easy to infer that the latter capability entails an appropriate representation of context. It follows that an important attribute of an Agent that is being trained to have the EB capability will be for it 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; the latter is in contrast to existing methods, whose Agent may be described as (only) context dependent. One of the distinctions here is that a context discerning Agent is to require little or no predefinition of context, or how to detect it. We re-emphasize that, even conceptually, there is a tight coupling between the representation crafted for context, the discernment that is to be accomplished, and the controller selection process. This coupling becomes particularly relevant for the aforementioned issue of achieving improved efficiency, especially as the experience repository grows larger and larger.

As an example, consider a careful crafting of the context representation such that it could be employed directly as a kind of “indexing” mechanism; this would allow the following to occur: recognize the various contexts when encountered again (via the discernment process), and once recognized, directly select (via the “index”) and redeploy the corresponding previously developed solution(s) – with possible refinement(s) apropos the current discerned context.

There is an approach in the literature that may indeed be said to perform context discernment; this approach applies recurrent neural networks to system identification and control, primarily under the rubric Fixed Weight Neural Networks (FWNNs) [8][10][11][35]. However, while the operation of these networks is rather remarkable, no principled explanation has existed for what is here called their context discerning capabilities. Also significant, these networks require a large amount of computation resources for even small problems. Analysis of this early work at the author’s Lab yielded a principled explanation of the FWNN’s context discerning capabilities and, further, yielded an approach that reproduces many capabilities of FWNNs with reduced computational cost [38].

As a precursor to further describing the Experience-Based notions of this paper, we comment that while the existing methods of Adaptive Control and Learning Control perform (a modicum of) controller redesign during operation to accommodate certain types of changes in plant and/or environment, much of the process is prespecified by the controls engineer (more about these methods in Section II). The EB Control approach suggested here seeks to go beyond the limitations implicit in the need for such prespecifications, or at least to reduce their number.

This paper has three main components: 1) Define the notion of Experience-Based Identification and Control (EBIC), including a component notion called Higher Level Learning Algorithm (HLLA), and some requirements for their realization, suggesting EBIC to be a significantly new development phase, indeed a vision, for the field of controls; 2) As a basis for the latter assertion, provide a historical overview of the field of controls from the perspective of how the notion of context has (or has not) been explicitly dealt with to date; and 3) Provide an overview description of the method we call Contextual Reinforcement Learning [38] as a candidate Higher Level Learning Algorithm (HLLA) that has promise for fulfilling the new vision. Our starting point is to be a historical overview, but first, an interlude.
IA. DEFINITIONS

Before proceeding, we ask the reader’s indulgence, and present a list of definitions to facilitate
the remainder of the presentation.

1) **Agent**: computational intelligence device (that, in this paper, is to perform the acts of context discern-
ment and selection, along with possible design refinement).

2) **Context Variables** (Agent centric): those attributes of i) the environment and ii) the plant/process
whose variations *could engender changes* to the decision rule / control policy *employed by the Agent*
while accomplishing the Agent’s current objective or goal; and in addition, iii) the criteria (representing
the objective or goal) to be used for designing and subsequent selection of the decision rule or control
law. [We use the term **Criterion Function** (CF) to represent these criteria.]

3) **Context Space** (Agent centric): a vector space in which each context variable associates to a dimen-
sion. The Context Space concept comprises three sub-spaces; one each associated with the i) Plant, ii)
Environment, and iii) Criterion Function.

4) **Context** (Agent centric): a point in Context Space; the *set of values* taken on by the context variables in
a given situation.

5) **Context Awareness**: the act of monitoring the application to take notice (become aware) that a change
may be occurring in the Context.

6) **Context Discernment**: the act or process of determining the current values of the context variables
(current point in Context Space) appropriate to the task being performed. [Webster on-line for ‘dis-
cern’: to recognize or identify as separate and distinct.]

7) **Experience-Based approach**: A two-component concept:
   Component A: **Repository** of previously developed context-specific models (controller or plant mod-
els), and
   Component B: **Algorithms** used by the Agent to effectively and efficiently *select* a model from the
repository as changes in context occur. [Note: A key task of the Higher Level Learning Algorithm (de-
 fined below) is to train the Agent to *learn* Component B.]

8) **Selection**: the act of choosing/retrieving an appropriate element of the repository corresponding to the
discerned context.

9) **Higher-Level Learning Algorithm (HLLA)**: The reference level for the term ‘higher’ is the case
where the learning algorithms are applied directly to the design of optimal controllers (as in Learning
Control), ones that would be accumulated in the repository. ‘Higher-Level’ here means applying the
learning method to create a strategy for *selecting* a good controller from the repository, where the *process of selection* is optimized; thus, the ‘focus’ of the learning process is at the “next level up”. Definition of the Utility function (CF) in an appropriate way for this new design process is important. Note: When the Contextual Hierarchy ideas mentioned in Section I are developed, more levels will be in-
volved.

10) **World Space (Agent Centric)**: A vector space whose dimensions are associated to designated attrib-
utes of the Agent’s relevant environment, its physical body, and the external CF.
   [Note: This definition is included for completeness. It is not explicitly used in this paper, but is
   used in related publications in terms of mappings from World Space to Context Space, e.g. [39].]

**Construction Guidelines:**

Parametric models/equations are used to represent the Plant, Criterion Function (CF), and Environment
(for the latter, measurements may serve as *parameters* w/o an explicit model). We construct a **Parameter
Space** that comprises three sub-spaces: (Plant, Environment, CF). These *parameters* serve as Context
*variables* for the discernment activity; not all may be pertinent to the Agent’s task, so Agent’s Context
Space may be a sub-space of Parameter Space. Controllers are also represented via parametric models.
II. HISTORICAL OVERVIEW of CONTROL FIELD vis-à-vis EXPLICIT ROLE of CONTEXT

Since the notion of context is fundamental to the Experience-Based (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 controls 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 experience-based processes, including autonomous context discernment and model selection.

These four phases in the evolution of the controls field perceived from the perspective of how the notion of context has (or has not) been explicitly dealt with to date are as follows:

**Phase 1: DESIGN BASED on INTUITION and INVENTION**

Various histories of controls (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 design 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.

**Phase 2: DESIGN BASED on MATHEMATICAL TOOLS**

Mathematics has played a fundamental role in the development of the controls field as it is understood today. This development path began with Maxwell’s use of differential equations to analyze the flyball governor’s dynamics (e.g., Ch 1 in [25]), 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 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 off-line, and de facto, each controller design is based on a single point in the Context Space, or at most, a small neighborhood of points.

This phase includes at least the following well known design methods: Classical Control, Modern Control, Optimal Control, Stochastic Control, and Robust Control (e.g., see [7][9][12][25][26][29][33][34][48]). We note that even though the progression of these methods has employed ever more sophisticated mathematical tools and insights, and yielded substantially enhanced controllers (with respect to the criteria defined for the objectives of each approach), nevertheless, in the end, after the controller is designed, it is placed in service with no associated mechanism to modify its design in response to changes in context, and hence the methods meet the criterion for this phase. We note, however, that in practice the designs in this category are often crafted to have “low sensitivity” (later called ‘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 on-line changes in the design itself.
Phase 2 is here characterized as a design approach where the design is done off-line, and de facto, each (fixed) controller design is based on a single point in the Context Space, or at most, based on a small neighborhood of points.

**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 on-line instantiation of different controller designs based on the observed variations. We note, however, that there are important distinctions between how knowledge of a changed context is attained, and how the different controller designs and/or instantiations occur in these methods vs. those to be described for Phase 4.

**A. Partitioning Methods.** A large segment of the methods in Phase 3 might be labeled Partitioning Methods, as these methods partition a nonlinear operating region into approximately linear regions, and develop a linear controller appropriate to each region. These methods may be said to focus on the Environment subspace of the Context Space (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, neural networks, Fuzzy logic, statistics, etc. The associated methods have, understandably, appeared under a variety of labels – e.g., multiple models (e.g., [49][50][51]), piecewise models, mixture of experts, Fuzzy models, local regression, and various others, as mentioned earlier.

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

**B. Adaptive Control.** Adaptive Control (e.g., [3][15][27][37][43]) distinguishes itself from Phase 2 type control primarily by allowing multiple policies to be available for performing the desired control. Gain Scheduling (e.g., see [28]) is perhaps the oldest and simplest form of such control. The (newer) Adaptive Control methods are traditionally grouped into two separate classes: Model-Reference Adaptive Controllers (MRACs), and Self-Tuning Controllers (STCs) [3][29]. In both classes, there is the equivalent of an “outer loop” that performs adjustments to controller parameters (sometimes called tuning).

In the MRAC case, specifications are given in terms of a reference model that indicates how the plant output should respond ideally to the command signal. The controller’s parameters are adjusted to make the plant output close to the model outputs.

In the STC case, specifications are given in terms of a performance index. There is an on-line identifier that estimates the current values of the plant parameters, and these estimates are used by an online “design block” that designs a controller to meet the performance index (cf. Ch 9 in [29]).

Generally, the Adaptive Control methods may be thought of in terms of parameterized controller models, and the adaptation refers to changes made in parameter values (of a pre-designed controller structure) to accommodate for performance errors that occur due to inaccurate plant models, and/or changes in the plant that is being modeled (i.e., changes in context). The controller structure is pre-designed (off-line) for the engineering task at hand, and the engineer’s job is to put together a principled method to be implemented for converging on the appropriate control policy (via controller parameter adjustments) from the set so defined, based on observations.
One feature that distinguishes the Adaptive Control methods from those to appear in Phase 4 is that on-line controller design occurs in Adaptive Control, in contrast to on-line selection of previously designed controllers (with possible refinement) to be described for Phase 4.

C. Learning Control. Learning Control is similar to the above, except that less is assumed known about the plant, and the controller model that is being adjusted is of a more general form than that used in Adaptive Control. The engineer’s job for Learning Control is to specify a parameterized controller model, with less a priori information embedded in it than is typical for the Adaptive Control case, and provide the learning algorithm used to iteratively arrive at an optimal policy, the latter process being based on a sequence of state observations and performance evaluations. The policy resulting from such Learning Control methods is typically capable of covering a larger portion of state space than is possible via previous methods. While other varieties exist (cf. [4][43]), the learning method focused on in the author’s work is Reinforcement Learning (RL), and in particular, Approximate Dynamic Programming (ADP) via Adaptive Critics [5][6][24][36][41][42][45][46][47]. In the general case of RL, the controller model is initialized based on little or no a priori knowledge related to parameter value selection, and during the learning process, the sequence of controllers that become instantiated may include designs that are not stabilizing controllers. This latter fact led to the practice of using the RL method off-line to come up with an (approximately) optimal controller, which could then be implemented on-line; however, various modifications have been developed in recent years to allow on-line employment of RL (e.g. [2][14][23][30]).

A feature that distinguishes Learning Control methods from those to appear in Phase 4 is signaled by the label ‘learning’, as the key interest in Phase 4 is to be experience. The notion of experience assumes that a learning phase has already occurred, and indeed is required for acquiring the experience (wherein the results of learning are remembered in an accessible way). While the proposed Phase 4 does entail training (of neural networks, via a Higher Level Learning Algorithm), what is to be learned is NOT a controller design, per se, but rather, how to effectively and efficiently select a controller design from a repository of previously designed controllers.

Phase 3 is here characterized as a design approach that 1) yields controllers for which selected on-line adjustment of parameter values is allowed, and 2) accommodates a modicum of variations in context. The mechanism for performing the ‘accommodations’ is distinct from that to be defined for Phase 4.

D. Adaptive Control vs. Learning Control. Summarizing the above, Adaptive 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 (or environment) observations and/or performance evaluations. In the Adaptive Controls case, the engineer specifies a set of available controllers and a corresponding algorithm to select a member of this set, based on observations. In the Learning Controls case, the engineer specifies a parameterized controller structure, and a corresponding algorithm to adjust the parameters, incrementally, 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 Adaptive and Learning Control methods lies in the amount of a priori information that is embedded in their respective pool of controllers, and the aspects of the approaches that are done off-line vs. on-line. Apropos the latter aspect, the Adaptive Control methods normally provide a guarantee that switching among the policies in the set can be done safely, say related to stability, in an online manner [3][29][37]. Historically, such guarantees were not available with Reinforcement Learning methods in general (cf. several chap-
 ters in [41]), but recent extensions demonstrate selection methods that permit on-line operation as well (e.g., [2][14][22][31]). A useful characterization of Adaptive vs. Learning Control for our purposes here is summarized in Table I.

<table>
<thead>
<tr>
<th>TABLE I. Comparison of Adaptive Control and Learning Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Candidate Pool of Policies</strong></td>
</tr>
<tr>
<td><strong>Adaptive Control.</strong></td>
</tr>
<tr>
<td>Much a priori information about plant and environment is available and used.</td>
</tr>
<tr>
<td><strong>Learning Control</strong></td>
</tr>
<tr>
<td>(of ADP type).</td>
</tr>
<tr>
<td>Little or no a priori information about plant and/or environment is available or used.</td>
</tr>
</tbody>
</table>

**E. Others.** There are myriad other approaches that could be visited here, but of the methods known to the author, none have yet made the transition into accomplishing the combined acts of context discernment and associated experience-based controller selection in a manner that is the hallmark of human-like control. Some methods may have come close. One that is known to the author is called neural-adaptive control, and combines neural networks with (traditional) adaptive control [32]. The adaptive control loop performs as described earlier; however, in order to speed up its design process when a new context occurs, a neural network (after having observed the adaptation process in a large variety of previous situations) is employed to essentially tell the adaptive loop “this is what you ended up with the last time you saw those initial conditions, so use this as your starting point this time.” I.e., a memory component was added to the adaptive process. This approach was motivated by stability-based requirements, and was very successful. In principle, it comes close to fitting the requirements of Phase 4.

The author concedes that while not explicitly mentioned in the literature as such, at some level, the essence of the Experience-Based ideas of this paper are to some extent incorporated in the various approaches known as multiple models [49][50][51], piecewise models, mixture of experts, and others of this type. Ensemble methods may fall somewhere between these latter categories; they employ multiple models, but instead of selection as defined here, there is a voting or merging scheme. Also, certain Fuzzy control approaches are difficult to classify in the present scheme.

Nevertheless, there appear to be 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 vs. size of the model repository) that we will face when attempting to develop human-level performance in our Agents. It is also true
that in multiple-model methods, the models employed are typically linear (except the more recent neural network ones [49]); no such constraint is involved with the proposed EB method.

**Phase 4 (new): DESIGN for EXPERIENCE-BASED PROCESSES, Including AUTONOMOUS CONTEXT DISCERNMENT and MODEL SELECTION**

The following are stipulated as requirements for this phase: Agent has the ability a) to use experience for model selection (controller or plant, depending on Agent’s task); and b) to do so efficiently and effectively. We posit four aspects whose consideration are fundamental to achievement of these two requirements: 1) context, 2) discerning current context, 3) selecting appropriate model from experience repository for the discerned context, and 4) doing the latter two in an efficient and effective manner. Phase 4 is defined and developed based on these, and collateral, requirements.

By the standards of today’s technology, we stand in awe at the human skill demonstrated in the apparent effortless context discernment and seemingly simultaneous selection of appropriate actions performed on a routine basis. Equally impressive is the human central nervous system’s (CNS’s) adjustment of its information processing based on cues available in the environment, and the appearance there is little (if any) explicit direction needed for development of this capability. This suggests a path for achieving Experience-Based 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. Two examples of the latter are the context variables crafted for the car steering and flight controllers to be described in Section IIIb.

Another capability of the CNS 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 do it. The what/when tasks have typically been left for the engineer or computer scientist to solve explicitly (cf. the gain scheduling method – where the ‘schedule’ refers to an explicitly determined set of criteria for determining the sequence of contexts). An objective we put forth for the Higher Level Learning Algorithm (HLLA) is to address (in the future) the when as well as the how questions.

Phase 4 is here characterized via anthropomorphism. A design approach that develops an (autonomous) Agent whose internal operations fills the experienced designer role in the following: A plant, environment, and control objective are provided to a designer, who is to design/select a controller. If the designer is experienced, has “seen” the situation before, and remembers the solution, after gathering context data pulls a design out of the archives and applies it to the current situation – perhaps with a little tailoring. The more experience the designer has, the faster the process goes and the better the results.

**III. EXPERIENCE-BASED CONTROL PRELIMINARIES**

As a starting point for describing what we want to mean by Experience-Based Control, we refer back to Section II.D, and comment that the comparisons made there, tabulated in TABLE I, were set up to facilitate making the following statement about a next developmental step toward human like control: Come up with learning methods to 1) accumulate and/or select a set of “relevant” policies for an engineering task (as is done by the engineer for Adaptive Control), 2) create a representation schema that facilitates accessing the policies, and 3) create an algorithm that efficiently and effectively switches between these policies (adapts) during online operation of the system, in response to changes in its context. The reader will recognize these as the three components listed for EBC in the fifth paragraph of this paper. This process is to entail the
Higher Level Learning Algorithm (HLLA) mentioned previously, where the HLLA is to select an optimal controller from the repository in an optimal way, in contrast to adjusting the existing controller’s design. It is important to take note that the two uses of the word ‘optimal’ here refer to two distinctly different Criterion Functions: the first is related to the control task, and the second is related to the process of selecting the corresponding optimal controller.

The presentations in Fig. 2 and Fig. 5 in the sequel are intended to provide the reader with a birds-eye-view of the key new components entailed in the Experience-Based notions.

Fig. 2 represents a generic structure for the Phase 2 Adaptive and Learning methods. The algorithm part of the Adaptive Control and of the Learning Control methods takes current observations and employs them as a basis for making modifications to controller/policy parameter settings. As noted earlier, the Adaptive Control methods are allowed to make selected adjustments to controller parameters online, i.e., to perform movements through restricted ranges of controller parameter space. The Learning Control methods also perform movements through the controller parameter space, typically over a larger range than allowed for in the Adaptive Control case. As mentioned above, these were historically employed for off-line operation, but recent developments allow on-line performance as well.

![Figure 2. Generic structure for A: Adaptive Control or B: Learning Control.](image)

We provide a simple example for the Adaptive Control and Learning Control methods to complete this part of the exposition.

**ADAPTIVE CONTROL EXAMPLE:** For a basic example of Adaptive Control, we examine the stereotypical (linear) Model-Reference Adaptive Control system shown annotated in Fig. 3 (as noted in Section II, there are two general classes of Adaptive Control: a) Model-Reference Adaptive Control, and b) Self-Tuning Control [4][37]. The process being controlled in Fig. 3 is linear with transfer function \( kG(s) \), where \( G(s) \) is known and \( k \) is an unknown parameter. A feedforward controller is used to yield a system with the desired transfer function \( G_{des}(s) = k_0G(s) \), where \( k_0 \) is a given constant. The control structure is of the form \( u = \theta \circ \hat{u}_c \), where \( \hat{u}_c \) is the command signal, \( u \) is the control signal, and clearly, \( \theta \circ \) is the control parameter. The cost function is given as \( e = y - y_m \). The rule stipulated for adapting the parameter \( \theta \circ \) is to make changes proportional to the negative gradient (relative to that parameter) of the Cost Function. The usual math yields the adaptation law shown in the diagram.
The adaptive “algorithm” employed in this Adaptive Control example is confined to a very specific set of (linear) control policies, parameterized by the coefficient $\theta^c$, and the adaptation law takes the current observation of $e$ and from this, generates a new value of $\theta^c$.

**LEARNING CONTROL EXAMPLE:** For an example of Learning Control, we show a basic form of Reinforcement Learning / Approximate Dynamic Programming, namely, the Dual Heuristic Programming (DHP) method, shown in Fig. 4. It is common in the ADP type of Learning Control method that the controller is implemented by a Multi-Layer Perceptron (MLP), and as such, represents a looser specification of available policies than is done in the Adaptive Control method. In this scheme the weights and biases of the MLP form the space of parameters for the controller (or policy), $\Theta^C$. The initial weight values for the MLP represent a starting policy in $\Theta^C$, and each subsequent observation is the basis for making a change in this parameter using the ADP procedures (gradient descent, etc.)

![Figure 3. Example of a (linear, Model Reference) Adaptive Control system.](image)

![Figure 4. General layout of Adaptive Critic DHP structure.](image)
In Fig. 4, the dashed lines (including the little short ones) represent calculated values being fed into the respective boxes, and the heavier dash-dot lines indicate where the learning/updating processes occur. The controller is labeled Action; it receives the current state of the plant, and issues a control signal $u(t)$, whereupon the plant generates its next state. The two Critic boxes are copies of one another. In this method, both the Critic and the Action (controller) boxes are adjusted during the training process, and are typically implemented as neural networks (but other means are possible). The plant is typically nonlinear. A Utility function is defined to represent the control objectives, say, in terms of “costs”; for the discrete-time regulator case, the CF is defined as the sum of all future costs required to get from the current plant state to the desired end state (called cost-to-go). The rule for adjusting (or training) the action/controller weights is to make changes proportional to the negative gradient of the CF relative to those weights. For the present purposes, the key thing to note is the dash-dot line with the label $\theta^C$, indicating that the neural network weights are the parameters of interest here, and they are being adjusted via the Adaptive Critic method. See [24] for operational details.

We proceed now to describe some distinctions of the HLLA notion from the Adaptive and Learning Control methods. We focus on two main aspects of the HLLA process, namely, system identification (employed as a component of context discernment performed on the Plant subspace), and controller selection. We look first at Fig. 5a, and see that the box of Fig. 2 labeled “Adaptive Algorithm / Learning Algorithm” has been replaced with two boxes, one labeled EBSID and the other HLLA. Both of the boxes are significant. The HLLA (Higher Level Learning Algorithm) works with a set of plant models appropriate to the given system identification task and 1) creates a representation for the (sub-) set of these models relevant to the given engineering task, and 2) trains the algorithm embedded in the EBSID; we give this algorithm the generic name: Experience-Based Algorithm (EB-Algorithm). The EBSID employs the EB-Algorithm thus learned to effect the system identification. In addition to the box-replacement we noted between Fig. 2 and Fig. 5a, we also removed the connection to the controller box in Fig 1 for Fig. 5a. This reflects the view that the systems identification task does not normally feed into the controller, at least not directly.

Proceeding next to Fig. 5b, we note that the aforementioned box of Fig. 2 has again been replaced with two boxes, one labeled EBC and the other HLLA. This time, we include the (trained) EBSID 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) and 1) creates a representation for the (sub-) set of these policies relevant to the given engineering task, and 2) trains the algorithm embedded in the EBC; we again give this algorithm the generic name Experience-Based Algorithm (EB-Algorithm). The EBC employs both the EBSID sub-process (as part of the context discernment activity) and the EB-Algorithm thus learned to effect the Experience-Based 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 used to index the set of (relevant) controllers. In the next Section, we describe what is meant here by ’coordinate space’.

We comment that the HLLA process as described above is on the surface an “off-line” method that designs the EB-Algorithms that are used subsequently to perform EBC. More generally, HLLA could continue to perform on-line, and when doing so, employ a means for implementing the notion of “safe-fail” for the process (in distinction to “fail-safe”), which is often as-
associated with nature’s ecological processes (e.g., forests have built-in mechanisms for re-growth following severe forest fires), and clearly observed in animal systems.

**IIIa. PROPOSED EXPERIENCE-BASED CONTROL CONFIGURATION**

Fig. 6 below provides a conceptual layout of the Experience-Based control idea as it would be performed when the various aspects of the EB-method, as described in this paper, are worked out. The reader is directed to the upper left corner of Fig. 6 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, our Agent will monitor the situation to become aware of 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. 6 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, with the resulting controller and plant models, the Agent runs a simulation and assesses its performance according to the CF. If all is OK, then 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. As a side note, the author has been informed by colleagues who study human processes that humans appear to perform what they call ‘rehearsing’ in such situations; it is easy to deem the ‘simulation’ activity as an equivalent stage.

Figure 6. Proposed model of Experience-Based Control Configuration

The next Section describes two early experiments related to the context-dependent ideas where much of the above is rolled into a single neural network controller; in those cases, the NN was directly trained with specialized information about the context. The training was performed with a DHP adaptive critic method; the controller learned to change the mapping it performs between its ‘regular’ controller inputs (e.g., plant state variable values) and the control outputs as a function of the value the ‘context variable’ input assumes. This worked very well for both settings [18][21]. It is believed that substantial more “mileage” can be obtained using that approach, but the author chose to instead delve deeper into the inner workings of what the NN accomplished, and developed the context-dependent ideas presented herein.

Referring back to the comments made about the multiple-model methods in Section II (Phase 3, subsection D), we comment here that the many issues mentioned in that literature related to the care that must be taken when switching from one controller to another will, per force, have to be taken into account in the present undertaking as well. Relative to the layout in Fig. 6, this comment relates most directly to the activities performed in the two boxes labeled CFA; in particular, before the “Install” action is performed.
IIIb. PRECURSOR CONTEXT-DEPENDENT CONTROLLER EXPERIMENTS

In work at the author’s laboratory related to Reinforcement Learning and Approximate Dynamic Programming, two controller design examples were developed several years ago that are precursors to the Phase 4 type system: 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 discerning a changed context – via the vehicle’s changed behavior – would invoke a “higher level” process to acquire appropriate information, say by sensing the vehicle’s response to small perturbations to selected control inputs, to assist in deciding what to do next. This could include modifying the CF being employed (e.g., safety vs. getting to destination on time), based on current knowledge of Environment and/or Plant parameter values, and then selecting appropriate control actions. Guided by this reasoning, we (the human designers) developed a proxy measurement that provided information about the car/road interaction (tire slip angle) in one case (i.e., data about the environment), and a proxy measurement for location of c.g. in the other case (data about the plant dynamics). In the vocabulary of this paper, these proxy measurements were ‘context variables’. Important to the present discussion is that during the Adaptive Critic controller design process, the context variable was also used as an input to the controller, in addition to the usual plant state variables (cf. Fig. 7). During the AC learning process, in both examples, the controller learned to use a change in context variable value to select a different controller instantiation. Using vocabulary of the present paper, the context discernment was (externally) provided to the NN controller as a value of the context input (cf. Fig. 7). 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, the intent is to take the human out of that part of the design loop. While crafting the context variables employed in the above two 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 we did not yet know how to use Reinforcement Learning to itself accomplish such higher level design, and in particular, in a way that facilitates the con-

Figure 7. Left: A “standard” controller designed via DHP (cf. Fig. 4) using Plant state variables as inputs. Right: Controller designed by DHP with additional Context variable (Plant parameter) input to controller.
controller selection process. That context could be employed as an input during Reinforcement Learning and have it serve as a selection mechanism was, however, demonstrated.

IV. MANIFOLDS

The mathematical construct of manifolds as defined in the geometric topology branch of mathematics provides a useful formalism for conceptualizing issues associated with the experience-based ideas suggested above, and in particular, the issues of representation to be discussed in following Sections. While the mathematical concept of manifolds entails deep mathematical properties, 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) [1]. As a demonstration example, let the set comprise a collection of neural networks (NNs) generated via a NN whose structure and neuron type have been specified, and its adjustable parameters (e.g., 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 (we also use the term ‘index’). When the set part of a manifold comprises neural networks, we use the label neural manifold [38][39]. We refer to an indexed element in the manifold’s set as a point of the manifold. An important caveat when employing a neural manifold such as the one just defined: Care will need to be taken 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 (SAPM) on the NN’s input domain to its output range (called the NN’s Performance Subset [17][19]).

The fixed-weight NN (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, as one typically thinks in terms of the generic quadratic function with coefficients in front of each component, 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 and its coefficients, but the functionality discerned in the trained FWNN is amenable to being described via the neural manifold construct [38]. It has been intriguing to notice that in the representation learned by that FWNN, activation levels of the recurrent connections operate like virtual parameters. They accomplish this by essentially adjusting the bias levels of selected neural elements in the FWNN. We ascribe the role of manifold coordinates to these activation levels – i.e., they provide the indexing mechanism for selecting the current mapping [38]. During operation, when a change in context occurs these biases can be seen to change their value, and then hold constant at the new level until the next change in context occurs.

These and related observations motivated a special construction to assist in “deciphering” the functionality of the trained FWNN of [35]: define a NN in such a way that the parameters of the NN (the weights and biases) are continuous; hold some set of its parameters static (e.g., the connection weights) and allow the remaining ones (e.g., the biases) to be set dynamically. Once the static parameters are fixed, each setting of the dynamical parameters defines a unique NN, so it is possible to ascribe the role of coordinates (indexing mechanism) to the dynamical parameters. This appears to be what the FWNN learned to do. A remaining challenge is to learn how the
FWNN was able to choose the “correct” NN from the manifold’s set, and to for us to then employ such insights for the larger objectives associated with HLLA.

V. CONTEXT DISCERNMENT

The notion of context discernment was defined in Section IA to be the act or process of determining the current values of the context variables (current point in Context Space) appropriate to the task being performed. Also, the Context Space was defined to comprise three (major) subspaces: Plant, Environment, and CF. Both of these definitions are Agent centric, with specifics dependent on whether the Agent is performing System Identification (EBSID) and/or Control (EBC) activities. In any case, current knowledge about the plant, its environment, and the CF are required by the Agent to perform context discernment and selection activities appropriate to the Agent’s current tasks/objectives.

Recall the MRAC and STC classes of Adaptive Control methods described in Section II. The MRAC involves an on-line assessment of a CF in terms of a Reference Model, and the STC class involves an on-line identification to estimate the plant’s parameters. In both cases, the controller’s parameters are then adjusted based on the newly acquired measurements. In terms of the present vocabulary, both of these Adaptive Control methods 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 design a controller (via adjusting the controller’s parameter values), the EB-methods of this paper use this knowledge of context variables to instead select a controller or plant model from the respective experience repository, and do so in an effective and efficient manner.

In order for the efficiency attribute of the methods to be achieved, the proposed process of selecting a controller (or model) from the experience repository must be tightly intertwined with the process of context discernment, and in particular, it will be critical that representation schemas for the two processes be tightly related, and potentially, identical. The neural manifold formalism introduced in the previous Section is employed as a conceptual vehicle for accomplishing this. The label ‘neural manifold’ as defined earlier refers generically to a specific type of collection of neural networks (NNs). In refining this designation to reflect the use to which we put the NNs in the manifold’s set, if the NNs represent the controllers (really, models thereof) in the experience repository, we assign the name policy manifold (alternatively, controller manifold). Similarly, if the NNs represent the plant models used to represent the plant during a System Identification process (this collection of plant models is here also thought of as a kind of experience repository), we assign the name plant manifold. So far in our investigations, we make the underlying assumption that both manifolds are constructed in a manner that they contain solutions for points in the region of interest in the specified Context Space. See Section VII for comments on populating the experience repository.

NB: We have defined the policy and plant manifolds above in terms of neural manifolds; we emphasize here that this is just for conceptual convenience in the present description and the present stage of developing the underlying theory. In practice, it may be practical to employ other kinds of parameterized models; if so, then the parameters of those models are used to define the coordinate space. Two such versions are demonstrated in examples provided in Section VI.

Since the coordinate space for a neural manifold is defined via the weights of the NNs in the manifold’s set, it follows that the indexing schemas for both the plant and policy manifolds are provided by the weights of their respective collection of NNs. So far so good. But, how does one
go about crafting a mapping between, say, the plant manifold’s coordinate space to that of the policy manifold? Such a mapping will be required for our Agent to select a policy based on information about the plant model. More generally, how does one craft an appropriate mapping from the full Context Space (whatever form of representation is employed) to the coordinate system of the policy manifold? Of the plant manifold? The task of answering these questions is assigned to the Higher Level Learning Algorithm (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. In the controls vocabulary, changes in the plant dynamics or in its environment do not necessarily imply a needed change in control policy. Perhaps the ultimate efficiency will be to have the two spaces so coordinated that the mapping turns out being one-to-one.

An integral component of crafting the mapping between 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 for optimal performance, the EBC’s selection task is to pick the “best” controller from the repository (the policy manifold’s set). See Fig. 8. While it is conceivable that such a selection can be made in one step, it may be that the EBC will have to 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 such 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 explicitly provided to HLLA by the engineer for its training process, which for example, minimizes that CF over the course of the trajectory. It may even turn out that such a trajectory appears to “hop around” in the coordinate space – in distinction to a smooth type trajectory we are used to seeing when employing a gradient descent process. Getting back to the first sentence, the efficiency will be directly impacted by the trajectory taken in policy coordinate space, and this aspect will no doubt be affected by the mapping adopted.

There is a further aspect of applying the above conceptual tools to the experience-based notions being formulated that relates even more directly to the efficiency issue. If we make the reasonable assumption that for a given application area 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 that a subsequent observation is associated with a previous experience increases. A goal for the mappings to be developed is to make each component in the set selectable in one or a few steps (the optimal trajectory in coordinate space mentioned in the previous paragraph). Thus, given an observation that calls for an experience existing in the repository, and given an EB-Algorithm that has the capability of gen-
erating 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 recently made in [40] relative to the long-standing 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 the Higher Level Learning Algorithm (HLLA) is as follows:

1. Train the Experience-Based Algorithm (EB-Algorithm) for the EBC and for the EBSID, working with the set of controllers or plant models in the respective experience repositories.
   [While the underlying general theory is not yet developed, the HLLA will in principle also be involved in the generation of, or at least in crafting the representation for, the repositories.]

2. The resulting EB-Algorithms will have developed their own:
   a) respective representations of context;
   b) respective algorithm for discerning context; and
   c) respective mapping from the Context Space (whatever form of representation is employed) to the coordinate space of the policy manifold and of the plant manifold.

3. The resulting EB-Algorithms will have the capability to generate an optimal trajectory in the coordinate space of their respective controller or plant manifold in the process of selecting the “correct” controller or plant model. [Recall distinction of ‘optimal’ as used for the trajectory vs. ‘optimal’ as used for the controller or plant model to be selected.]

4. To remain “mindful” of the desired efficiency issues, particularly as more experience is accumulated, while accomplishing the above.

5. While the HLLA may perform the above off-line, it is to have the capability to operate on-line, with a safe-fail attribute.

6. To address the “when” as well as the “how” questions, where we note that the “when” is itself a context dependent concept.

The major purpose of the HLLA is to train the EB-Algorithm, and to do so in a way that the EB-Algorithm manifests the attributes listed in the second and third items in the above list. This begs the question of what training methodology to use. No general answer is offered here, but, given that the author’s research is heavily related to application of Reinforcement Learning methods, in particular the DHP version of the Adaptive Critic method, the suggestion is made here that this method appears as an excellent candidate. So, is there a way to apply the DHP Adaptive Critic method in a manner that will allow it to solve this higher-level task? To help answer this question, we point out that when setting up an application of the Adaptive Critic methods, the **sole vehicle for providing information to the method** about the objectives of the application is the Criterion Function (CF). Beyond that requirement, any application that can be formulated to have components that fill the **roles** of “plant” and “controller” (this doesn’t have to...
be literal) are amenable to application of the Adaptive Critic method to train up a near-optimal “controller”.

Thus, the key to applying the Adaptive Critic method to achieve the above tasks is to craft an appropriate Criterion Function. 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 the list above, and in addition, pay attention to some of the issues to be addressed in Section VII.

As described in Section II, the HLLA has two important sub-tasks: train the Experience-Based System Identifier (EBSID), and Train the Experience-Based Controller (cf. Fig. 5). As was mentioned there, the EBSID is a prerequisite to the EBC, as it performs the context discernment function within the Plant sub-space of the Context Space. In the remainder of this Section, examples of using a DHP Adaptive Critic implementation of the HLLA concept (called Contextual Reinforcement Learning [13][40]), demonstrate successful application of the notions put forth in this paper to the training of an EBSID algorithm for three different situations (later, the EBSID will be embedded in the box labeled CDN in Fig.9).

The following three examples each exemplify some portion of the above, relative to the System Identification genre of tasks.

**EB System Identification Example 1: QUADRATIC MAPPING**

As a partial example of the material presented so far, consider a family of quadratic mappings, and consider applying an input to a “black box” whose output is determined by a specific one of these quadratic mappings. Assume that occasionally the quadratic mapping instantiated inside the black box changes to another member of the family. As an application of the above ideas, populate a neural manifold’s set with neural networks (NNs) that each perform (an approximation of) a single one of these quadratic mappings, and per the definition of neural manifold given earlier, the NN weights serve as the coordinate space. Call this neural manifold the **model manifold**.

We assign to the HLLA the task of generating an EB-Algorithm for the EBSID box in Fig. 5a. 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 quadratic mapping 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 the criterion is to minimize the squared-error between the data observed from the black box and data generated by the candidate NN (note that this is not the ‘higher level’ CF).

From the system identification perspective, the Context Variables are 1) parameters of the quadratic mapping and 2) parameters adopted for the above CF (equivalent to the A and C boxes in Fig.1 in Section I). The point in the Environment sub-space 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 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 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 above task in some optimal way (the ‘higher-level’ CF is defined for this optimality). In the present case, a DHP Adaptive Critic approach is employed. Once the EBSID is designed (via training), it is then employed to accomplish the selection task (for system identification).

To set up the training process, we first adopt the perspective of a different Agent, one whose task is to *train* the EBSID (we have thus invoked a two level hierarchy here – hence the term
‘higher level’ in the label HLLA). We in turn define a new Context Space corresponding to this perspective. The two key sub-spaces of this new Context Space are: 1) the family of quadratic mappings (the role of Plant here), and 2) the new higher-level training CF. The training CF’s definition is to capture the requirement that the resulting EB-Algorithm is to have the ability to sequentially specify an optimal trajectory through the neural manifold’s coordinate space in the process of selecting the “correct” NN.

During the training process, if an incorrect NN is currently selected, a set of observations / measurements dictated by the training CF are 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. We note that each observation provides only partial information, so it is unlikely that one step will suffice. After each NN instantiation, new data is 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 1) minimize the accumulated squared error during the trajectory, or 2) minimize the number of steps in the trajectory. In a controller setting, stability requirements could be embedded in the CF.

Once the EB-Algorithm design is completed, when the EBSID discerns that a new quadratic mapping has been instantiated (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 experiments of the above design were carried out in late 2003, and reported in [38]. Further experiments are underway and results will be provided in future publications.

**EB System Identification 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 parallel to the track; the cart’s top 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° from vertical). This type of plant has been useful due to its conceptual simplicity and at the same time characterized by nonlinearities that are not trivial. 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 enables us to construct a plant manifold, implicitly populate its set with various instantiations of the analytic model, and define 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 system identification process could be used, for example, as the prerequisite step to controller selection.

We again assign to the HLLA the task of generating an EB-Algorithm for the EBSID box in Fig. 5a. This EB-Algorithm is to observe a sequence of data emanating from the system, and select the model from the plant manifold that corresponds to the pole-cart system that is 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 quality of fit of the data emanating from the candidate models; assume the criterion is to minimize the squared-error between the data observed from the pole-cart system and data generated by the candidate model. Fig. 9 below contains the basic
architecture for context discernment, and in addition, the basic architecture for training the EB-Algorithm that performs the context discernment, called CDN in this figure.

The context discernment portion is the left 2/3 of Fig. 9, and the HLLA “trainer” comprises the components in the upper right of the figure. The training process is the DHP Adaptive Critic class of Reinforcement Learning, whose basic structure was shown in Fig. 4. Most of those details are not repeated in Fig. 9. An outline of the experiments performed with this plant is described next; more details may be found in [13]. The notation of that paper is used in this description. The variables subscripted with an ‘A’ represent the Actual pole-cart system being controlled; those subscripted with a ‘D’ represent Discerned values resulting from the context discernment process. At time $t$, the pole-cart system is in state $R_A(t)$, comprising the horizontal displacement of the cart from a specified reference point, $x(t)$, the horizontal velocity of the cart, $dx(t)/dt$, the angular displacement of the pole from the upright position, $\theta(t)$, and the angular velocity of the pole, $d\theta(t)/dt$. The control signal, $u(t)$, is the magnitude of the force applied to the cart parallel to the track. Positive control signals indicate a force to the right and negative control signals indicate a force to the left. The context vector contains parameters corresponding to the length and mass of the pole.

Referring again to Fig. 9, the current estimated values of the mass and length of the pole in the pole-cart system (the discerned context) constitute the vector $C_D(t)$ [in the vocabulary of this paper, this would be $\theta_D^P(t)$] and the actual values constitute the vector $C_A(t)$ [$\theta_A^P(t)$]. NB: The training process never requires explicit knowledge of $C_A(t)$. The current state information $R_A(t)$ and a control $u(t)$ are applied to the model (resulting in a transition to state $R_D(t+1)$) and to the pole-cart system (resulting in a transition to state $R_A(t+1)$). The difference between $R_D(t+1)$ and $R_A(t+1)$ is designated $D(t)$. If the value of $D(t)$ is non-zero, this indicates that the current model is not correct and needs to be adjusted. The ensemble of data comprising $C_D(t)$, $R_A(t)$, $u(t)$, and $D(t)$ and are presented to the Context Discerning Network (CDN). The CDN then provides an adjustment, $\Delta C_D(t)$, to $C_D(t)$ such that $C_D(t+1) = C_D(t) + \Delta C_D(t)$. The goal of the CDN is to iterate the above process and to provide a sequence of $\Delta C_D(t)$s such that $C_D(t)$ ultimately selects $C_A(t)$. If the model structure is crafted “correctly” at the beginning, the selected model’s behavior will accurately match the behavior of the pole-cart system when $C_D(t) = C_A(t)$ [$\theta_D^P(t) = \theta_A^P(t)$]. When this occurs, we say the values of $C_A(t)$ have been discerned. The task of the HLLA is to train the CD-Algorithm for CDN (the stand-in for CDSID in this example) to accomplish the just described activity.

The method used to train the CDN borrows heavily from applications of Reinforcement Learning in the domain of controller design – specifically the DHP Adaptive Critic method. Only sketchy details of the approach are included here; various chapters in [41] may be consulted for more details. As mentioned in Section III, a Criterion Function is defined to represent the control objectives, say, in terms of “costs”; for the discrete-time regulator case, the CF is defined as the sum of all future costs required to get from the current plant state to the desired end state (called cost-to-go). By redefining a few variables, we re-purpose this framework and apply this same methodology for the task of context discernment. For example, we let $C_D(t)$ be the state of the plant that is to be “controlled” at time $t$, and the corresponding control at time $t$ to be $\Delta C_D(t)$. With these definitions, $R_{plant}(t) = C_D(t)$ and $R_{plant}(t+1) = C_D(t) + \Delta C_D(t)$. Next, we define a primary utility $U(t)$ to be $D(t)^2$, the squared error between $R_A(t+1)$ and $R_D(t+1)$. Looking back at Figure 5 and considering this new framework, the CDN plays the role of controller, and the summing node that follows it plays the role of plant. The (higher level) CF employed by HLLA is to
minimize the summed values of \( D(t)^2 \) over time as the model selection process continues, via adjustment of the model parameters. Not only is the CDN to learn a policy (CD-Algorithm) that enables it to discern the current system context (that is, determine a model whose behaviors match those of the system), but it is to do so in a way that meets this (higher level) optimality criterion.

![Diagram](image)

Figure 9. Basic architecture for Context Discernment and for training CDN (after [13]).

Details of the actual experiments are given in [13]. The test results for the trained CDN are reproduced below in Fig. 10 and Fig. 11. The mass and length of the pole in the pole-cart system \((C_A(t))\) were randomly reset after every 50 iterations. The flat lines in Fig. 10 correspond to these “actual” parameters. Based on deviations between \(R_A(t+1)\) and \(R_D(t+1)\) at each step of the testing process, the CDN produces a correction to \(C_D(t)\). The curved lines in Fig. 10 correspond to these discerned parameters. As can be seen, the values of \(C_D(t)\) converge rather well to the values of \(C_A(t)\). In some cases, this convergence happens in as few as 10 iterations, and since the sampling time for the process is set at 0.02s, it follows the CDN is able to discern the mass and length of the pole in the pole-cart system in as little as 0.2s. Fig. 11 shows the squared error between \(R_A(t+1)\) and \(R_D(t+1)\) at each step of this test. Clearly, a jump occurs in the error between the actual and expected state trajectories every 50 steps when new values for the mass and length of the pole are instantiated. As may be observed, this error quickly drops to near zero as the context discernment process refits the model to the changed pole-cart system.
EB System Identification Example 3: Neural Network as PLANT

For the final example of this section, consider an NN of the type described at the end of Section IV. Namely, the parameters of the NN (the weights and biases) are continuous; its connection weights are held fixed and the biases are allowed to vary; each setting of the biases defines a unique NN. Generate a set of such NNs by varying the bias values; let the biases be the coordinates, thereby constructing a neural manifold. An entirely equivalent procedure may be used here as for the previous example. Thus, a configuration identical to that of Fig. 9 may be constructed, replacing the contents of the box labeled “Pole-Cart Model” with NNs of the type just described. Similarly, replace the contents of the box labeled “Pole-Cart System” with a specific one of these NNs. Re-label the two boxes accordingly (for simplicity here, call them “Model” and “System”). For this construction, the task of the Context Discerner is to determine which of the NNs in the repository corresponds to the specific one currently instantiated in the System box of Fig. 9 [39].
As mentioned, the process undertaken by the HLLA for this example is entirely analogous to that used in the previous example. Fig. 12 shows test results of the context discernment process after training, where the test changed the instantiated NN in the System box every 100 iterations. We observe that at each change of NN in the System box, the context discerner successfully noted the change and rapidly selected the corresponding NN from the repository. The error plot demonstrates a spike in error value when the change occurs, followed by a quick recovery. As for the previous example, the context discerner uses 10 or fewer observations of the context to make the correct selection. The top graph of Fig. 12 shows the parameter guesses; we note that at each change of instantiated NN in the System box, the shift in parameter guess is very fast; qualitatively, the results for this example closely mirror those demonstrated for the pole-cart example. Variations of this experiment included NNs with different ratios of fixed to variable parameters, with good success involving some 40 parameters, and ratios of about five to one. In addition, an RBF NN structure was also successfully explored.

Figure 12. Demonstration of Context Discernment in response to change in NN plant parameter values (context change) at every 100th iteration. [39]

Another observation to be made here relates to an aspect of neural manifolds mentioned in Section IV, namely, 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”, so to speak, when the process was being tested, 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 Criterion Function only looked at the relative performance of the outputs of the plant and candidate models. This brings to the fore our comment at the beginning of this Section that the Criterion Function is the sole vehicle for providing information to the method about the objectives of the application.

VII. DISCUSSION

A. Experience Repository

The experience repository of this paper, 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 available to the EBSID for performing a system identification task.

So far in this paper, we have not directly addressed the issue of how the experience repository gets populated in the first place. In principle, when the repository is empty, “experience” does not yet exist. In practice, the repository will likely have to 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 Phase 1, Phase 2, or Phase 3 methods to generate a set of controllers for a selected 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, however, is the significant technical difficulty associated with coming up with 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 systems identification task, and the collection of components in the controller repository for the control selection task, and at the same time, characterize the Context Space in a way that useful mappings between it and the two repositories can be crafted. The success of the entire Experience Based enterprise suggested in this paper will rest with success in these endeavors, i.e., in crafting appropriate coordinate spaces of the manifolds for the two 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.

Whereas we have stipulated that one of the tasks of the HLLA is to in fact develop such representations and mappings, and indeed 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 controls field, continued assistance will be required of the human designer for inventing the schemata that will be 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 above issues when initializing the repositories that 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 meta-level 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-meta-level role in crafting the manifolds as we did. For the quadratic equation example described there, the repository is defined via the quadratic equation; the manifold formalization is crafted such that its set portion in principle contains instantiations of the equation, and the coordinate space comprises the coefficients of the equation. Our role was invoked in the sense that we knew (at another level up) the principle we wanted to demonstrate, and crafted the situation accordingly. Similarly for the pole-cart system identification example: the model repository for the pole-cart system is defined via an analytic equation; the set portion of the manifold is generated via the analytic equation, and again, the coordinate space is constructed via the equation’s coefficients, and we stipulated ahead of time which of the context variables (pole length and mass) were to be focused on for the context discernment task. In the last example of Section VI, 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; the manifold was a neural manifold as defined earlier in this paper. In all three cases, 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, a Reinforcement Learning process could be configured to implement an HLLA to develop a context discerner with effective and efficient selection capability. That is, it was demonstrated that a Reinforcement Learning process could be employed to train an Agent to learn Component B of definition 7 given in Section IA; the particular HLLA employed so far is called Contextual Reinforcement Learning [13][38] [39]. While the demonstrations in this paper are for the systems identification part of the task, promising work is underway to develop demonstrations of the controller selection task as well (beyond that described in Section III), and will be reported in future publications.

B. Model Refinement / Generalization

In various places in this paper we have used the phrase “with possible refinement” when describing the controller or plant model selection process. The intention behind this phrase may be tied to the notion used in neural networks called generalization: to provide an output for an input not seen during training, and in particular, “good” generalization, with the obvious connotation. The latter is akin to what we usually mean by the term ‘interpolation’, or even ‘extrapolation’. In the examples provided in Section VI, the manifolds were crafted in a way that the interpolation/generalization/refinement in a sense came rather easily, by virtue of the way the manifolds were crafted. 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 interpola-
tion, then perhaps the HLLA would need to invoke an on-line re-design method, such as one of the Learning Control methods mentioned in the Phase 3 discussion.

There are various aspects of the repository that impact the quality of generalization and the efficiency/effectiveness attribute mentioned throughout the paper, and are useful to consider when constructing the repository. For example, larger population count in the repository impacts two important aspects: generalization and selection response time. The generalization aspect is influenced by virtue of more and more constraint (of the good kind) being present; for neural networks, this added constraint is known to increase chances of better generalization. Also, this increased ‘richness’ of the set of solutions in the repository is expected to result in fewer steps in the coordinate space for the selection process. Any a priori knowledge available relative to the application tasks may be employed to build in such constraints when parameterizing the context and controllers, and while populating the repository.

Using the vocabulary of the special NN structure described at the end of Section IV, we characterize one way that the HLLA itself could embed more and more constraint into a manifold as learning progresses as follows: start out with all the weights of the NNs in a neural manifold being dynamic, and as more and more information about the application domain is learned, start changing the status of selected weights to that of being static. In essence, this is exactly what we inferred that the FWNNs described in Sections I and IV accomplished.

C. Prespecification

A comment was made in Section I related to the prespecifications entailed in the Adaptive and Learning Control methods, and suggested that “the EBC seeks to go beyond the limitations implicit in the need for such prespecifications, or at least to reduce their number.” We now revise that comment to say that the prespecifications 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 based on the loosely described meta-meta level notion above, and in particular, use of a priori information about the engineering application for which the EBC is being developed, to tailor the various parameterizations and definition of the coordinate spaces in a way that predisposes successful context discernment and controller selection/refinement. One extreme of such tailoring was demonstrated in the examples of Section VI; progress in the field will be measured by the amount we succeed in moving away from this extreme.

VIII. CONCLUSION

There are many issues to still 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 of current 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 experience-based 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 Experience-Based 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 Experience-Based Controller and Experience-Based System Identification 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.
ACKNOWLEDGEMENTS

The author wishes to acknowledge the creative role filled by the various graduate students who have worked in his Laboratory, both present and past. These include, but not limited to, in alphabetical order: Michael Carroll, Lars Holmstrom, Steven Hutsell, Bryan Johnson, Greg Lankenau, Josef Lotz, Karl Mathia, Shari Matzner, Jay McCarthy, Christian Paintz, Alec Rogers, Andreas Rustan, Larry Schultz, Steve Shervais, Andrew Toland, and Sangsuree Vasupongayya. Particular acknowledgements go to my Lab Managers: Thaddeus Shannon and more recently, Roberto Santiago.

The pole-cart experiments in Section VI were performed by Lars Holmstrom; the quadratic mapping and NN as plant experiments by Roberto Santiago. The Contextual Reinforcement Learning method is being developed as part of the latter’s dissertation research.

REFERENCES


