

## 2009 Special Issue

## Adaptive dynamic programming approach to experience-based systems identification and control

George G. Lendaris\*

Portland State University, Systems Science Graduate Program, Electrical & Computer Engineering and NW Computational Intelligence Laboratory, SySc, P.O. Box 751 Portland, OR 97207, United States

## ARTICLE INFO

## Article history:

Received 5 May 2009

Received in revised form 1 June 2009

Accepted 25 June 2009

## Keywords:

Approximate Dynamic Programming (ADP)

Experience-based identification and control

Context discernment

Systems identification

Optimal control

Neural networks

## ABSTRACT

Humans have the ability to make use of experience while selecting their control actions for distinct and changing situations, and their process speeds up and have enhanced effectiveness as more experience is gained. In contrast, current technological implementations slow down as more knowledge is stored. A novel way of employing Approximate (or Adaptive) Dynamic Programming (ADP) is described that shifts the underlying Adaptive Critic type of Reinforcement Learning method “up a level”, away from designing individual (optimal) controllers to that of developing on-line algorithms that efficiently and effectively select designs from a repository of existing controller solutions (perhaps previously developed via application of ADP methods). The resulting approach is called Higher-Level Learning Algorithm. The approach and its rationale are described and some examples of its application are given. The notions of context and context discernment are important to understanding the human abilities noted above. These are first defined, in a manner appropriate to controls and system-identification, and as a foundation relating to the application arena, a historical view of the various phases during development of the controls field is given, organized by how the notion ‘context’ was, or was not, involved in each phase.

© 2009 Elsevier Ltd. All rights reserved.

## 1. Introduction

In a recent paper, the notion called *Experience-Based Identification and Control* (EBIC) was put forth, with the objective of bringing to the attention of the controls community “. . . some hopefully seminal ideas that will inspire and guide even greater application of Adaptive Dynamic Programming (ADP) methods . . .” (Lendaris, 2008). In the present paper, this objective is further pursued, this time in the context of the neural networks community. While the underlying ideas of necessity are repeated, where the previous paper focused on some of the more abstract/mathematical aspects, this paper extends into neural network implementations of selected entry points into the proposed approach.

Development of the EBIC approach evolved out of the widely-held desire in the research community to achieve more **human-like capabilities** in identification and control technologies. It is generally understood (at least intuitively) that human performance levels for such tasks depend on effective and efficient use of *experiential* knowledge. While the control engineering field has indeed accumulated remarkable achievements, even exceeding human control capabilities for some applications, substantial additional progress is still needed towards 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 (even novel ones), and importantly, to do so effectively and efficiently.

Two observations concerning human abilities are pertinent here:

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 close-to-optimal performance on the new task, based on the previously learned skills (i.e., effective selection from experience).
2. The more knowledge a human attains, the speed and efficiency of performing tasks are improved (in a relevant environment) – in stark contrast to Artificial Intelligence systems thus far developed, wherein the more knowledge acquired (typically stored as “rules”) the *slower* the decision/action processing.

For computational Agents to achieve the above-noted kind of effectiveness and efficiency, we posit they will have to implement the equivalent of *experience*. The approach here assumes three components for such an Agent system:

- (1) A collection of models appropriate to a given engineering application (models are of plants or controllers, depending on whether they are doing control or system identification).
- (2) A characterization of this set of models in a form that facilitates accessing the models.

\* Corresponding author. Tel.: +1503 725 4988.

E-mail address: [lendaris@syc.pdx.edu](mailto:lendaris@syc.pdx.edu).

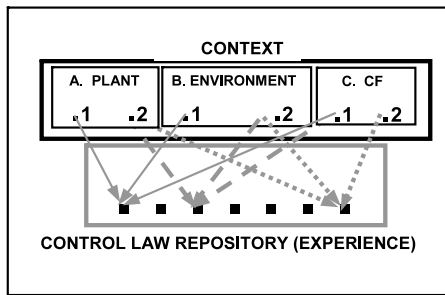


Fig. 1. Schematic of CONTEXT and corresponding control law REPOSITORY.

- (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.

These three components are here deemed fundamental to what is meant when humans are said to have attained *experience* related to a class of identification/control tasks, and note that implicit in these requirements is a *memory* property for the Agent.

To fix ideas, consider a control-engineering setting in which a plant, environment, and control objective are provided to a (human) 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 below), 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.*

### 1.1. Context

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

The notion of *context* is here formulated to comprise three components: (1) plant, (2) environment, and (3) objectives plus associated performance criteria (labeled **CF**). See Fig. 1. The 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.

As an intuitive entrée to this use of the term *context*, think of driving on a clear afternoon (1) on dry pavement, or, (2) on an icy pavement. The general driving skills in both scenarios are the same, but selected adjustments are needed to your control law and/or decision logic. If instead of a change in the environment (road conditions) there is a change in attributes of the car (plant) – e.g., a slightly flat tire – new adjustments are needed for the car to perform your desired maneuvers; driving your friend’s car that day rather than your own is another example. A third consideration involves performance criteria (CF); e.g., in a road race, a criterion is to minimize time, but for an elderly relative on an excursion, maximizing comfort is more likely. Each of these examples may be represented via a triplet of lines from the Context in Fig. 1, pointing to a corresponding control law in the Repository (and we note, this selected control law could be locally adaptive as well).

### 1.2. Experience

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

### 1.3. Representation and mappings

The context-discernment and controller-selection processes are directly impacted by the representations crafted for context and repository, and in particular, by the indexing schemas defined for the various sets involved. Theoretical considerations related to such representation and mapping issues uncover a myriad of difficulties. A candidate tool deemed useful for delving into the associated issues is the formalism of *manifolds* from geometric topology, where the manifold comprises a *set* and a *coordinate system*. In the present setting, **the manifold’s set is to comprise the experience repository**, and the **manifold’s coordinate system is to be a searchable indexing mechanism with useful “nearness” properties**, more details in Lendaris (2008). In the author’s preliminary work, this approach provided the framework in which a novel concept for applying Reinforcement Learning (the previously mentioned HLLA, Higher Level Learning Algorithm – see Section 3) was developed for evolving the nascent experience-based ideas.

**The key idea** for HLLA is to **re-purpose** the Reinforcement Learning (RL) method 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 (repository) of such designs for a variety of related contexts is provided, 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). A detailed set of definitions of the terms employed for the above framework are given in Lendaris (2008).

The *selection process* is to be triggered by the Agent becoming aware that a *change in context* may have occurred. This is followed by the Agent seeking information about what has changed – a process here called *context discernment*; the latter process typically entails a form of system identification (SID), also enhanced via *experience*. Two examples are given in Section 5.

### 1.4. Overview summary

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. Beyond this base level, it is further posited that context discernment is fundamental not only for the selecting aspect mentioned above, 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 will no doubt be fundamental to such considerations; a concept that might be called *Context Space Hierarchy* would powerfully assist in this endeavor. Development of such a concept is deemed an important issue for future work.

## 2. Historical perspective of context and the controls field

While the existence of control devices dates back to antiquity (call it PHASE 1 for the controls field), Maxwell’s use of differential equations to analyze the dynamics of a flyball governor (ca. 1870) (Lewis, 1992) that had been invented by James Watt in 1788 (Watt, 2005) may be said to have ushered in a new phase for the controls field (call it PHASE 2). Whereas intuition and inventive genius may be said to characterize Phase 1, mathematics played a fundamental role in Phase 2, 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 has also been significant, not only relative to implementation, but also as

a driver and motivator for various mathematical and algorithmic developments as well.

An important limiting aspect of the Phase 2 methods, however, is that the controller so 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. This phase includes at least the following well known design methods: Classical Control, Modern Control, Optimal Control, Stochastic Control, and Robust Control, e.g., (Bertsekas & Shreve, 1996; Dorf & Bishop, 2001; Goodwin, Graebe, & Salgado, 2000; Lewis, 1992; Lewis & Syrmos, 1995; Mosca, 1995; Ogata, 2000; Phillips & Harbor, 2000; Zhou & Doyle, 1998). Even though the progression of these methods has employed ever more sophisticated mathematical tools and insights, and yielded 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. To somewhat accommodate the latter, the designs in this category are often crafted to have low sensitivity (“robustness”) to selected changes in plant or environment parameter values. While these controllers accommodate certain context changes, this is accomplished by virtue of *margins* in controller design rather than by on-line changes in the design itself.

The Phase 2 Optimal Control methods have achieved great success for linear systems (via Riccati type equations), but the success is mediated for some applications, because the methods require a knowledge of plant dynamics, and are not readily amenable to on-line applications. The results for non-linear systems are more measured, as the available solution methods (e.g., via Hamilton-Jacobi type equations), which also require a knowledge of plant dynamics, are not as generally applicable.

The Phase 2 limitation mentioned above became prominent in applications where the context changes so much during operation that the fixed controller designs resulting from the Phase 2 methods were not sufficient. A design path emerged that accommodates context variations via on-line instantiation of different controller designs based on the observed variations, ushering in another new phase in the development of the controls field (call it PHASE 3).

A large segment of the methods developed for this Phase 3 may be labeled PARTITIONING, as they partition a nonlinear operating region into approximately linear regions, and develop a linear controller for each. These methods may be said to focus on changes in the environment component of the context in which the control system operates. 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 other technology sectors as well, e.g. artificial intelligence, neural networks, Fuzzy logic, statistics, etc. Thus, the partitioning methods have appeared under a variety of labels – e.g., multiple models, piecewise models, mixture of experts, Fuzzy models, local regression, etc.

Another class of methods developed for Phase 3 may be labeled ADAPTIVE CONTROL and LEARNING CONTROL. Both operate over a specified pool of controllers, and they converge on a design within this pool based on a sequence of state and/or environment observations and/or performance evaluations. In the Adaptive Controls case, the engineer specifies a set of available controllers (via parameterized mathematical models) and an algorithm to select from this set based on observations. In the Learning Controls case, the engineer specifies a parameterized controller structure (e.g., of an NN), and a corresponding algorithm to incrementally adjust the parameters to converge on an appropriate design as each new situation is encountered.

**Of significance here is that such Adaptive and Learning Control methods do not retain memory of solutions as they are achieved.** Every time the context changes, the adaptive/learning process is reinvoked to come up with a solution. The two methods are distinguished primarily by the amount of *a priori* information embedded in their respective pool of controllers, and the off-line vs. on-line aspects. Apropos the latter, 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 on-line manner (Astrom & Wittenmark, 1984; Mosca, 1995; Sastry & Bodson, 1989). Historically, such guarantees were not available with Reinforcement Learning methods, but recent extensions demonstrate selection methods that permit on-line operation as well, see Lendaris (2008, 2009) for references.

In Lendaris (2008), I posited a next phase for the development of the controls field (PHASE 4), and dubbed it *Experience Based Identification & Control*. The focus defined for this new phase relates to endowing our technology with more human-like capabilities for identification and control – in particular, to employ **experience** for model selection (controller or plant, depending on Agent’s task), and to do so **efficiently** and **effectively**.

Fundamental to the posited Phase 4 methods will be *selection strategies* (of solutions existing in the experience repository), to be designed by optimization methodologies for effectiveness and efficiency. The HLLA mentioned earlier employs the Adaptive Dynamic Programming (ADP) method, and applies it at a “higher level” to develop optimal strategies for selecting controller designs from the experience repository, in contrast to the more typical application of ADP to develop optimal controller designs (that could populate the experience repository) (Lendaris, 2008).

While there is considerable conceptual overlap between the Phase 3 methods and those posited for Phase 4, nevertheless, it is claimed there are sufficient distinctions among the respective *guiding principles* that explication and pursuit of the EB ideas are warranted, as some of the distinctions seem crucial to the **scalability issues** (e.g., size of the repository/memory vs. speed of selection) to be faced when attempting to develop human-level performance by a computing Agent. Further, whereas multiple-model methods have historically used linear models (except more recent neural network ones), no such constraint is involved in the proposed EB method.

For more details and a substantially longer list of citations relevant to this section, see (Lendaris, 2009).

### 3. Experience-based control preliminaries

A starting point for accomplishing Experience-Based Control (EBC) is to come up with *learning methods* to (1) accumulate a set of “relevant” control 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 during online operation of the system, in response to changes in its context. The reader will recognize the similarity of these with the three components listed for EBC in Section 1. 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 (simply) 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.

For a bird’s-eye-view of the new components entailed in the Experience-Based notions, refer to the next two figures. Fig. 2 represents a generic structure for the Phase 2 Adaptive and Learning methods. The algorithm parts of the Adaptive Control and of

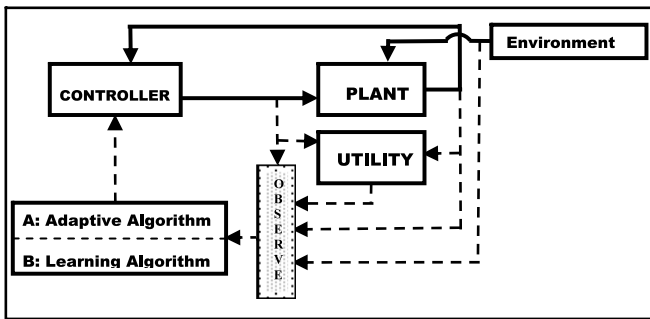


Fig. 2. Generic structure for A: Adaptive Control or B: Learning Control.

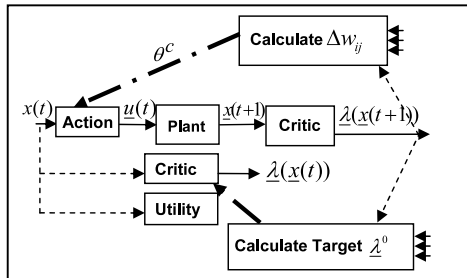


Fig. 3. General layout of Adaptive Critic DHP structure.

the Learning Control methods take current observations and employ them as a basis for making modifications to controller/policy parameter settings. 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. These methods were historically employed for off-line operation, but recent developments allow on-line performance as well. For fuller exposition and examples of Adaptive Control, see (Sasthy & Bodson, 1989).

We provide here an overview description of Learning Control that employs a basic form of Reinforcement Learning/Approximate Dynamic Programming (ADP), namely, the Dual Heuristic Programming (DHP) method. Fig. 3 provides a diagrammatic view of the DHP process; due to space limitations, the mathematics of this method are not reproduced here, but may be found in Lendaris and Neidhoefer (2004, chap. 4) and other papers cited there. It is common in the ADP type of Learning Control 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 controller (or policy) parameters,  $\theta^c$ . 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 parameter values using the ADP procedures (gradient descent, etc.).

In Fig. 3 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

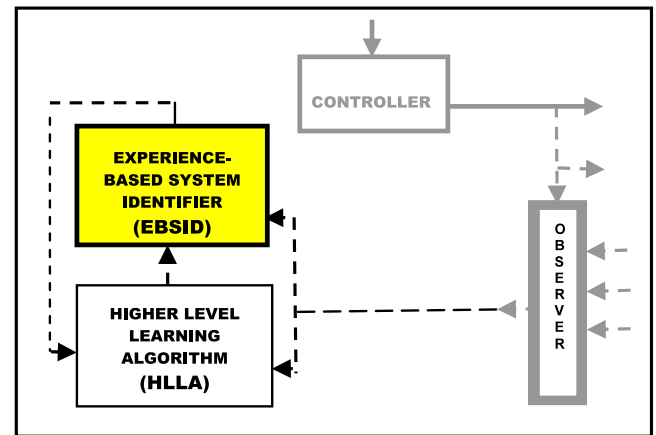


Fig. 4a. Generic Structure for developing Experience-Based Systems Identifier (EBSID).

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 item 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. Operational details for this method may be found in Lendaris and Neidhoefer (2004, chap. 4).

We turn now to some distinctions between the HLLA notion and the Adaptive and Learning Control methods, and focus on two main aspects of the HLLA process, namely, system identification (employed as a component of *context discernment* performed on the Plant sub-space), and *controller selection*. Looking first at Fig. 4a, note that the box of Fig. 2 labeled “Adaptive Algorithm / Learning Algorithm” has been replaced with two boxes, one labeled EBSID (Experience-Based System Identification) and the other HLLA (Higher Level Learning Algorithm). Both of the boxes are significant. The HLLA 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; this algorithm is given 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 noted between Fig. 2 and Fig. 4a, the connection to the controller box in Fig. 1 is also removed for Fig. 4a. This reflects the view that the systems identification task does not normally feed into the controller, at least not directly.

Proceeding next to Fig. 4b, note that the aforementioned box of Fig. 2 has again been replaced with two boxes, one labeled EBC (Experience Based Control) and the other HLLA. This time, the (trained) EBSID is included as a component of the EBC. The HLLA works with a set of policies available for a given engineering task 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; this algorithm is again given 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

Fig. 4b. Generic Structure for developing Experience-Based Controller.

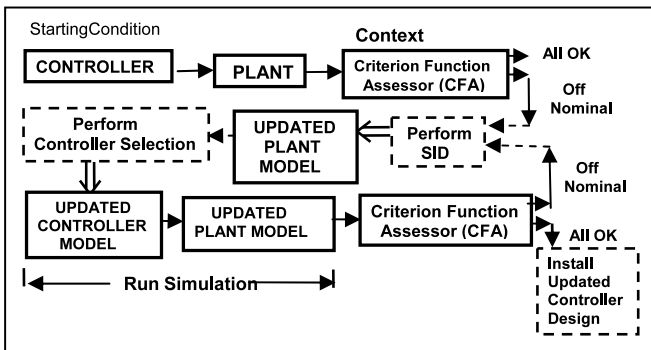


Fig. 5. Conceptual layout of Experience-Based Control (EBC) process.

effectively' becomes instantiated as an optimal search trajectory in the coordinate space used to index the set of (relevant) controllers.

We comment that the HLLA process as described above appears as 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, potentially implement another higher-level notion for the process, that of “safe-fail” (in distinction to “fail-safe”), which is often 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.

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

### 3.1. Conceptual configuration of experience-based control process

Fig. 5 provides a conceptual layout of the Experience-Based control idea as it would be performed when the various aspects of the EB-method are worked out. The reader is directed to the upper left corner of Fig. 5 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 monitors 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. 5 this action is labeled Perform SID (hence specializes the figure to a control setting with changing

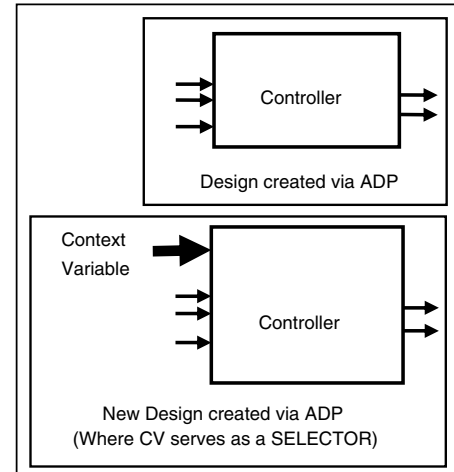


Fig. 6. Upper: A “standard” controller designed via DHP (cf. Fig. 3) using Plant state variables as inputs. Lower: Controller designed by DHP with additional Context variable input to controller.

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 human’s appear to perform what they call ‘internal rehearsing’ in such situations; it is easy to deem the ‘simulation’ activity as an equivalent stage.

### 4. Phase 4 – Approach 1: Embed experience in NN repository, with context variable as index

The detailed definitions and framework given in Lendaris (2008) and recapitulated above were constructed to facilitate theoretical/analytical exploration of the many representation and mapping issues implicit in the suggested experience-based approach. There are many potentially complex issues yet to be worked on – say, via the suggested formalism of manifolds from geometric topology.

In the present paper, however, we describe an ADP approach to the experience/selector idea wherein the mapping shown in Fig. 1 going from context to controller repository is fully embedded inside associated neural networks, and somewhat sidesteps the representation/mapping issues mentioned. In general, numerous such issues remain, but this approach has much power to offer for appropriate problem scenarios. The approach entails a focusing of the human designer’s attention on the notions of context, context discernment, Context Variable (CV), and associated design of the CVs to input to the NN during ADP training.

Descriptively, the idea is to embed multiple memories (“experience”) in the NN, and index them via the CV inputs. This entails predefining mechanisms for context discernment and employs the CV input to NN controller during ADP training. During operation, the NN has learned to employ the CV input(s) as an “index” to the embedded distinct controller designs for different contexts. The experiments reported below provide evidence that a NN is capable of employing such CV inputs to specify different mappings to be performed by the NN between the state-variable input nodes to the controller-output node(s) for different contexts (i.e., as a function of CV) – cf. Fig. 6. Put alternatively, the CV serves as an “index” to different memories in the NN related to mapping the state-variable inputs to the NN’s outputs. The selection and

definition of the CV for each case was informed by the underlying experience-based ideas, and the training was performed with a DHP adaptive critic method. Those explorations are described in the next two subsections. In both cases, it was reasoned that an experienced human operator, upon discerning a changed context – say via the vehicle’s changed behavior – would invoke a 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 to serve as context variable CV that provided information about the environment in one case (tire slip angle) and about plant dynamics (location of center-of-gravity, c.g.) in the other case. Important to the present discussion is that with the added context variable as an input during training (and subsequent use), the Adaptive Critic designed controller learned to use a *change in CV value to select a different controller instantiation*. As an aside, again comment that when using NNs in this way, the Context Discernment related to that aspect of context being measured by CV is performed within the NN, and thus the more general theoretical aspects raised in Section 1.3 about dealing with appropriate mappings and indexing mechanisms from context to the repository do not need explicit attention – the NN “takes care of it for you”. A controller endowed with such CV input(s) is dubbed **Contextually-Aware Controller**.

#### 4.1. Phase-4 Approach-1 applied to steering control of Autonomous 2-Axle, 4-wheel terrestrial vehicle (with context variable input).

The objective in this example is to design a controller that accomplishes selected front-wheel steering maneuvers for a two-axle terrestrial vehicle. For this demonstration, the (autonomous) vehicle is assumed to be traveling along a straight highway, and its steering controller receives a command (say, from a higher level component of its overall guidance & control system) to “change lanes”.

The condition of the vehicle just prior to initiation of the steering maneuver is represented as a velocity vector for the center-of-mass of the vehicle. The controller’s task is to specify a sequence of steering angles which will provide the accelerations (forces at the front wheels) needed to effect a sequence of changes in the orientation of the vehicle’s velocity vector appropriate to the lane-change steering task. Forward velocity of the vehicle is required to remain approximately constant; this is accomplished via controlling angular velocity of the wheels – as without such control, the vehicle tends to slow down during lane changes. Accordingly, the controller effectively specifies two accelerations: one to accomplish a change in orientation of the vehicle’s velocity vector (via a change in steered angle), and one to accomplish a change in the vehicle’s forward velocity (via a change in wheel rotation velocity).

A potentially major unknown for the controller is the coefficient of friction (cof) at the wheel/road interface at any given point in time. The value of the cof changes gradually as a function of tire wear, and more dramatically, depends on road surface conditions – such as water, oil, gravel, ice, etc. These latter changes can occur abruptly, and if such changes are outside the robustness region of the controller, the system would require fast on-line adaptation.

For the purposes of the present paper, the full problem definition is omitted (which would have included vehicle equations and description of the DHP Approximate Dynamic Programming (ADP) training procedures) (Lendaris & Schultz, 2000; Lendaris, Schultz,

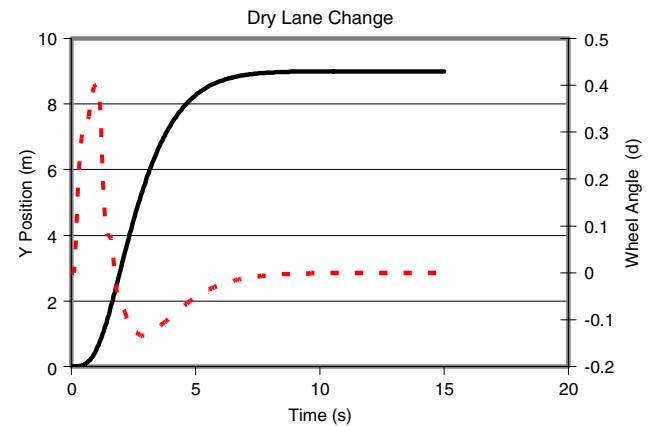
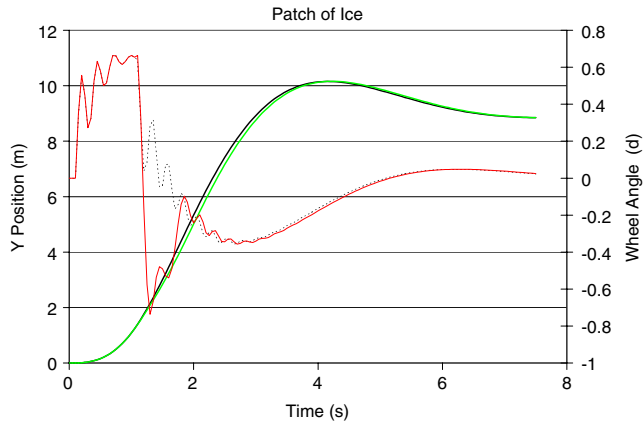


Fig. 7. Vehicle response to lane-change command at time 0, on dry pavement, with controller trained on dry pavement only.

& Shannon, 2000). The first step was to demonstrate that a lane-change controller could be designed via the ADP method, with a minimum-squared-error of lateral position (from one lane to the next) employed as the Utility function, and the method was indeed successful. The resulting control response is shown in Fig. 7. The solid line shows the lateral position of the vehicle as a function of time, where the starting lane is at 0 m, and the next lane’s center line is at 9 m. The dashed line shows the sequence of steering angles commanded by the controller. A (generalization) test was performed on this controller by introducing an ice patch on the road while the lane change was being executed. As expected, since the controller had no learning experience with such road conditions, it gave steering-angle instructions that resulted in the car going off the roadbed.

An ‘experienced’ human driver is one that has accumulated exposure to various tire/road conditions and has learned the vehicle’s response in each of them. In practice, when an experienced driver senses a vehicle response different from what was expected, the driver typically ‘tests’ the current tire/road interface capability via, say, a small dithering action on the steering wheel. The result of this test is then employed to modify the driver’s control action (select an appropriate control policy).

The above line of reasoning about an experienced human driver led us to define a proxy for the hypothesized driver’s ‘test’ (context discernment), and to input its corresponding measure(s) as a **Context Variable** to the neural network controller during the ADP design process. The proxy defined was a “sliding index” (SI) that incorporated successive values of acceleration generated by recent steering actions—motivated by the mentioned “dithering” test, and measured via assumed on-board accelerometers as a proxy for the driver’s seat-of-the-pants acceleration sensing, and was used as the CV input for this experiment. In addition, the Utility function was modified accordingly. A variety of values for the tire/road coefficient of friction were instantiated in the simulation equations during training, and the resulting values of the SI were calculated and used as the CV input values to the neural net. The resulting neural net controller functionally employs the proxy CV at its input as a “pointer” or “index” for selecting different control policies as the CV value changes – i.e., the neural network changes the mapping it performs according to the value of the CV at its input, cf. Fig. 6. The response to a change-lane command with an ice patch encountered part way through the maneuver is shown in Fig. 8. Note that the time axis is expanded to show just the first 8 s, and thus shows more details of the steering angles, including in this case, the distinctly different set of wheel angles commanded by the controller when the ice patch is encountered. Also, the scale of the vertical axis increased. Further, note the vehicle’s lateral position overshoots the 9 meter centerline of the next lane, but slowly comes back to it.



**Fig. 8.** Vehicle response to lane-change command at time 0, ice patch encountered at about 1.5 s, employing contextually-aware controller, with CV input representing different values of cof.

#### 4.2. Phase-4 Approach-1 applied to augmentation control for a hypersonic airframe (with Context Variable Input)

The example provided here is taken from a final report to NASA (Lendaris, 2003) that documents successful application of the DHP Adaptive Critic method to designing controllers for a hypersonic-shape aircraft known as LoFlyte<sup>®</sup>. The first task was to demonstrate the Approximate Dynamic Programming (ADP) method was capable of designing a control system that would augment the signals going from the pilot's commands to the aircraft inner control loop in a manner to make the LoFlyte<sup>®</sup> aircraft "feel" to the pilot as though it were an idealized version of the airplane, designated LoFlyte<sup>®\*</sup>. The report provides full details of the experiments and results. It suffices to say in the present paper that the ADP methodology performed extremely well – and pertinent to this paper, including demonstration of an experience-based component.

For nominal flight parameter values, the LoFlyte<sup>®</sup> aircraft is open-loop stable. The augmenting controller is designed to create a closed loop around LoFlyte<sup>®</sup> that modifies its flight characteristics to match those of LoFlyte<sup>®\*</sup>. The resulting system dynamics with the ADP-designed augmentation controller closely match those of LoFlyte<sup>®\*</sup>, hence are also stable, in principle at least. All test flight maneuvers performed on the augmented LoFlyte<sup>®</sup> demonstrated fully stable responses.

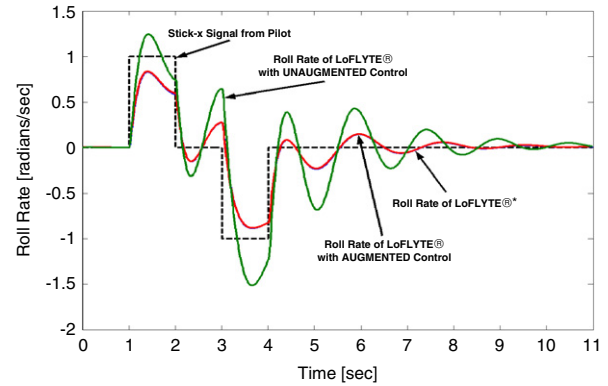
An example of tests run on the resulting control augmentation system is given via the following two figures. Three response curves in Fig. 9 are, respectively, the response of **Unaugmented** LoFlyte<sup>®</sup>, response of **Augmented** LoFlyte<sup>®</sup>, and the (idealized) desired response of LoFlyte<sup>®\*</sup>.

Fig. 9 shows roll-rate time histories for each of the three defined vehicle types in response to a Stick-X Doublet commanded by the pilot. Note that the histories for the Augmented LoFlyte<sup>®</sup> and the idealized LoFlyte<sup>®\*</sup> are virtually identical, whereas the Unaugmented LoFlyte<sup>®</sup> has significantly larger excursions.

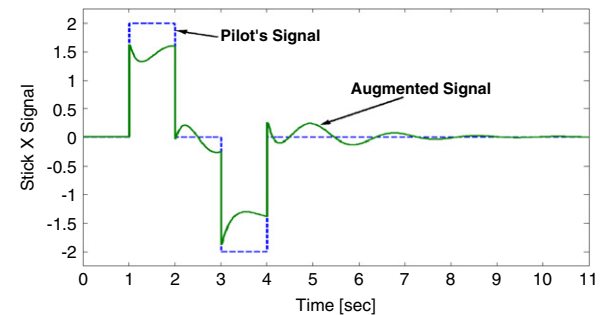
Fig. 10 shows a representation of the stick command given by the pilot during the same time interval of Fig. 9, overlaid by the augmented command signal actually given to the airplane's actuator control subsystem to effect the time histories shown in Fig. 9.

After the above task was successfully demonstrated, the more difficult task of accommodating a change of center of gravity of the aircraft in flight was undertaken (from the controller's perspective, a change in context occurs).

A reasoning similar to the one described for the car driver was applied relative to how a pilot responds to situations where the airplane doesn't "feel right" in its response to control inputs. In



**Fig. 9.** Response of LoFlyte<sup>®</sup>, Loflyte<sup>®\*</sup> and Augmented LoFlyte<sup>®</sup> to a Stick-X Doublet.



**Fig. 10.** Stick-X doublet: Pilot's stick command vs. augmented signal (the latter is sent to the aircraft actuators).

those cases, the pilot would also do some 'testing' – perhaps by inputting small commands via the stick and noting the response of the airplane, including "seat of the pants" sensing. In the case of a shift of center of gravity (c.g.) of the airplane, the pilot would determine its approximate shifted position via the tests performed, and "trim" the controls accordingly.

It turns out that for the LoFlyte<sup>®</sup> aircraft, the nominal c.g. is at about 50% of its length. At about 56% of its length (from front to back), LoFlyte<sup>®</sup> becomes open-loop unstable. For such a c.g. location value (and larger), the controller has the critical task of providing a stable closed-loop system. An ADP design process was set up that calculated (in a real-world implementable manner) an estimated location for the current c.g. and employed it as a **context variable** (CV) input to the neural network controller being trained. This entailed implementing calculations that emulate what the hypothesized pilot's test (context discernment) would do, again employing on-board accelerometer readings. During testing, the test flight maneuvers performed via the controller developed by this process demonstrated fully stable and acceptable responses for a variety of such conditions – extending out to over 58% of its length (it was trained only out to 57%, and thus represented a significant generalization capability). See Fig. 11 for the response to a shift of c.g. from 50% to the critical 56% position at about 4.5 s into the flight scenario. Note the Unaugmented LoFlyte<sup>®</sup> experiences a large nose-up response and gains some 250 ft. of altitude (and eventually goes unstable), whereas the Augmented LoFlyte<sup>®</sup> closely tracks the idealized LoFlyte<sup>®\*</sup>, whose c.g. is assumed not to have changed.

#### 4.3. Concluding comments for this section

The context discernment described in these two examples was (externally) provided to the NN controller via a Context Variable input, and the NN controller in essence contained its own local

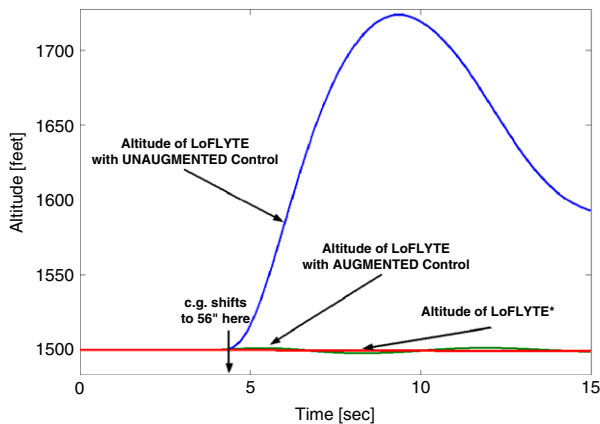


Fig. 11. Altitude during destabilizing c.g. shift employing contextually-aware augmenting controller (where CV is the estimated current location of c.g.).

library (repository) of controllers that are selectable via the context input. What distinguishes this approach from the more general approach defined for Phase-4 systems in general, however, is the fact that the context variables (and their respective sensors) were crafted by the control engineer. In general for Phase 4, the intent is to take the human out of that part of the design loop as well. 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 the higher level design (context discernment, etc.), and in particular, in a way that facilitates the controller selection process. Nevertheless, these examples do demonstrate that context can be employed as an input to the neural network during Reinforcement Learning and have the neural net learn to use the CV as a selector input.

## 5. Phase 4 – Approach 2: Employ HLLA methods to perform system identification (for context discernment)

We next describe experiments related to application of the HLLA ideas described in Section 3 to the task of context discernment for the Plant component of Context (cf. Fig. 1). This was defined in Section 3 as Experience-Based System Identification (EBSID).

### 5.1. EBSID example 1: 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 pole of a specified length and mass; the objective is to apply a sequence of control actions (discrete time is assumed here) to make the pole stand vertically following a disturbance in its angle (measured from the vertical, maximum angles are  $\pm 90^\circ$ ). 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 represented via an analytic model whose form is such that the length and mass of the pole are among the model’s parameters. This enables us to construct a plant manifold, (see (Lendaris, 2008) for description of the manifold concept in the EB applications), 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, where the context variables of interest are the pole length and mass. The result of this system identification process

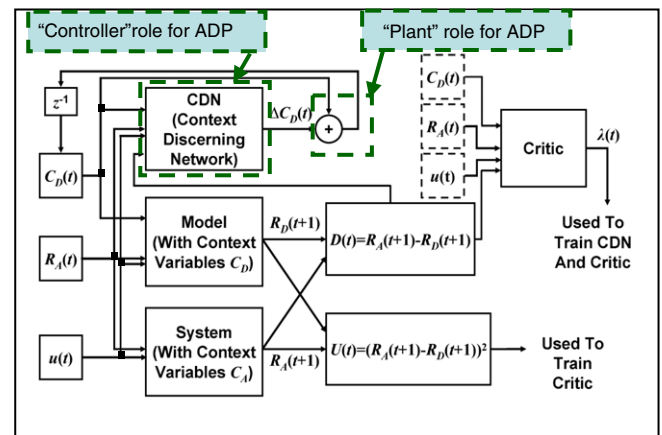


Fig. 12. Basic architecture for Context Discernment and for training CDN (after [13]).

could be used, for example, as the prerequisite step to controller selection.

We assign to the Higher Level Learning Algorithm (HLLA) the task of generating an EB-Algorithm for the EB System Identification (EBSID) box in Fig. 4a. 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. 12 contains the basic architecture for context discernment (system identification in this case), 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. 12, 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. 3. Most of those details are not repeated in Fig. 12. An outline of the experiments performed with this plant is described next; more details may be found in Holmstrom, Santiago, and Lendaris (2005). The notation of that paper is used in this description.

While the standard DHP method is employed for this training, the key to its use here is a mental shift of what fills the role of “plant” and what fills the role of “controller” in this “higher level” application. Referring to Fig. 12, it turns out the controller role is filled by the box labeled CDN, and the plant role is filled by the summing node that follows CDN (!). The latter took some getting used to, but this is the kind of mental shifting that is needed for the higher-level application of ADP suggested here. Continuing with Fig. 12, 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.

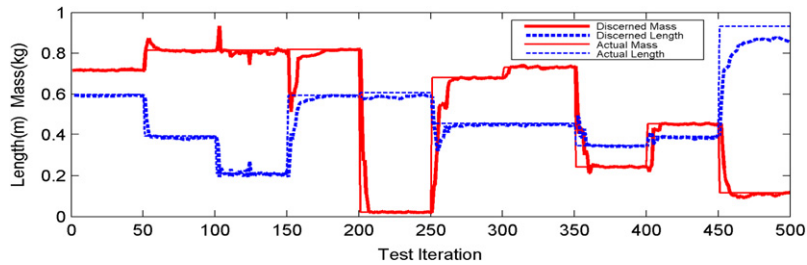


Fig. 13. Demonstration of Context Discernment in response to change in plant parameter values (context change) at every 50th iteration.

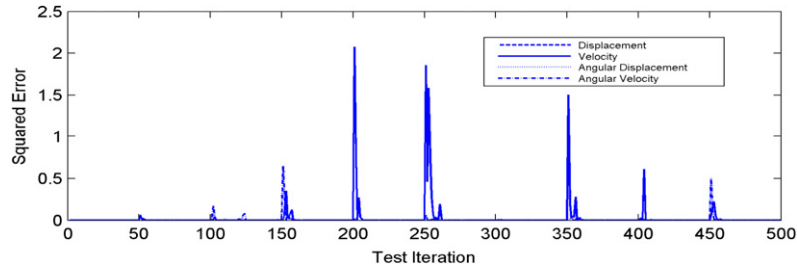


Fig. 14. Errors between state variable values for pole-cart system and for models selected during discernment process. Pole-cart parameters (context variables) change every 50th iteration [13].

Referring again to Fig. 12, 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. Due to space limitations, only sketchy details of the approach are included here; various chapters in Si, Barto, Powell, and Wunsch (2004) may be consulted for more details. As mentioned in the text near Fig. 3, 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

Fig. 3 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.

Details of the actual experiments are given in Holmstrom et al. (2005). The test results for the trained CDN are reproduced below in Figs. 13 and 14. 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. 13 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. 13 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 0.02 s, it follows the CDN is able to discern the mass and length of the pole in the pole-cart system in as little as 0.2 s. Fig. 14 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.

## 5.2. EBSID example 2: Neural network as plant

For the final example, consider an NN whose parameters (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, cf. (Lendaris, 2008). An entirely equivalent procedure may be used here as for the previous example. Thus, a configuration identical to that of Fig. 12 may be constructed, populating the contents of the box labeled “Model” with NNs of the type just described. Similarly, populate the



Fig. 15. Demonstration of Context Discernment in response to change in NN plant parameter values (context change) at every 100th iteration [39].

contents of the box labeled “System” with a specific one of these NNs. 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 (Santiago, 2005).

The process undertaken by the HLLA for this example is entirely analogous to that used in the previous example. Fig. 15 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 in the previous example, the context discerner uses 10 or fewer observations of the context to make the correct selection. The top graph of Fig. 15 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.

## 6. Future work

As noted in Lendaris (2008), there are many issues to still confront as the idealizations in the experience-based notions are relaxed, little by little. In the examples provided in Sections 4 and 5, the repositories were populated with models of the precise form of the plant being identified, and similarly for work related to controller selection. What will happen as the plant and controller models are close approximations but not exact? What will 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. (Lendaris, 1986), it will be useful to more deeply explore their application to the suggested Experience-Based Control notions.

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.

## 7. Summary

A recently proposed novel way of employing Approximate (or Adaptive) Dynamic Programming (ADP) is described and the author suggests the proposed experience-based approach portends a new, exciting phase in the development of the controls field. A key motivator for this approach is the human ability wherein the more knowledge/experience attained, the speed and efficiency of performing tasks are improved – in stark contrast to Artificial Intelligence systems thus far developed, wherein the more knowledge acquired (typically stored as “rules”) the *slower* the decision/action processing. The notions of context, context discernment, and experience are important to understanding such human abilities, so carefully defined versions of these are given apropos the identification and controls applications.

The key to implementing the approach is a shift in perspective when applying the ADP methodology. This refers to the approaches described both in Section 4 (conceptualizing “hand crafted” Context Variables to serve as index inputs to the NN controller) and in Section 5 (applying ADP “up a level” to develop on-line algorithms that efficiently and effectively *select designs* from a repository of existing controller solutions – called HLLA).

A previous paper (Lendaris, 2008) gives additional details about some of the theoretical aspects mentioned here. The present paper provides description of experiments wherein special Context Variables (CVs) were provided as inputs to a neural network controller in addition to the usual plant “state variables” during (regular) ADP type training (cf. Section 4) – yielding contextually-aware controllers. The NNs succeeded in learning to employ the CV inputs as essentially an “index” to *different functions* (memories) in the NN that map state variable inputs to the NN’s control outputs. The act of generating values for the CV inputs fills the Context Discernment step (related to that aspect of context that is being measured by the CVs) and the task of mapping from the context to the repository is performed within the  $N$ . In these cases, the more general theoretical aspects raised in Section 1.3 about dealing with appropriate mappings and indexing mechanisms from context to the repository do not need explicit addressing – the NN “takes care of it for you”. Much progress can be made along this path, while

parallel progress is made with the more theoretical considerations and application of the HLLA more generally.

As the author is soon to retire, a motivating objective for this paper (and its recent predecessor (Lendaris, 2008)) is to hopefully stimulate and inspire other researchers to pick up the mantle of the suggested next phase of the (intelligent) controls field. The approach requires a simple mental shift of the (human) control engineer, and it is this shift I desire to stimulate with the broad presentation approach adopted in this paper. The mathematics and more details are mostly available in cited publications.

### Acknowledgements

The author wishes to acknowledge the creative roles filled by the various graduate students in his Laboratory over the years. These include, but are 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, Adreas Rustan, Larry Schultz, Steve Shervais, Andrew Toland, and Sangsuree Vasupongayya. Particular acknowledgements go to my Lab Managers along the way: Thaddeus Shannon and Roberto Santiago. The pole-cart experiments in Section 5.1 were performed by Lars Holmstrom; the NN-as-plant experiments in Section 5.2 by Roberto Santiago.

In addition, the author acknowledges the many years of funding by the NSF, including ECS-9904378 and ECS-0301022.

### References

- Astrom, K. J., & Wittenmark, B. (1984). *Computer controlled systems, theory and design*. Englewood Cliffs, NJ: Prentice-Hall.
- Bertsekas, D. P., & Shreve, S. E. (1996). *Stochastic optimal control: The discrete-time case*. Belmont, MA: Athena Scientific.
- Dorf, R. C., & Bishop, R. H. (2001). *Modern control systems* (9th ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Goodwin, G. C., Graebe, S. F., & Salgado, M. E. (2000). *Control system design*. Prentice Hall.
- Holmstrom, L. A., Santiago, R. A., & Lendaris, G. G. (2005). On-line system identification using context discernment. In *Proc of the international joint conference on neural networks'05*. IEEE Press.
- Lendaris, G. G. (1986). On systemness and the problem solver: Tutorial comments. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(4), 604–610.
- Lendaris, G.G. (2003). DHP adaptive critic design of controller for LoFlyte<sup>®</sup>. *Final Report*. NASA P.O. L-17041 (also, for NAS1-01070; AAC Subcontract # 462), June 15.
- Lendaris, G. G. (2008). Higher level application of ADP: A next phase for the control field? *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 38(4).
- Lendaris, G. G. (2009). A retrospective on adaptive dynamic programming for control. In *Proceedings of international joint conference on neural networks*. IEEE Press.
- Lendaris, G.G., & Schultz, L.J. (2000). Controller design (from scratch) using approximate dynamic programming. In *Proceedings of IEEE international symposium on intelligent control '2000*.
- Lendaris, G. G., Schultz, L. J., & Shannon, T. T. (2000). Adaptive critic design for intelligent steering and speed control of a 2-axle vehicle. In *Proceedings of international joint conference on neural networks '2000*.
- Lendaris, G. G., & Neidhoefer, J. C. (2004). Guidance in the use of adaptive critics for control. In *Handbook of learning and approximate dynamic programming* (pp. 97–124). Piscataway, NJ: IEEE Press.
- Lewis, F. L. (1992). *Applied optimal control and estimation*. Englewood Cliffs, NJ: Prentice-Hall.
- Lewis, F. L., & Syrmos, V. L. (1995). *Optimal control* (2nd ed.). NJ: Wiley.
- Mosca, V. (1995). *Optimal, predictive, and adaptive control*. Englewood Cliffs, NJ: Prentice-Hall.
- Ogata, K. (2000). *Modern control engineering* (3rd ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Phillips, C. L., & Harbor, R. D. (2000). *Feedback control systems* (4th ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Santiago, R. A. (2005). *Dissertation proposal*, unpublished.
- Sastry, S., & Bodson, M. (1989). *Adaptive control: Stability, convergence, and robustness*. Englewood Cliffs, NJ: Prentice-Hall.
- Si, J., Barto, A. G., Powell, W. B., & Wunsch, D., II (2004). *Handbook of learning and approximate dynamic programming*. IEEE Press/Wiley-Interscience.
- Watt, J. (2005). *Microsoft Encarta Online Encyclopedia 2005*. 1997–2005 Microsoft Corporation. [Online]. Available: <http://encarta.msn.com>.
- Zhou, K., & Doyle, J. C. (1998). *Essentials of robust control*. Englewood Cliffs, NJ: Prentice-Hall.