ALERA: Accelerated Reinforcement Learning Driven Adaptation to Electro-Mechanical Degradation in Nonlinear Control Systems Using Encoded State Space Error Signatures

SUVADEEP BANERJEE, Intel Labs, USA
ABHIJIT CHATTERJEE, Georgia Institute of Technology, USA

The successful deployment of autonomous real-time systems is contingent on their ability to recover from performance degradation of sensors, actuators, and other electro-mechanical subsystems with low latency. In this article, we introduce ALERA, a novel framework for real-time control law adaptation in nonlinear control systems assisted by system state encodings that generate an error signal when the code properties are violated in the presence of failures. The fundamental contributions of this methodology are twofold—first, we show that the time-domain error signal contains perturbed system parameters' diagnostic information that can be used for quick control law adaptation to failure conditions and second, this quick adaptation is performed via reinforcement learning algorithms that relearn the control law of the perturbed system from a starting condition dictated by the diagnostic information, thus achieving significantly faster recovery. The fast (up to 80X faster than traditional reinforcement learning paradigms) performance recovery enabled by ALERA is demonstrated on an inverted pendulum balancing problem, a brake-by-wire system, and a self-balancing robot.

CCS Concepts: • **Computer systems organization → Maintainability and maintenance**; *Embedded and cyber-physical systems*; • **Computing methodologies →** *Markov decision processes*;

Additional Key Words and Phrases: Reinforcement learning, control systems, real-time systems, dependability

ACM Reference format:
Suvadeep Banerjee and Abhijit Chatterjee. 2019. ALERA: Accelerated Reinforcement Learning Driven Adaptation to Electro-Mechanical Degradation in Nonlinear Control Systems Using Encoded State Space Error Signatures. *ACM Trans. Intell. Syst. Technol.* 10, 4, Article 44 (July 2019), 25 pages.
https://doi.org/10.1145/3338123

1 INTRODUCTION

The successful deployment and assimilation of intelligent autonomous systems in human society depend on the trustworthiness and dependability [12, 13, 17, 18, 25, 30, 33] of these systems. For reliable operation, it is imperative that internal failures in sensors, actuators, and electro-mechanical subsystems of such autonomous systems be detected early and that control procedures for such systems be modified with low latency to adapt to these failures. In this context, reinforcement

We acknowledge support for this research from NSF Grant Nos: S&AS 1723997 and CCF:1421353.
Authors' addresses: S. Banerjee, Intel Labs, 3600 Juliette lane, Santa Clara, CA, 95054; email: sbanerjee49@gatech.edu; A. Chatterjee, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA, 30332; email: abhijit.chatterjee@ece.gatech.edu.

2157-6904/2019/07-ART44 $15.00
https://doi.org/10.1145/3338123

learning (RL) [15, 26] has been used in several nonlinear control problems [9, 27, 28, 31, 39, 45] and is a machine learning framework to optimize controller behavior by episodic interactions with the environment. The quality of the control action, known as the *policy*, is determined by the *reward* (specified by the user to achieve a particular goal) received by the controller. The objective of the RL controller is to apply optimal *policy* such that the cumulative future rewards, estimated by the *value function*, is maximized. In this work, we use the actor-critic (AC) network as the reinforcement learning algorithm. In the AC algorithm, the *critic* approximates the value function and a separate *actor* serves as the policy approximator. These actor-critic methods have been widely used as optimal controllers in nonlinear control problems [10, 29, 37, 41].

The typical AC algorithm starts learning without any model information and performs episodic experiments to explore the system state-space while gradually developing an optimal actor policy through trials. This leads to a long period of unpredictable and potentially damaging behavior. Hence, the learning speed of an AC algorithm needs to be significantly accelerated for practical use in nonlinear control systems. Prior works have explored mechanisms for accelerating the learning speed of AC algorithms. In [11] and [16], information from the process model of system dynamics is carefully assimilated in the AC algorithm for efficient learning. Hwang and Lo [19] utilized an average reward predictor between two learning episodes for improving the policy evolution. In [36], instead of using the slow incremental update of value function parameters, the authors employed batch algorithms for learning state value functions. A few works [46, 47] have also explored ideas from Probably Approximately Correct (PAC) learning framework to reuse previously discarded experiences intelligently to accelerate the value function learning. Recently, in [40], the authors have analyzed the rate of parameter convergence for RL algorithms in presence of unstable system dynamics and random exploration noise, thus showing the significant potential of accelerating the learning process.

Recently, reinforcement learning algorithms have also been explored as robust control mechanisms. In [42], a model-free policy iteration algorithm has been used to design a robust controller for nonlinear systems with model uncertainties. Wang et al. [43] designed an online RL-based fault-tolerant controller that can withstand unknown fault dynamics in discrete-time MIMO systems. However, the proposed scheme assumes all outputs and states to be measurable, which may not be true for all autonomous systems. The adaptive dynamic programming method has been used as a robust control scheme to handle actuator uncertainties in nonlinear multi-player systems in [20]. The three major drawbacks of these works are: (i) there is no explicit fault detection methodology and it is assumed that the compensatory algorithms are executed at all times, (ii) there is no discussion on latency of the compensatory action applied to mitigate the fault effects (the research on RL acceleration, discussed previously, proposes methods for faster convergence of learner parameters during episodic trials and are not designed for in-field learning acceleration under failures), and (iii) only numerical examples are provided in these research and no data on practical real-time systems have been provided.

To the best of our knowledge, no previous research has addressed the latency of using reinforcement learning algorithms as adaptive controllers to mitigate performance degradation due to parametric failures and component malfunctions in real-time systems. In this work, we introduce ALERA, an Accelerated Learning Enabled Reinforcement Architecture as a real-time self-learning framework for quick recovery of operational performance in presence of component failures. The use of nonlinear state-space encodings was first proposed in [4] as an error detection scheme. The nonlinear encoding produces an error signal whose magnitude is used to detect errors. In this article, for the first time, we demonstrate that the transient error signal waveform, in response to input stimuli, contains diagnostic information that can be used to significantly speed up the learning process. In [3], early ideas in encoding-assisted reinforcement learning framework were first

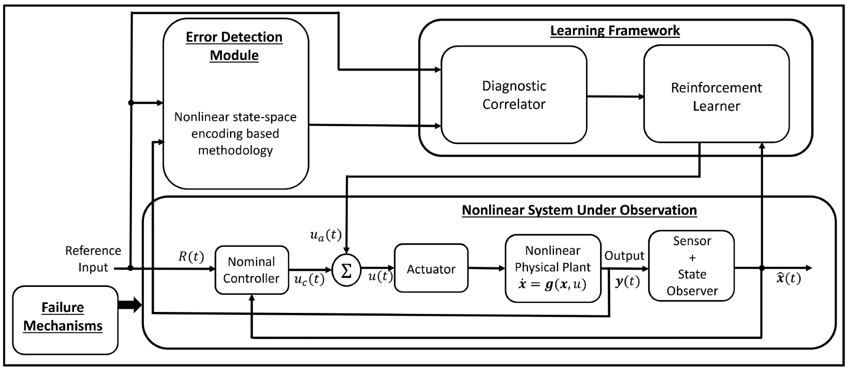

Fig. 1. Overview of the proposed methodology of ALERA.

developed and led up to the core methodology developed in this article. The core contributions of this research are:

(i) This research shows that the error signal generated through state-space encoding of the system contains diagnostic information that can be used to rapidly re-design the nonlinear control algorithm in real time using reinforcement learning to adapt to component failures.
(ii) Two methods of accelerating the reinforcement learning algorithm from the diagnostic information of the error signatures are demonstrated—one based on a parametric estimation approach followed by a signature clustering algorithm and another based on the use of a recurrent neural network directly operating on the time-series error signal.
(iii) The underlying technique ALERA is applied to real-world nonlinear control problems and adaptation results are shown using simulation data as well as fault-injection experiments conducted on a self-balancing robot (hardware).

The rest of the article is organized as follows: In Section 2, we provide an overview of the proposed architecture and describe the different components. The methodology of real-time self-learning enabled by ALERA is discussed in Section 3 along with the description of the fault models used in this research. Two different test cases of inverted pendulum balancing and an automotive brake-by-wire system are described in Sections 4 and 5 along with simulation results and analysis. Hardware validation of the proposed scheme is presented on a self-balancing robot in Section 6. Finally, we conclude in Section 7.

2 OVERVIEW OF PROPOSED ARCHITECTURE

The proposed scheme is illustrated in Figure 1. It consists of three primary components—(i) nonlinear system under observation, (ii) error detection module, and (iii) learning framework. The nonlinear system under observation consists of the plant, sensors, actuators, state observers, and the nominal controller. The error detection module detects any deviation of system performance from the expected behavior. The diagnostic correlator in the learning framework of Figure 1 extracts information from the time-domain error signal produced by the error detection module and uses that information to bootstrap the weights of the reinforcement learner in Figure 1. These weights are determined in a way (from time-domain analysis of the error signal) that allows the optimal control law for the modified nonlinear system under failure to be learned rapidly (10x–50x faster) using reinforcement learning algorithm than starting from fixed default weights for all failure modes of the system.

Hence, the key contributions of this research are: (i) demonstrating that the error signal from the error detection module contains diagnostic information about the faults (Section 2.5) and (ii) this crucial information can be extracted from the time-domain waveform of the error signal in the diagnostic correlator module for accelerating the reinforcement learning and hence mitigating the fault effects with low latency (Sections 2.5 and 3).

In the following sections, we describe each of the blocks of Figure 1.

2.1 Nonlinear System under Observation

The nonlinear physical plant with system dynamics of $\dot{\mathbf{x}}(t) = \mathbf{g}(\mathbf{x}(t), u(t))$ represents the physical system. The sensor measures the output $\mathbf{y}(t)$ and the state observer (nonlinear observer such as extended Kalman filter [22]) estimates the system states $\mathbf{x}(t)$ from the measurement $\mathbf{y}(t)$. The predicted state $\hat{\mathbf{x}}(t)$ is used along with the externally provided reference input $R(t)$ in the nominal controller to generate the required input $u_c(t) = h(\hat{\mathbf{x}}(t), R(t))$ that is applied to the plant through the actuators. The nominal controller $h(.)$ is designed to meet the system specifications under the assumption that all physical components of the system perform accurately. Under faults, the nominal controller is sub-optimal and cannot meet the system specifications and may even fail to achieve system functionality under critical errors. The learning framework provides the appropriate compensating action $u_a(t)$ with the assistance of the error detection module in such a way that $u(t) = u_c(t) + u_a(t)$ (see Figure 1). Under faults, $u_c(t)$ is a sub-optimal control and an appropriate $u_a(t) \neq 0$ is applied to restore system performance.

2.2 Failure Mechanisms

We assume that sensors, actuators and other electro-mechanical subsystems can malfunction over time and that failures are permanent (transient performance disruptions such as soft errors in the digital processor implementing the control algorithm are not considered in this research). Under failure, multiple sensor/actuator/electro-mechanical subsystem parameters can simultaneously deviate from their nominal values. We assume that error monitoring is performed at specific intervals of time and that performance degradation of system functions occurs either rapidly or gradually over time. The core methodology is also applicable to the case where error monitoring is performed on a continuous basis but performance degradation occurs rapidly from the onset of a failure. Once a failure is detected, we assume the nonlinear system is operated in a degraded but safe mode until system performance is restored using control law adaptation. *The goal of this research is to employ diagnosis-aided control law adaptation to minimize the time needed to recover system performance from the onset of failures.*

The mathematical model for temporal degradation of a sensor/actuator/electro-mechanical subsystem parameter p is expressed as:

$$\tilde{p}(t)|_{t>\tau_p} = p(t)|_{t<\tau_p} \times \alpha e^{\beta(t-\tau_p)} \tag{1}$$

where $t = \tau_p$ is the start of parametric degradation, $p(t)|_{t<\tau_p}$ is the nominal value of parameter p before $t = \tau_p$ (usually a fixed value), $\tilde{p}(t)|_{t>\tau_p}$ is the degraded parameter value and α and β denote the degree and rate of degradation. This parametric perturbation model alters the sensor/actuator parameters $p_1, p_2, \ldots, p_n$ into a modified set $\tilde{p_1}, \tilde{p_2}, \ldots, \tilde{p_n}$ such that the nonlinear dynamics of the entire system is changed.

Equation (1) represents the mathematical model of the synthetic failure mechanisms considered for fault injection purpose in this article and this model is not part of the core ALERA methodology shown in Figure 1.

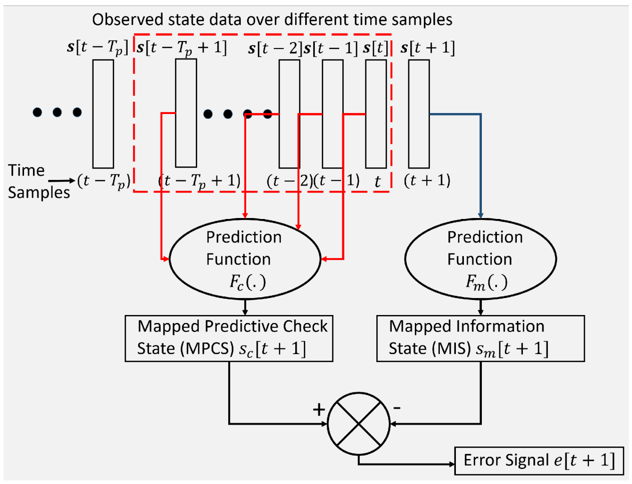
Observed state data over different time samples
$\mathbf{s}[t-T_p]$
$\mathbf{s}[t-T_p+1]$
$\mathbf{s}[t-2]$
$\mathbf{s}[t-1]$
$\mathbf{s}[t]$
$\mathbf{s}[t+1]$
Time Samples
$(t-T_p)$
$(t-T_p+1)$
$(t-2)$
$(t-1)$
t
$(t+1)$
Prediction Function $F_c(.)$
Prediction Function $F_m(.)$
Mapped Predictive Check State (MPCS) $s_c[t+1]$
Mapped Information State (MIS) $s_m[t+1]$
+
-
Error Signal $e[t+1]$

Fig. 2. Error detection is performed by nonlinear state-space encodings.

2.3 Error Detection Module

The error detection is accomplished by a state-encoding-based methodology described in [4] and illustrated in Figure 2. In this approach, a nonlinear prediction function generates a single encoded check state at each time instant, known as *mapped predictive check state* (MPCS) from the reference input and observed output over a fixed number of data samples. Mathematically, in the discrete domain, a unified system variable $\mathbf{s}[k] = [R[k], \mathbf{y}[k]]^\intercal$ is formed and the MPCS is predicted from T_p prior data samples $\mathbf{s}[k-1], \mathbf{s}[k-2], \ldots, \mathbf{s}[k-T_p]$ as

$$s_c[k] = F_c(\mathbf{s}[k], \mathbf{s}[k-1], \ldots, \mathbf{s}[k-T_p]), \tag{2}$$

where $F_c(.)$ indicates the nonlinear prediction function. Similarly, a different prediction function $F_m(.)$ generates an additional encoded state, known as *mapped information state* (MIS) from a separate unified variable $\mathbf{z}[k] = [\mathbf{y}[k], u_c[k-1]]^\intercal$ as

$$s_m[k] = F_m(\mathbf{z}[k]). \tag{3}$$

The prediction functions $F_c(.)$ and $F_m(.)$ are trained such that the error, defined as $e[k] = s_c[k] - s_m[k]$, is minimal with its magnitude lying below a certain threshold. It must be noted that this error signal $e[k]$ is different from the tracking error (in classical control theory) for the nominal controller that indicates the difference between the reference input and the system state that follows the reference input. The tracking error is used inside the nominal controller for generating the appropriate control signal. The error signal in the detection module is designed to detect any anomaly for which the prediction functions are not trained. In a faulty system, the magnitude of error signal $e[k]$ is greater than the pre-computed threshold, thus detecting presence of faults. Possible candidates for prediction functions include nonlinear regression such as multivariate adaptive regression splines (MARS), Volterra filters and neural networks that are described in [4] and [5] along with the tradeoffs between nonlinear mapping accuracy and implementation overhead. In this work, MARS is selected as the prediction function.

It has been demonstrated in both linear [32] and nonlinear [3] systems that in presence of faults, the transient waveform of the error signal contains diagnostic information and possesses strong correlation with the optimal control parameters to compensate such faults. This diagnostic

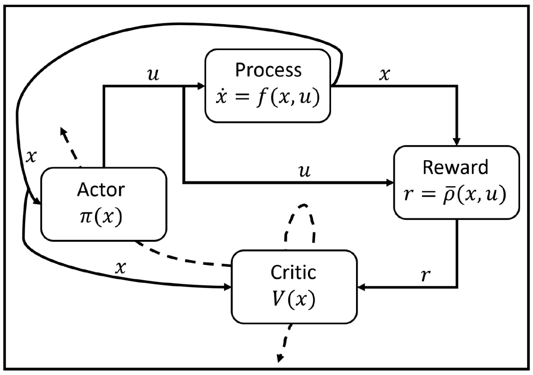

Fig. 3. Schematic overview of an actor-critic algorithm. The dashed line indicates the updating of critic and actor by the critic output.

capability of the error signal $e[k]$ is exploited in the current research for fast reconfiguration of control law in real-time.

2.4 Learning Framework

The learning framework contains a diagnostic correlator that bootstraps the weights of the reinforcement learning algorithm. Before describing this in detail, we describe the reinforcement learning (RL) framework of this research.

A deterministic RL framework is modeled by a Markov Decision Process (MDP) tuple $M(X, U, \bar{f}, \bar{\rho})$ where X is the state space of the control system, U is the space of control inputs, $\bar{f} : X \times U \mapsto X$ is the state transition function that describes the system dynamics and $\bar{\rho} : X \times U \mapsto \Re$ provides the reward function, assumed to be bounded. At each time step t, the controller takes an action $\mathbf{u}(t) \in U$ depending on the system state $\mathbf{x}(t) \in X$ from a control policy $\pi : X \mapsto U$. The objective of a RL algorithm is to find an optimal policy that maximizes the cumulative future rewards over an infinite horizon of time. The discounted sum of such rewards is known as *return* and defined as the continuous value function $V^\pi : X \mapsto \Re$ as $V^\pi(\mathbf{x}(t)) = \int_t^\infty e^{-\frac{s-t}{\tau}} \bar{\rho}(\mathbf{x}(s), \mathbf{u}(s))ds$ for any initial state $\mathbf{x}(t)$ and τ is the time constant for discounting future rewards.

Actor-critic methods [6, 7, 23, 24, 34], used in this research as RL algorithms, contain two distinct learning units—a *critic* that tries to learn the value function of the policy and an *actor* that provides the action according to a certain policy. A structure of an actor-critic algorithm is illustrated in Figure 3. The actor generates the policy action $\mathbf{u}$ depending on the current state $\mathbf{x}$. The critic receives the reward r for applying the current action and evaluates the quality of the present policy by updating the value function estimate. After a few policy evaluation steps by the critic, the actor is updated using gradient descent on the actor parameter space.

The actor-critic methods utilize the popular framework of temporal difference (TD) learning [38]. Any imperfect estimate of the value function generates the TD error,

$$\delta(t) \equiv r(t) - \frac{1}{\tau}V(t) + \dot{V}(t). \tag{4}$$

At each step of the learning algorithm, the value function estimate is adjusted to reduce the TD error $\delta(t)$. Thus, the magnitude of the TD error $\delta(t)$ provides a quantitative estimate of how different the current value function estimate is from the actual value function. The convergence of the value function and the policy to their optimal values is signified by the TD error approaching zero. Hence, the TD error is exploited for gradient descent update [38] of the critic parameters. The learning is further accelerated by the use of *eligibility traces* that introduce a notion of memory in the learning procedure. To define eligibility traces, the TD error in (4) is expressed in its discretized form as

$$\delta_t = r_t + \gamma V_t - V_{t-1}, \tag{5}$$

where $\gamma = 1 - \frac{\Delta t}{\tau} \simeq e^{-\frac{\Delta t}{\tau}}$ is the discretized discount factor with Δt as the sampling duration and the variable values are scaled as $V_t = \frac{1}{\Delta t} V(t)$. Eligibility trace is an additional memory variable $z_t(\mathbf{x})$ associated with each state $\mathbf{x}$ that evolves with time t. At each learning step, the eligibility traces for all states decay by $\gamma\lambda$ and the eligibility trace for the current state visited is incremented by 1 where $\lambda \in [0, 1]$ is the trace decay parameter. At any time, the eligibility traces record the past history of recently visited states and indicate the degree to which each state is eligible for undergoing learning changes. This evaluates the trajectory of system states instead of each particular state in response to the current policy. This is because the reward r_t is received as a result of a series of actions taken. Hence, eligibility traces are used to assign credit to states visited several steps earlier and provides multiple-step backup for distributing the reinforcement learning.

In practical implementations of RL algorithms, function approximators [38] such as linear coding, tile coding, radial basis functions, Kanerva coding, and the like are used to represent an actor and a critic due to the infinite size of continuous state and action spaces. In this work, normalized Gaussian networks are used as function approximators [14] for implementation of actor and critic. The value function is represented by the parameter vector $\mathbf{w}^c$ as

$$V(\mathbf{x}, \mathbf{w}^c) = \sum_{k=1}^{K} w_k^c b_k(\mathbf{x}), \tag{6}$$

where

$$b_k(\mathbf{x}) = \frac{a_k(\mathbf{x})}{\sum_{l=1}^{K} a_l(\mathbf{x})}, \quad a_k(\mathbf{x}) = e^{-||\mathbf{s}_k^{\mathsf{T}}(\mathbf{x}-\mathbf{c}_k)||^2}.$$

The vectors $\mathbf{c}_k$ and $\mathbf{s}_k$ define the center and the size of the kth Gaussian basis function. The number of basis functions K denote the level of discretization of the continuous state space and is a design choice. The discretization granularity is chosen based on the tradeoff between the available computational resources and the learning performance of the RL algorithm. The eligibility trace update equation [24] is

$$\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \nabla_{\mathbf{w}^c} V_{\mathbf{w}_t^c}(\mathbf{x}_t), \tag{7}$$

where $\mathbf{w}_t^c$ is the critic parameter at time t. Defining the TD error δ_t and eligibility trace $\mathbf{z}_t$ from (5) and (7), the update equation of the critic parameter is

$$\mathbf{w}_t^c = \mathbf{w}_{t-1}^c + \eta^c \delta_t \mathbf{z}_t \tag{8}$$

where $\eta^c > 0$ is the learning rate of the critic.

The actor policy is also implemented as a normalized Gaussian network based approximator $\pi(\mathbf{x}, \mathbf{w}^a)$ parametrized by $\mathbf{w}^a$ and given as

$$\begin{aligned} \mathbf{u}_t &= \mathbf{u}^{\max} s\Big(\pi(\mathbf{x}_t, \mathbf{w}_t^a) + \mathbf{n}_t\Big) \\ &= \mathbf{u}^{\max} s\left(\sum_k w_{k(t)}^a b_k(\mathbf{x}_t) + \mathbf{n}_t\right) \end{aligned} \tag{9}$$

where $\mathbf{u}^{\max}$ is the maximum action possible, $\mathbf{n}_t$ is a stochastic term that is initially used for exploration of the parametric space for evaluating new policies and gradually reduced to zero with

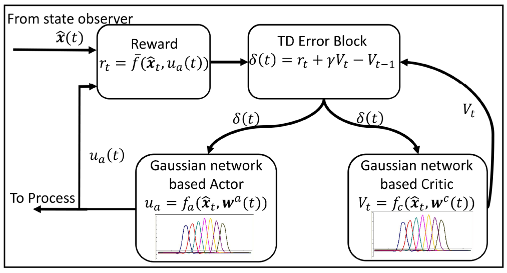
From state observer
$\hat{\mathbf{x}}(t)$
Reward
$r_t = \bar{f}(\hat{\mathbf{x}}_t, u_a(t))$
TD Error Block
$\delta(t) = r_t + \gamma V_t - V_{t-1}$
$\delta(t)$
$\delta(t)$
V_t
$u_a(t)$
To Process
Gaussian network based Actor
$u_a = f_a(\hat{\mathbf{x}}_t, \boldsymbol{w}^a(t))$
Gaussian network based Critic
$V_t = f_c(\hat{\mathbf{x}}_t, \boldsymbol{w}^c(t))$

Fig. 4. Schematic of the actor-critic based reinforcement learner module.

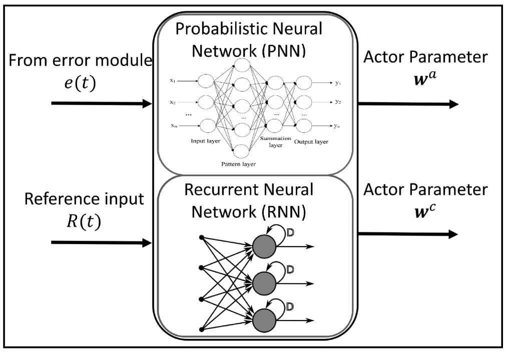

Fig. 5. Schematic of the diagnostic correlator module. It consists of either a PNN or a RNN depending on the pre-deployment training cost required.

progress of learning and $s(.)$ is a component-wise sigmoid function. The update equation of the actor parameter is

$$\mathbf{w}_t^a = \mathbf{w}_{t-1}^a + \eta^a \delta_t \nabla_{\mathbf{w}^a} \pi_{\mathbf{w}_t^a}(\mathbf{x}_t), \tag{10}$$

where $\eta^a > 0$ is the learning rate of the actor.

The above equations demonstrate that an actor-critic network is completely defined by its weight vectors $\mathbf{w}^c$ and $\mathbf{w}^a$. The proposed research in this work intends to reduce the exploration while finding optimum weight vectors significantly by initializing these weights to appropriate parameters, thus accelerating the self-learning process and enabling fast system adaptation under faults or unforeseen situations.

The actor-critic reinforcement learner is an augmentation to the nominal controller as shown in Figure 1. The nominal controller with output $u_c(t)$ is designed for the physical plant to satisfy certain system specifications. The estimated state $\hat{\mathbf{x}}(t)$ along with the reference input $R(t)$ is used to compute the reference tracking error $e_R(t)$ and the required control input $u_c(t)$ to reduce the tracking error. In a nominal fault-free system, the reinforcement learner is initially trained online with the nominal controller present in the control loop for improved closed-loop tracking performance [8]. The reward function for the reinforcement learner is derived from the estimated state $\hat{\mathbf{x}}(t)$ and the actor policy $u_a(t)$ such that the actor-critic algorithm assumes the nominal controller as part of the environment with which it reacts and learns the actor-critic weights appropriately. In a system compromised by faults, the TD error $\delta(t)$ of the trained nominal reinforcement learner becomes non-zero indicating exertion of sub-optimal control policy. The learning algorithm of the actor-critic network is triggered by the non-zero TD error to reconfigure the actor and critic weight vectors $\mathbf{w}^a$ and $\mathbf{w}^c$ such that the return is maximized in the altered environment. However, due to the episodic task based updates inherent in reinforcement learning, the optimal policy takes significant time to converge. The diagnostic correlator module is designed to reduce the latency of learning by bootstrapping the learning process. The schematic of the actor-critic reinforcement learner module is shown in Figure 4.

2.5 Diagnostic Correlator

The diagnostic correlator is shown in Figure 5 and takes as input, the error signal $e(t)$ produced by the error detection module and the reference input $R(t)$ to the nonlinear system (see Figure 1). It produces as outputs the weights $\mathbf{w}^a$ and $\mathbf{w}^c$ of the actor-critic networks, respectively, from which the AC reinforcement learning algorithm can rapidly relearn the optimal control law of the system under failure. The mapping from $e(t)$ and $R(t)$ to $\mathbf{w}^a$ and $\mathbf{w}^c$ is learned using supervised machine learning experiments described later in Section 3.1 and implemented via probabilistic neural networks (PNNs) or recurrent neural networks (RNNs) as shown in Figure 5 with appropriate tradeoffs discussed later in Section 4.2. A brief overview of PNNs and RNNs is given below.

(i) *PNN* - PNNs assign a class to an input data depending on the learned classification of class-labeled training vectors. The input layer contains N neurons where N is the size of the input vector. The number of neurons in the hidden layer is equal to the number of classes in the training dataset. Each of the neurons stores the input data from the training set along with the target class. Presented with a new test vector from the input layer, each hidden neuron computes the ℓ_2-norm of the test vector from the neuron's data and applies a radial basis function (RBF) to the distance to compute the weight of each training point with respect to the test data. The next layer consists of one pattern neuron for each target class. The weighted value from the hidden neuron is fed to the pattern neuron that corresponds to the neuron's category. All the class nodes add the values for the category they represent and finally the decision layer compares the weighted votes for each target class accumulated in the pattern layer and uses the largest vote to predict the target class for the input data.

(ii) *RNN* - The primary idea behind RNNs are to exploit sequential information while learning a function. Unlike traditional feed-forward neural networks, RNNs implement a recurrent connection with a tap delay associated with it. This allows the network to integrate "memory" in the learning procedure. An RNN is structurally similar to a feed-forward neural network with an additional feedback of the hidden layer output to the input of the hidden layer neurons. This enables the learning of nonlinear mapping from a temporal data sequence to a target vector associated with each sequence.

With this description of the different components of ALERA, we next focus on the proposed methodology of diagnosis-aided rapid performance recovery.

3 PROPOSED DIAGNOSIS-AIDED RAPID PERFORMANCE RECOVERY

The key idea of the proposed methodology is to construct a mapping between the error signal and starting conditions for the actor-critic algorithm in pre-deployment phase and use this mapping to determine optimal actor-critic weights in presence of failures in post-deployment phase.

3.1 Pre-deployment

The primary objective of the pre-deployment phase is to construct a mapping between the error signal and the optimal AC weights for proper control of a fault-injected system as shown in Figure 6. The different steps are:

(i) *Fault Sampling*. Based on the assumed failure models and parametric degradations of the system (plant, sensors, actuators, and mixed-signal interfaces), a finite fault universe is created. A Monte Carlo sampling of the system is performed over the parameter space of the assumed fault models, thus generating a number of fault parameter sets.

(ii) *Fault Injection.* The parametric perturbations are introduced in the nominal parameter values of the original set to create a collection of fault-injected systems.

(iii) *Error Signal.* Before fault injection, the nominal system with unperturbed parameter values is simulated with a set of expected reference inputs. The MARS regression functions in the error detection module are trained such that the fault-free error signal $e(t)$ is minimal and the resulting threshold is noted. Note from Section 2.3 that the error signal $e(t)$ is

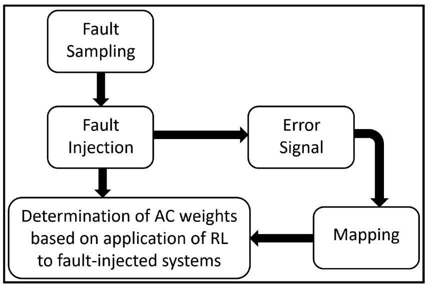
Fault Sampling
Fault Injection
Error Signal
Determination of AC weights based on application of RL to fault-injected systems
Mapping

Fig. 6. Schematic flow of the pre-deployment procedure to create mapping between error signal and optimal AC weights for proper control of a fault-injected system.

the difference between two encoded states MPCS (*mapped predictive check state*) and MIS (*mapped information state*) generated by two separate mapping functions F_c and F_m. In this research, the optimal training of function F_m is not explored and the MIS is created as a sum of the states representing linear mapping. Thus, F_c is trained to generate the MPCS from the prior T_p states to match the MIS at each time instant such that the instantaneous mismatch $e(t)$ is minimal. The error detection threshold is selected as the maximum mismatch between the MPCS and MIS during the training of F_c with the simulation data of nominal system.

The nominal controller is designed to meet system specifications of a fault-free system and the reinforcement learner is trained with different inputs $R(t)$ for achieving tighter reference tracking and ensuring better closed-loop system stability. During this phase, the AC unit starts from zero weights and performs episodic experiments on the system and both the critic and the actor learn the optimal value function V^* and policy π^* with convergence to nominal weight vectors $\mathbf{w}^c_{(nom)}$ and $\mathbf{w}^a_{(nom),}$ respectively. The completion of learning is indicated by the TD error $\delta(t)$ converging to zero as discussed in Section 2.4.

For the Monte Carlo set of fault-injected systems, the error signal $e(t)$ is recorded for a particular duration from the instant it exceeds the predetermined detection threshold (chosen as design parameter and traded off between diagnostic capability and latency of detection) along with storing the fault parameter set. Observation of the transient error signal over an elongated duration is helpful in diagnosing the probable cause of error with higher accuracy but delays the compensation procedure along with placing higher computational burden on the diagnostic correlator.

In fault-injected systems, the TD error $\delta(t)$ becomes non-zero from the instant when fault effect starts and the AC algorithm starts learning new optimal weight vectors such that $\delta(t) \rightarrow 0$. The TD error $\delta(t)$ is recorded for the same time duration for which the error signal $e(t)$ is recorded. The temporal waveform of $\delta(t)$ provides an indication of the learning progress.

(iv) *Mapping between Optimal AC Weights and Error Signal.* Two different mapping techniques are explored in this research: (a) PNN-based clustering and (b) RNN-based interpolation. Depending on the choice of mapping technique, the optimal AC learning for the fault-injected systems vary.

(a) PNN-Based Clustering. The non-zero TD error trajectory indicates the degree of sub-optimality in a fault-injected system and signifies how far the nominal weight vectors $\mathbf{w}^c_{(nom)}$ and $\mathbf{w}^a_{(nom)}$ are from the new optimal values for the faulty system. This notion is exploited in forming the different clusters.

First, a regression model is built with the error and reference input signal waveforms as inputs and the fault parameter set as output such that critical parameters can be predicted from the transient error signal. Then, the Monte Carlo samples are clustered into L classes in the fault parameter set based on the summed TD error magnitude during the learning of the nominal AC network over the chosen time duration. The PNN is trained with the class labels for each Monte Carlo sample. For each of the L clusters, a cluster medoid is selected to represent one particular fault injected system that best reflects the learning performance of the nominal AC unit for all the systems in that cluster. Unlike centroids, medoids are always members of the data set and do not represent a sample that may not be a practical parameter set. Next, for each of these L cluster medoids, an optimal AC network is learned such that the TD error $\delta(t)$ converges to zero, thus satisfying the design specifications in presence of injected faults. Each of the cluster medoids generates a different set of actor and critic parameter vectors $\mathbf{w}_l^a$ and $\mathbf{w}_l^c$ $\forall\, l = 1$ to L. The number of clusters L is an important metric in deciding the benefits achievable through the learning acceleration. With an increase in L, the number of fault injected systems with relearned optimal AC networks increases. Hence, the computational effort in training a new AC for each of the cluster medoids also increases along with the memory requirement for storing the AC parameters for each of those learned units. As the cluster count increases beyond a certain point, the achievable latency benefits by reinitializing the AC weight from stored pre-learned values are not sustainable due to the high memory requirements.

(b) *RNN-Based Interpolation*. The RNN-based scheme represents the other end of the spectrum of increasing cluster counts. In the RNN-based scheme, M maximally separated (ℓ_2-norm in fault parameter space) fault-injected systems are chosen from the Monte Carlo samples to represent the fault universe as best as possible. For each of these M systems, an optimal AC unit is trained to convergence generating M different actor and critic weights $\mathbf{w}_i^a$ and $\mathbf{w}_i^c$ $\forall\, i = 1$ to M. The RNN is trained with the time series data of the error signal $e(t)$ along with the input $R(t)$ for each of the M systems as input data and the respective actor and critic weights as output data. This enables the RNN to learn a nonlinear mapping from the temporal signal waveform of $e(t)$ and $R(t)$ such that actor and critic weights can directly be predicted by appropriate interpolation from an arbitrary error signal of a fault-injected system without the need for any intermediate parametric diagnosis and regression model.

3.2 Post-Deployment

The operational flow of the post-deployment phase is shown in Figure 7. In the presence of faults or anomalies at $t = t_0$, the MPCS based-error detection module generates an error signal $e(t)$ at $t = t_0$ above the predetermined threshold, indicating that an error or an unlearned event has occurred. In the PNN-based clustering scheme, the pre-trained regression model described above predicts the component parameters from the error signal and system input, recorded over a short time duration until $t = t_1$. During the time duration $t_0 \rightarrow t_1$, the nominal AC unit has started the reinforcement learning tasks to adapt to the new situation. The PNN categorizes the system behavior to one of the clusters in the parameter space and the AC network is initialized with the weight vectors corresponding to that cluster medoid at a later time $t = t_2$ and learning is resumed from the reinitialized weight vectors. In the RNN-based interpolation scheme, the RNN is invoked at $t = t_1$ directly with the error $e(t)$ and reference $R(t)$ as time series data input to compute the actor-critic weights. If the TD error $\delta(t)$ increases in magnitude after this reinitialization, it indicates that the AC unit has better adapted from the learning experience between t_0 and t_2 and the

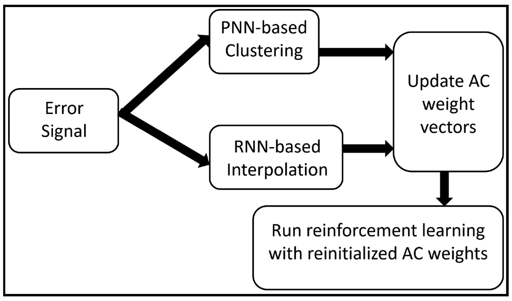

Fig. 7. Operational flow of post-deployment phase in the proposed methodology.

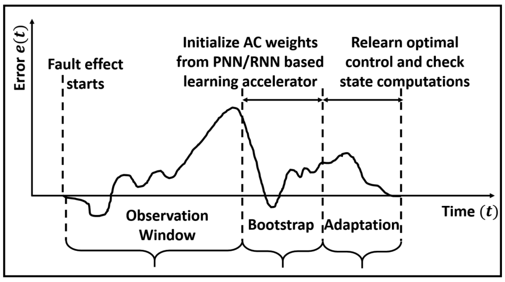

Fig. 8. Detection, diagnosis, and repair cycle.

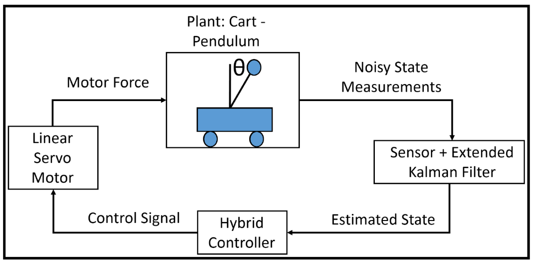

Fig. 9. Inverted pendulum system mounted on a cart.

reinitialization procedure deteriorates the learning process. In such a situation, the reinitialization is not adopted and the usual reinforcement learning is continued. However, if the TD error $\delta(t)$ magnitude reduces due to reinitialization, it indicates an accelerated boost to the learning process and significantly reduces time to converge. The temporal steps of error detection and control compensation are shown in Figure 8.

4 TEST CASE I: INVERTED PENDULUM SYSTEM

We describe the test case of an inverted pendulum balancing and demonstrate how ALERA accelerates the performance recovery along with discussion of different tradeoffs.

4.1 System Description

Figure 9 shows the first test case of balancing an inverted pendulum on a moving cart. The control objective is to optimally move the cart in a manner such that the pendulum is balanced vertically upright. An extended Kalman filter (EKF) is used to predict unobserved states from noisy measurements. The actuation for the cart movement is provided by a linear servo motor [2]. A hybrid (with linear and nonlinear components) energy-based controller is used as the nominal controller along with an AC unit as discussed later. The system dynamics $\mathbf{g}(.)$ is expressed as:

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \dot{\theta}\\ x\\ \dot{x}\end{bmatrix} = \begin{bmatrix}\dot{\theta}\\ g_{\theta 1}(\theta,\dot{\theta},u)\\ \dot{x}\\ g_{\theta 2}(\theta,\dot{\theta},u)\end{bmatrix} \tag{11}$$

where

$$g_{\theta 1}(\theta,\dot{\theta},u) = \frac{u\cos(\theta) - (M+m)g\sin(\theta) + ml\cos(\theta)\sin(\theta)\dot{\theta}^2}{ml\cos^2(\theta) - (M+m)l}$$

$$g_{\theta 2}(\theta,\dot{\theta},u) = \frac{u + ml\sin(\theta)\dot{\theta}^2 - mg\cos(\theta)\sin(\theta)}{M + m - m\cos^2(\theta)}$$

and M is the mass of the cart, m is the mass of the pendulum's bob, θ is the angle between the pendulum and the vertical axis pointing upward, x is the position of the cart, l is the length of the pendulum (assumed mass-less), g is the acceleration due to gravity and u is the motor force applied to the cart to control its motion. The energy-based hybrid controller in [44] is selected as the nominal controller. The nonlinear control action of the energy-based controller is expressed as

$$u_c = (M + m\sin^2(\theta))\{g_1(x_d - x) - g_2\dot{x}\} + mg\cos(\theta)\sin(\theta) + ml\dot{\theta}^2\sin(\theta), \tag{12}$$

where the term x_d represents a sinusoidal reference input constructed from $(\theta,\dot{\theta})$ at each time instant and is designed to serve as the reference input for gradual increase of energy of the cart-balance system. g_1 and g_2 represent parameters of a second order transfer function from the reference input x_d to the actual cart position x. As the pendulum angle θ approaches the upright position, the control is switched over to a linear negative state feedback controller for fine balancing after $|\theta| \le \theta_{th}$, where θ_{th} is the switching angle from one controller to the other. The linear control law is of the form

$$u_c = -\mathbf{K}.\mathbf{x}, \tag{13}$$

where $\mathbf{K}$ is the feedback gain matrix and $\mathbf{x}$ is the system state vector. For measurements, only the angular position θ of the pendulum and the cart displacement x are observed in presence of measurement noise and modeled as,

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \tag{14}$$

where $\eta = [\eta_1 \quad \eta_2]^\intercal$ represent two Gaussian distributed measurement noise components.

4.2 Simulation Results

(i) *Pre-Deployment Training.* The simulation experiments are conducted in Matlab on a Windows PC with a 3.10 GHz processor. The pendulum-cart system is simulated with parameter values of $M = 3$ kg, $m = 0.2$ kg, $l = 0.31$ m, and $g = 9.81$ m/s^2. The hybrid controller is designed to switch from the nonlinear controller to the state feedback controller for $|\theta| \le 30^\circ$ where $\theta = 0^\circ$ is the vertically upright position. The linear feedback controller is designed to achieve 20% maximum overshoot and a 2% settling time of 2 seconds. The appropriate feedback gain K in (13) is determined using Ackermann's formula for pole placement. The measurement noise is modeled by a Gaussian distribution with mean 0 and standard deviation of 0.1. The values of the different parameters for the normalized Gaussian network based actor critic method described in Section 2.4 are shown in Table 1. Exploration is done every fifth step instead of every step [16] to allow for large exploratory actions while providing sufficient time to the AC controller to learn from the exploratory actions. Each episode of AC learning trial is 20 seconds long and the balancing is initiated with the pendulum hanging vertically downwards. If a learning episode is terminated due to

Table 1. Actor-Critic Learning Parameters

Parameter	Value
$\lvert\theta\rvert_{max}$	π rad/s
$\lvert\dot{\theta}\rvert_{max}$	2.5π rad/s
$\lvert x\rvert_{max}$	2.4m
$\lvert\dot{x}\rvert_{max}$	2m/s
Actor grid size	$10 \times 10 \times 10 \times 10$
Size of $\mathbf{w}^a$	10,000
Critic grid size	$10 \times 10 \times 3 \times 3$
Size of $\mathbf{w}^c$	900
γ	0.5
λ	0.5
η^c	0.8
η^a	0.8
Sigmoid $s(x)$	$2\frac{\tan^{-1}(\frac{\pi x}{2})}{\pi}$
$\mathbf{n}(t)$	$\sim \mathcal{N}[0, 2]$
Reward $\bar{\rho}(\mathbf{x})$	$\cos\theta$

violation of state restrictions caused by the exploration of actor policy, the trial is rewarded with $r = -1$ to penalize the corresponding action taken.

After learning the nominal AC controller, the system is simulated with 100 different initial angles uniformly sampled between $\theta = 180°$ and $\theta = 10°$ over 20 seconds of simulation. The sliding temporal window length is selected as 12 samples in order to accurately map the nonlinear dynamics. The unified variable $\mathbf{z}$ is selected as $\mathbf{z} = [y_1 \quad y_2 \quad u_c]$ where y_1 and y_2 are the two measurements in (14). The function $F_m(.)$ is selected as a linear function $F_m(\mathbf{z}) = \frac{y_1}{y_{1(max)}} + \frac{y_2}{y_{2(max)}} + \frac{u_c}{u_{c(max)}}$ where $y_{1(max)}$, $y_{2(max)}$, and $u_{c(max)}$ denote the absolute maximum values of the respective variables obtained from the 100 different simulations. Thus, a MARS model with memory depth 12 is trained for implementing $F_c(.)$ such that $F_m(\mathbf{z})$ is predicted from the past 12 observations. The trained MARS model is evaluated on test data of 30 different simulation experiments with initial angles uniformly sampled between $\theta = 180°$ and $\theta = 10°$, with 15 systems as fault-free and 15 systems as faulty. The injected faults are perturbations of the torque constant and the back-emf constant of the linear motor by a uniform distribution of 35%. The nominal detection threshold of $|e_{nom}| = 0.05$ is established from a target F_1-score of 0.9.

For pre-deployment training, the torque constant and the back-emf constant of the linear motor are perturbed by a uniform distribution of 35% to create the systems with injected actuator faults. The sensor data in (14) is corrupted with a bias selected from a uniform distribution of 10% to generate systems with sensor faults. 4,000 Monte Carlo samples are created by sampling the fault universe of sensor and actuator faults. For all these systems, the error signal $e(t)$ is recorded along with the reference input $x_d(t)$ as defined in (12) for $\Delta t = 2$ seconds. The TD error $\delta(t)$ is also recorded for the same duration and the magnitudes are summed up to generate a single value as an evaluation of the learning performance of the nominal AC controller for each fault-injected system. For the PNN-based scheme, a nonlinear regression model (MARS) is trained with the error signal $e(t)$ and reference input $x_d(t)$ to predict the perturbed parameter set - torque constant, back emf constant and the sensor output biases. Next, a PNN classifier is used to cluster the 4,000 parametric samples on the basis of the TD error magnitude generated previously into three clusters.

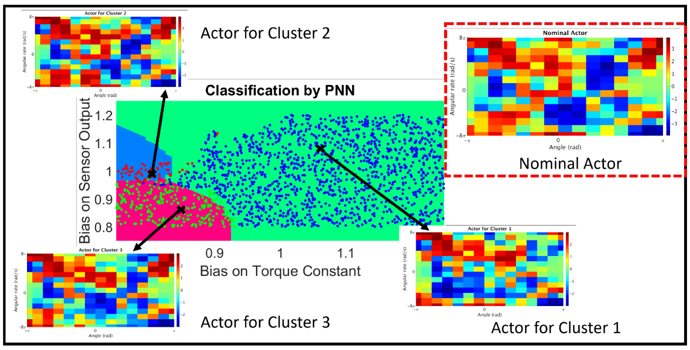

Fig. 10. Classification in the parameter space by the PNN and subsequent generation of representative optimal AC controllers for each of the cluster centers.

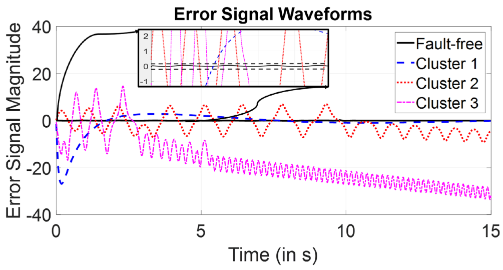

Fig. 11. Error signal waveforms for the three different clusters along with the fault-free error.

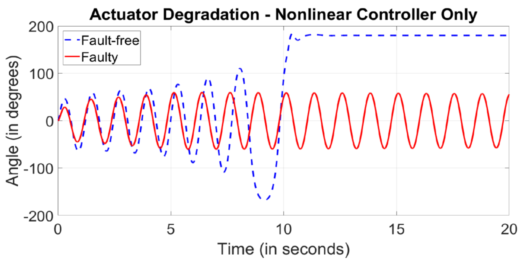

Fig. 12. This shows the need of a self-learning assisted controller design where a nominally designed controller is unable to meet system functionality under actuator degradation.

For each of the clusters, one medoid sample is chosen to represent all other points in that cluster. For each of the medoid systems, a new optimal AC controller is trained by performing episodic experiments until the TD error $\delta(t)$ converges to zero, thus relearning how to balance the pendulum in presence of sensor and actuator errors. The weights of the three optimal AC controllers for each cluster center are stored in memory. For the RNN-based scheme, 20 maximally separated samples are selected from the 4,000 Monte Carlo samples such that the 4-dimensional fault space is reasonably sampled. For each of these 20 fault-injected systems, an optimal AC controller is learned and an RNN is trained with the time series data of $e(t)$ and $x_d(t)$ to directly predict the AC weight parameters. The clustering and the different AC controllers for the PNN-based scheme are shown in Figure 10.

The subfigure in the top right inset of Figure 10 shows the nominal actor of the AC controller trained for the fault-free system. The different colors demonstrate the varying control force u_a between [−3,+3] imparted in addition to the nominal control u_c at different state space locations. The original state space is 4-dimensional and for visual demonstration, this figure is a projection of actor policy with changing θ and $\dot{\theta}$ on $x - \dot{x}$ plane. The classification plot in Figure 10 is also a projection from the 4-dimensional fault parameter space to a 2-dimensional subspace where the x and y axes show the bias on torque constant and bias on the sensor output y_1. The three optimal actor-critic controllers depicted are relearned on the fault-injected systems represented by the cluster medoids and they are different from the nominal one as seen in Figure 10. The corresponding error signals for the three cluster medoids are illustrated in Figure 11 along with

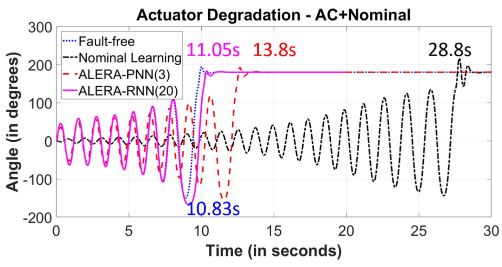

Fig. 13. Demonstration of learning acceleration of ALERA in presence of actuator degradations.

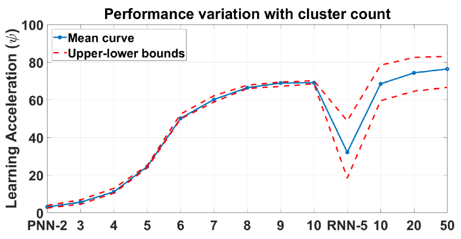

Fig. 14. Comparison of learning acceleration ψ between PNN and RNN schemes with PNN variable as number of clusters and RNN variable as number of training samples.

the fault-free error signal. The pre-determined threshold is shown in the magnified image in the inset of Figure 11 as dashed lines.

(ii) *Post-Deployment Experiments.* Figure 12 shows the system behavior in absence of any reinforcement learning controller. Only the hybrid controller is used and the augmentation $u_a(t)$ from the AC controller is disabled. In the fault-free system, the nominal controller is able to balance the pendulum vertically at $t = 10.83$ seconds. An actuator fault of 20% degradation in the torque constant of the motor is gradually injected from $t = 0$ and the maximum degradation of 20% is complete at $t = 4$ seconds according to Equation (1) (the fault is inserted at a high rate for illustrative purposes). It is seen that the hybrid controller fails to balance the pendulum in presence of the actuator fault and a self-learning module is necessary to supplement the nominal controller for providing correction capabilities.

Figure 13 demonstrates the learning acceleration enabled by ALERA-PNN and ALERA-RNN in comparison to the nominal learning of the augmented controller. The same 20% actuator degradation is injected as mentioned above. In the fault-free case, the pendulum is vertically balanced at $t = 10.83$ seconds. In presence of the actuator degradation, the pendulum is vertically balanced at $t = 28.8$ seconds without the assistance of the diagnostic correlator. With the PNN-based diagnostic correlator enabled with three clusters used for the PNN training, the pendulum gets balanced at $t = 13.8$ seconds, thus accomplishing the desired functionality faster than the nominal learning. However, the RNN-based learning scheme balances the pendulum at $t = 11.05$s providing significantly low latency in adapting to the altered circumstances.

Defining the additional time to achieve the same application level functionality through nominal and assisted schemes as t_n and t_a respectively, the "learning acceleration" ψ is defined as $\psi = \frac{t_n}{t_a}$. Thus, the acceleration ψ for ALERA-PNN is $\psi = \frac{28.8-10.83}{13.8-10.83} = \frac{17.97}{2.97} = 6.05$. Similarly, the learning acceleration ψ for ALERA-RNN is given as $\psi = \frac{28.8-10.83}{11.05-10.83} = \frac{17.97}{0.22} = 81.68$. Thus, the learning acceleration of ALERA-PNN and ALERA-RNN provides a 6.05X and 81.68X boost for application level compensation, respectively.

(iii) *Analysis and Discussion.* The learning acceleration ψ depends on how close the optimal actor-critic weights proposed by the correlator module are to the actual optimal values achieved after learning convergence. As discussed in Section 3.1, the cluster medoid is a representation for all systems in that cluster for which the learning behavior of the nominal AC is similar. With the increase in the number of clusters, the disparity between a system close to any cluster boundary and the medoid decreases and the potential for learning acceleration increases. The performance variation with the increase in cluster count is depicted in Figure 14 along with the upper and lower bounds. Five thousand (5,000) different fault-injected systems are created by exhaustive sampling of the

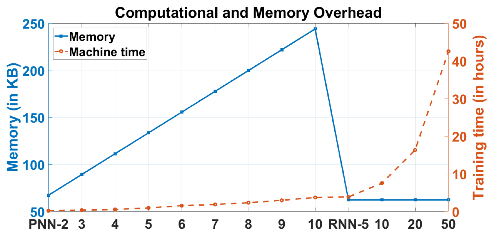

Fig. 15. Memory overhead and machine time required for pre-deployment training in the PNN- and RNN-based schemes.

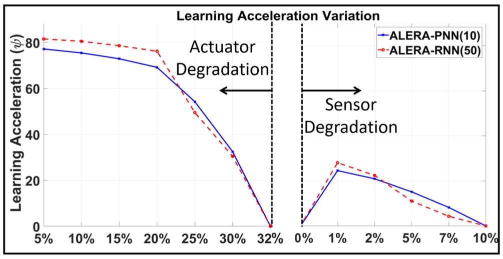

Fig. 16. Variation of learning acceleration ψ for ALERA-PNN and ALERA-RNN with different degrees of sensor and actuator degradation.

4-dimensional fault space described previously. For each of those systems, the ALERA-enabled learning acceleration is applied with the PNN trained with different number of clusters in pre-deployment phase. The mean acceleration ψ achieved along with the maximum and minimum for each cluster count is plotted in Figure 14. It is seen that ψ increases from 3X to 70X with increase in the cluster count from 2 to 10. However, the trend shown in Figure 14 illustrates that the rate of increase of ψ is initially high and it starts tapering off after cluster count of 6, showing diminishing benefits with increasing clusters. After 10 clusters, the acceleration of the RNN-scheme is presented. The RNN interpolates the optimal AC weights depending on the training received. An RNN trained with 5 data samples provides a mean ψ of 32X and the acceleration grows up to 80X for training with 50 data samples. However, due to the interpolation by an RNN trained on a few training cases, the performance varies widely with the maximum and minimum bounds gradually reducing with the increasing number of training data samples.

The memory overhead required along with the machine time consumed for pre-deployment training is shown in Figure 15. It is seen that the memory overhead increases linearly with the number of clusters for the PNN-based scheme. The depicted memory overhead indicates the space required to store the learned AC parameters corresponding to each cluster center along with the trained PNN classifier and the regression model. The RNN-based scheme requires significantly low memory since only the trained RNN is stored without separately storing any individual AC parameters. The memory requirement of the RNN is fixed with the number of training data as only the trained network is stored. However, the pre-deployment simulation time necessary to train the RNN is quite large in comparison to the PNN-based scheme. For example, the RNN scheme with only 5 training data samples takes 3.88 hours of machine time in comparison with 3.75 hours in the PNN scheme with 10 clusters. This significant machine time requirement is due to the primary reason that the RNN is trained to directly predict the AC parameters (10,900 output data points) from the time-series data of the error signal $e(t)$ and reference input (4,000 input data points), thus leading to significantly slower training than PNN. It is seen in Figure 15 that the machine time requirement for the RNN-scheme increases exponentially with the increase in number of training data. Thus, for applications where pre-deployment simulation is expensive, an RNN-based scheme may not be feasible despite providing better benefits than a PNN-based clustering scheme.

Figure 16 shows the variation of learning acceleration ψ for the PNN and RNN schemes in presence of actuator and sensor degradations. The x-axis shows the degree of bias for sensor degradation and the amount of parametric perturbation in motor torque constant for actuator degradation. It is seen that sensor faults are more difficult to correct than actuator errors. The data shown is for the ALERA-PNN with 10 clusters and ALERA-RNN with 50 data samples for training. The left portion of Figure 16 represents actuator errors ranging between 5%–32% while the right

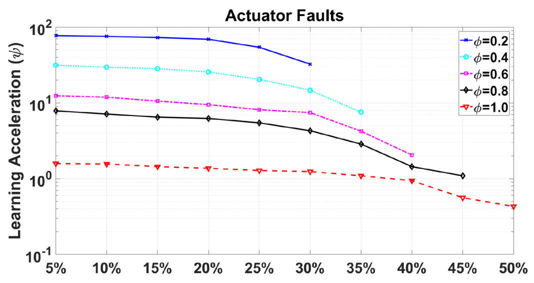

Fig. 17. Logarithmic variation of learning acceleration ψ with ϕ under different magnitude of injected actuator errors.

portion denotes sensor degradations with errors ranging between 1%–7%. It is seen that as the magnitude of injected faults increases, the learning boost reduces for both sensor and actuator errors. ψ is shown to be zero at actuator and sensor degradations of 32% and 10%, respectively, to indicate that the AC controller fails to balance the pendulum and functionality is never achieved with fault magnitudes equal to or greater than that. This is explained from the fact that the AC controller is used to augment the nominal controller by providing a supplementary action u_a to the nominal control u_c. However, the combined action $u = u_c + u_a$ has to satisfy the physical limitation of applied control u, defined by u_{max}. This is accomplished by choosing individual maximum restrictions $u_{c(max)}$ and $u_{a(max)}$ for the nominal and AC controller in the design phase. The *reconfiguration ratio* $\phi = \frac{u_{a(max)}}{u_{max}}$ determines the extent of impact that the AC controller exercises during the reconfiguration. $\phi = 0$ indicates that the nominal controller is solely responsible for providing adequate control with $u_a = 0$, whereas $\phi = 1$ represents that the AC controller is completely responsible with $u_c = 0$. A significant downside of choosing this approach is that it leads to a sub-optimal design because both the nominal and AC controller typically do not generate the maximum control simultaneously.

Figure 17 demonstrates the logarithmic variation of learning acceleration ψ with actuator error magnitude under five different values of ϕ - 0.2, 0.4, 0.6, 0.8, and 1. The key observations from this plot are: (i) As ϕ increases from 0.2 to 1.0, the contribution of the RL controller to the total applied control increases. This enhances the extent of self-learning capabilities of the RL controller since it forms a higher portion of the applied control. Hence, the AC controller is able to tolerate and correct higher magnitude of injected actuator errors as seen from the logarithmic plot of Figure 17. With $\phi = 0.2$, the maximum actuator error that can be corrected with restoration of system functionality is 32%, whereas with $\phi = 1.0$, the maximum actuator error that can be corrected is 51%. (ii) With the increase of ϕ, the learning acceleration ψ decreases. For example, with the same 5% actuator error, the learning boost is ≈80X with $\phi = 0.2$ whereas, with $\phi = 0.8$, the learning acceleration is only 8X. This is because, with increase of ϕ, the AC controller has to contribute more to the applied action and has to perform significant learning and takes longer to converge, thus effectively reducing the learning acceleration.

This analysis demonstrates that there is a tradeoff in the choice of the reconfiguration ratio ϕ. A lower value of ϕ provides faster performance recovery but corrects less errors, whereas a higher value of ϕ recovers from more severe errors but the adaptation is slow.

5 TEST CASE II: BRAKE-BY-WIRE SYSTEM

We describe an automotive brake-by-wire (BBW) system and demonstrate how ALERA restores system functionality in a braking event with a degraded brake pad with significantly low latency.

5.1 System Description

In a typical BBW system, electromechanical actuators are integrated with electronic controllers and communicating sensor interfaces. A supervisory brake-torque controller, implemented on a micro-controller, receives the desired braking input as displacement of pedal sensor and computes the required vehicle deceleration. Speed data are gathered from wheel-mounted sensors and the appropriate braking torque is generated by the designed control algorithm. The braking is applied through electro-magnetic brakes known as eddy-current brakes (ECBs). In this work, the BBW dynamics described in [21] are adopted. The longitudinal velocity and the rotational dynamics are

$$\dot{v}_x = -\frac{1}{M}\sum_{i=1}^{4}\mu_i(\kappa_i)F_{zi} \tag{15}$$

$$\dot{\omega}_i = \frac{1}{I_{wi}}(-T_{bi} + \mu_i(\kappa_i)F_{zi}R), \tag{16}$$

where v_x is vehicle longitudinal speed, M is total vehicle mass, ω_i is angular speed of ith wheel, I_{wi} is rotational inertia of ith wheel, R is the effective wheel rolling radius, F_{zi} is normal frictional force at the interface of ith wheel and road surface and T_{bi} is the generated brake torque at ith wheel. The term $\mu_i(\kappa)$ represents the friction coefficient as a function of time-varying wheel slip ratio $\kappa_i(t)$ defined as

$$\kappa_i(t) = \frac{v_x - R\omega_i}{v_x} \tag{17}$$

The nonlinearity in the function $\mu(\kappa)$ depends on road surfaces (asphalt, snow, ice, etc.) and the friction coefficient varies significantly with the slip-ratio κ. Equations (15–17) represent the plant equations of the BBW test case.

The nominal controller design is adopted from the anti-lock braking system (ABS) described in [1]. The primary control objective is to modulate the applied braking torque T_b such that wheel lock-up conditions can be avoided while maintaining the critical slip-ratio. In a wheel lock-up situation, one or more wheels start having higher rate of angular deceleration than the equivalent linear deceleration of the vehicle, causing loss of traction, slippage of wheel over road surface and eventual loss of vehicle control. The sliding mode nonlinear control law for ABS is

$$T_{bi} = R\alpha_{si}\kappa_i F_{zi} + \frac{I_{wi}}{v_x}\frac{\omega_i}{M}\sum_{i=1}^{4}\alpha_{si}\kappa_i F_{zi} + \eta\frac{I_{wi}}{R}v_x * \mathrm{SAT}\left(\frac{\kappa_{th}-\kappa_i}{\nu}\right),$$

where the saturation function $\mathrm{SAT}(\frac{x}{y})$ is defined as

$$\mathrm{SAT}\left(\frac{x}{y}\right) = \begin{cases} \frac{x}{y} & \text{if } |\frac{x}{y}| \le 1 \\ \mathrm{sgn}(\frac{x}{y}) & \text{if } |\frac{x}{y}| > 1. \end{cases} \tag{18}$$

The terms α_{si}, κ_{th}, η and ν are nonlinear parameters of the sliding mode controller. The wheel-slip dynamics along with the ECB implementation is described in details in [1]. The sensors measure the 4 wheel speeds ω_is and the linear vehicle speed v_x.

Similar to the previous test case, a nominal AC controller is trained in addition to the sliding mode controller described above with 500 fault-free episodic simulations of the vehicle dynamics with different applied pedal forces. The AC controller is provided a positive reward if the vehicle deceleration $\dot{v_x}$ matches the reference input deceleration and penalized with a negative reward proportional to the tracking error. The reward function generates the maximum negative reward if the angular speed of any wheel is less than the equivalent linear velocity of the vehicle, indicating a potential wheel lock-up condition. The range of angular speeds for each wheel is 0-100 rad/s

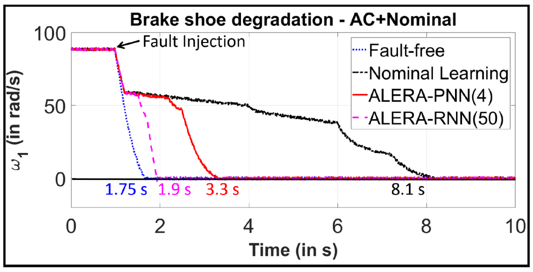

Fig. 18. The augmented scheme under ALERA achieves the functionality in presence of 20% actuator degradation with an additional stopping distance of 10 ft. The RNN-based scheme provides a 43X acceleration while the PNN-based scheme delivers 4.12X boost.

with a maximum linear velocity of 100 mph. The 5-dimensional state space is divided into grids of 10 for each dimension, thus creating $10 \times 10 \times 10 \times 10 \times 10 = 10^5$ parameters for each of the actor and critic. However, in this test case, the actor and critic parameter matrices are sparse since the wheel speeds ω_is and the vehicle speed v_x are tightly coupled. The parameter choices of the reinforcement learning and the sigmoid function selection are the same as in the previous test case.

5.2 Simulation Results

(i) *Pre-Deployment Training.* The values of the different parameters for the BBW system are M = 1,300 kg, $R = 0.4$ m, $I_w = 0.6$ kg/m^2, F_z = 2,000 N, $\alpha_{si} = 4.5$, $\kappa_{th} = 0.2$, $\eta = 0.8$, and $\nu = 1.5$. The MARS prediction function $F_c(.)$ in the error detection module is trained with braking data for 1,000 different initial vehicle speeds ranging between 50 mph and 100 mph with the temporal sliding window length chosen as $T_p = 6$ samples. The function $F_m(.)$ is selected to generate the sum of wheel speeds. A Gaussian distributed measurement noise with mean 0 and standard deviation of 2 rad/s is considered while measuring the angular speed ω_i of each wheel. The trained MARS model is evaluated on test data of 200 different braking experiments with initial vehicle speeds uniformly sampled between 50 mph and 100 mph, with 100 experiments as fault-free and 100 systems with injected faults of altering the braking torque by a uniform distribution of 20%. The detection threshold is chosen as $|e_{nom}| = 0.8$ (different from that in test case I, due to different dynamic ranges of state variables) for a target F$_1$-score of 0.9. For training the diagnostic correlator module, the fault model is defined as perturbations in the wheel speed readings (sensor errors) and corruption of the ECB signal (actuator errors) by altering the applied braking torque T_b by a uniform distribution of 20%. One thousand (1,000) fault-injected systems are created and a PNN classification with four clusters is adopted, followed by training of optimal AC controller for each cluster medoid. For the RNN-based scheme, 50 systems are selected out of 1,000 choices and an optimal AC controller is learned for each of these systems. The RNN is trained with the time-series data of $e(t)$ for all such systems to directly predict the 2×10^5 AC parameters.

(ii) *Post-Deployment Experiments.* Figure 18 shows the braking performance of the augmented scheme in ALERA where the AC network is used to supplement the performance of the sliding-mode controller. The fault-free braking is completed at $t = 1.75$s. A brake shoe degradation of 20% is injected at $t = 1.2$s, thus reducing the applied braking torque. After the actuator injection, the nonzero TD error $\delta(t)$ (not shown in the figure) guides the RL learning in the unassisted case and the learning converges at 7.2s, thus completing the vehicle stoppage at 8.1s. In comparison, the ALERA-PNN scheme with four clusters completes the learning at 2.5s and stops the vehicle at 3.3s, thus providing a learning acceleration of $\psi = 4.12$. The ALERA-RNN scheme trained with 50 pre-deployment training data completes the learning at 1.72s and stops the vehicle at 1.9s providing a boost of 43X. In both the ALERA schemes, the error $e(t)$ (not shown in the figure) is recorded from $t = 1.4$s to $t = 1.6$s.

6 TEST CASE III: SELF-BALANCING ROBOT

The objective of the self-balancing robot is to vertically stand up from a lying-down position by applying appropriate motor torque computed by the control algorithm implemented on its processor. In this study, we demonstrate that ALERA can relearn how to balance faster than traditional reinforcement learning scheme in presence of actuator degradations.

6.1 System and Controller Description

The commercially available Balboa 32U4 self-balancing robot [35] has an Atmel microcontroller along with an Arduino-enabled bootloader for programming the device. The torque to the two wheels is provided from two separate 6V, brushed DC motors controlled by on-board motor drivers. The actuator signals for controlling the motor torque are the rotational direction and speed for each motor formatted as pulse width modulated (PWM) signals from the drivers. For balancing, eight different sensor measurements are used - six degree-of-freedom (DOF) measurements (3 angular acceleration + 3 angular rate) from on-board inertial measurement unit (IMU) and two quadrature encoder measurements of the rotational velocity of the two wheels.

A hybrid controller, combining linear and nonlinear control laws, is the nominal choice for balancing the robot. Similar to the inverted pendulum case, the nonlinear controller brings the robot from a lying-down to a near-vertical position and switches to a PID controller for final balancing around the upright position.

6.2 Hardware Results

The actuator degradation is modeled by changing the motor drive signal transmitted from the micro-controller to the drivers that generate the appropriate driving current. Corrupting the motor drive signal alters the torque generated by the motors and affects the system operation. The effect of 20% actuator fault (motor drive signal altered by 20%) is illustrated in Figure 19.

Figure 19 demonstrates that without error, the robot manages to self-balance in 3s and stays upright with the angle θ around 0°. With the injected 20% error, the control algorithm attempts to swing up the robot and balance itself but fails to accomplish this task. The robot falls back to the lying down position with angle $\theta = 110^\circ$ at 3.4s and never manages to rise up and balance itself.

To implement ALERA in the hardware platform of the balancing robot, the nominal controller is augmented with a reinforcement learning controller with a reconfiguration ratio $\phi = 0.33$. The motor drive signal from the nominal controller is restricted at a maximum value of 200 rpm while the RL controller's maximum value is fixed at 100 rpm, with 300 rpm as the maximum drive restricted by the motor specifications. For practical feasibility, the fault model space is restricted to data corruptions in three signals—(i) the z-axis data from the accelerometer (alters the angular velocity), (ii) the quadrature encoder readings signifying the rpm of the motors (alters the linear speed), and (iii) the motor drive signals (alters the applied torque to the motor). The maximum deviation of each dimension is fixed at ±25% of the nominal values. To implement the checking function, the function F_m is defined to compute the normalized sum of 10 signals—8 sensor measurements and 2 actuation currents at each instant, and 40 balancing trials are performed to collect the training data. The sliding window length is $T_p = 6$ samples. A 3rd-degree polynomial

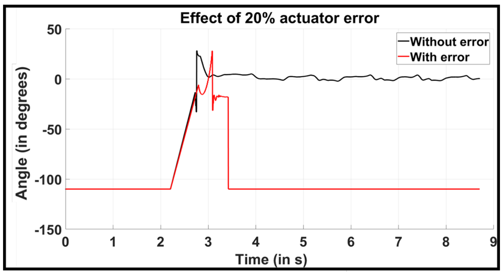

Fig. 19. With 20% injected actuator error, the robot fails to stand up and balance.

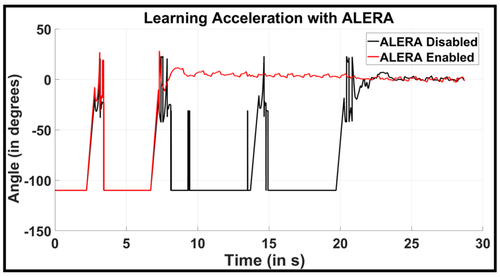

Fig. 20. The robot is balanced significantly faster than the nominal learning procedure of the RL controller by bootstrapping the learning using the error signal.

function is trained on PC to model the function F_c and is used as the checking function for the error detection module in ALERA. The trained polynomial is evaluated on test data from 20 separate balancing trials, where 10 of the trials are injected with faults uniformly sampled from the 3-dimensional fault space. Based on a target F_1-score of 0.9, the detection threshold is chosen as $e_{nom} = 0.22$. For the RL controller implementation, the four states of the system (angle, angular velocity, position, linear velocity) are discretized in $4 \times 4 \times 4 \times 4 = 256$ grid locations resulting in 256 actor and 256 critic parameter values. Fifty hardware episodic experiments are conducted such that the RL controller learns how to supplement the nominal controller in the balancing task. The 512 actor and critic parameter values are stored as nominal RL controller parameters.

For training the diagnostic correlator module, 40 faulty configurations are uniformly sampled from the 3-dimensional fault space. For these 40 configurations, the robot fails to balance using the nominal RL parameters. As described in Section 3.1, the error signal from the pre-trained polynomial model is recorded for 1 second. The 40 points are clustered in the fault parameter space based on the TD error $\delta(t)$ into 5 clusters and 5 different medoids are chosen to represent each of the clusters. For each of these five fault configurations, the learning of the nominal RL controller is allowed such that it relearns to balance the robot in presence of the injected errors. After the five RL controllers converge, the resulting AC parameter values are recorded and stored.

The 512 parameter values for the five optimal AC controllers along with the error signals for the fault sets they represent are stored in the flash memory of the microcontroller. In actual operation, the ℓ_2-norm of the recorded error signal is compared with the stored error signals and the AC parameters for the closest matching error signal is selected as the bootstrapped version of the RL controller. The balancing efforts with disabling and enabling of ALERA are shown in Figure 20.

In the ALERA-disabled scheme, the learning of the RL controller starts from the nominally learned values and through repeated episodic tasks, the learning converges and relearns to balance the robot as seen in Figure 20. In the ALERA-enabled scheme, in the first effort as the robot fails to balance itself, the error signal is recorded for 1 second and the matching algorithm bootstraps the actor-critic learning by reinitializing the parameter values from the stored repository. It is seen in Figure 20, that the robot gets balanced in the second attempt after reinitialization of the actor-critic parameter values. The nominal fault-free balancing is accomplished at 3 seconds while the ALERA-disabled scheme balances it at 23.8 seconds after four episodic experiments. The ALERA-enabled scheme manages to balance the robot at 9.3 seconds. Thus, the effective learning acceleration in this experiment is $\psi = \frac{23.8-3}{9.3-3} = \frac{20.8}{6.3} = 3.3$. This demonstrates that a learning boost of 3.3x is achieved in the actual hardware experiments using the assisted learning scheme of ALERA.

7 CONCLUSION

This article proposes a real-time control law adaptation in nonlinear systems using actor-critic network. The presented results from two nonlinear test cases and a hardware experiment demonstrate the accelerated learning potential of actor-critic networks for performance recovery with low latency. As future research, more realistic fault models will be studied to replace the synthetic fault models pursued here and self-learning capabilities for safety-critical systems will be explored where actor-critic network may not be the best controller topology because of the online experiments that it needs to execute to perform learning.

REFERENCES

[1] S. Anwar and B. Zheng. 2007. An antilock-braking algorithm for an eddy-current-based brake-by-wire system. *IEEE Transactions on Vehicular Technology* 56, 3 (May 2007), 1100–1107. DOI: https://doi.org/10.1109/TVT.2007.895604

[2] S. Banerjee, A. Banerjee, A. Chatterjee, and J. A. Abraham. 2013. Real-time checking of linear control systems using analog checksums. In *IEEE 19th International On-Line Testing Symposium (IOLTS'13)*. 122–127. DOI: https://doi.org/10.1109/IOLTS.2013.6604062

[3] S. Banerjee and A. Chatterjee. 2017. Real-time self-learning for control law adaptation in nonlinear systems using encoded check states. In *2017 22nd IEEE European Test Symposium (ETS)*. 1–6. DOI: https://doi.org/10.1109/ETS.2017.7968218

[4] S. Banerjee, A. Chatterjee, and J. A. Abraham. 2016. Efficient cross-layer concurrent error detection in nonlinear control systems using mapped predictive check states. In *2016 IEEE International Test Conference (ITC)*. 1–10. DOI: https://doi.org/10.1109/TEST.2016.7805861

[5] S. Banerjee, B. Samynathan, J. Abraham, and A. Chatterjee. 2019. Real-time error detection in nonlinear control systems using machine learning assisted state-space encoding. *IEEE Transactions on Dependable and Secure Computing* (2019), 1–1. DOI: https://doi.org/10.1109/TDSC.2019.2903049

[6] A. G. Barto, R. S. Sutton, and C. W. Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13, 5 (Sept 1983), 834–846. DOI: https://doi.org/10.1109/TSMC.1983.6313077

[7] Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. Natural actor-critic algorithms. *Automatica* 45, 11 (2009), 2471–2482. DOI: https://doi.org/10.1016/j.automatica.2009.07.008

[8] M. R. Buehner, C. W. Anderson, P. M. Young, K. A. Bush, and D. C. Hittle. 2007. Improving performance using robust recurrent reinforcement learning control. In *2007 European Control Conference (ECC)*. 1676–1681.

[9] H. Mirzaei Buini, S. Peter, and T. Givargis. 2017. Adaptive embedded control of cyber-physical systems using reinforcement learning. *IET Cyber-Physical Systems: Theory Applications* 2, 3 (2017), 127–135. DOI: https://doi.org/10.1049/iet-cps.2017.0048

[10] I. S. Comşa, S. Zhang, M. Aydin, J. Chen, P. Kuonen, and J. F. Wagen. 2014. Adaptive proportional fair parameterization based LTE scheduling using continuous actor-critic reinforcement learning. In *2014 IEEE Global Communications Conference*. 4387–4393. DOI: https://doi.org/10.1109/GLOCOM.2014.7037498

[11] M. Cutler and J. P. How. 2015. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2605–2612. DOI: https://doi.org/10.1109/ICRA.2015.7139550

[12] D. Danks and A. J. London. 2017. Regulating autonomous systems: Beyond standards. *IEEE Intelligent Systems* 32, 1 (Jan. 2017), 88–91. DOI: https://doi.org/10.1109/MIS.2017.1

[13] G. A. de Castro. 2001. *Convex Methods for the Design of Structured Controllers*. Ph.D. Dissertation. University of California, Los Angeles.

[14] Kenji Doya. 2000. Reinforcement learning in continuous time and space. *Neural Comput.* 12, 1 (Jan. 2000), 219–245. DOI: https://doi.org/10.1162/089976600300015961

[15] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (Nov 2012), 1291–1307. DOI: https://doi.org/10.1109/TSMCC.2012.2218595

[16] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema. 2012. Efficient model learning methods for actor critic control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42, 3 (June 2012), 591–602. DOI: https://doi.org/10.1109/TSMCB.2011.2170565

[17] V. Gupta, B. Hassibi, and R. M. Murray. 2004. On the synthesis of control laws for a network of autonomous agents. In *American Control Conference, 2004.*, Vol. 6. IEEE, 4927–4932.

[18] V. Gupta, B. Hassibi, and R. M. Murray. 2005. A sub-optimal algorithm to synthesize control laws for a network of dynamic agents. *Internat. J. Control* 78, 16 (2005), 1302–1313.

[19] K. S. Hwang and C. Y. Lo. 2013. Policy improvement by a model-free dyna architecture. *IEEE Transactions on Neural Networks and Learning Systems* 24, 5 (May 2013), 776–788. DOI: https://doi.org/10.1109/TNNLS.2013.2244100

[20] H. Jiang, H. Zhang, Y. Luo, and J. Han. 2018. Neural-network-based robust control schemes for nonlinear multiplayer systems with uncertainties via adaptive dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* PP, 99 (2018), 1–10. DOI: https://doi.org/10.1109/TSMC.2018.2810117

[21] Uwe Kiencke and Lars Nielsen. 2005. *Automotive Control Systems*. Springer-Verlag, Berlin.

[22] J. C. Kinsey, Q. Yang, and J. C. Howland. 2014. Nonlinear dynamic model-based state estimators for underwater navigation of remotely operated vehicles. *IEEE Transactions on Control Systems Technology* 22, 5 (Sept 2014), 1845–1854. DOI: https://doi.org/10.1109/TCST.2013.2293958

[23] Vijaymohan Konda. 2002. *Actor-critic Algorithms*. Ph.D. Dissertation. Cambridge, MA. AAI0804543.

[24] Vijay R. Konda and John N. Tsitsiklis. 2003. On actor-critic algorithms. *SIAM Journal on Control and Optimization* 42, 4 (2003), 1143–1166. DOI: https://doi.org/10.1137/S0363012901385691

[25] C. Langbort, R. S. Chandra, and R. D'Andrea. 2004. Distributed control design for systems interconnected over an arbitrary graph. *IEEE Transactions on Automatic Control* 49, 9 (2004), 1502–1519.

[26] F. L. Lewis and D. Vrabie. 2009. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine* 9, 3 (2009), 32–50. DOI: https://doi.org/10.1109/MCAS.2009.933854

[27] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis. 2012. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems* 32, 6 (Dec 2012), 76–105. DOI: https://doi.org/10.1109/MCS.2012.2214134

[28] J. Li, B. Kiumarsi, T. Chai, F. L. Lewis, and J. Fan. 2017. Off-policy reinforcement learning: Optimal operational control for two-time-scale industrial processes. *IEEE Transactions on Cybernetics* 47, 12 (Dec 2017), 4547–4558. DOI: https://doi.org/10.1109/TCYB.2017.2761841

[29] R. Li, Z. Zhao, X. Chen, J. Palicot, and H. Zhang. 2014. TACT: A transfer actor-critic learning framework for energy saving in cellular radio access networks. *IEEE Transactions on Wireless Communications* 13, 4 (April 2014), 2000–2011. DOI: https://doi.org/10.1109/TWC.2014.022014.130840

[30] Y. Liu and B. Xu. 2015. Improved protocols and stability analysis for multivehicle cooperative autonomous systems. *IEEE Transactions on Intelligent Transportation Systems* 16, 5 (Oct 2015), 2700–2710. DOI: https://doi.org/10.1109/TITS.2015.2427198

[31] H. Modares, F. L. Lewis, and Z. P. Jiang. 2016. Optimal output-feedback control of unknown continuous-time linear systems using off-policy reinforcement learning. *IEEE Transactions on Cybernetics* 46, 11 (Nov 2016), 2401–2410. DOI: https://doi.org/10.1109/TCYB.2015.2477810

[32] M. I. Momtaz, S. Banerjee, and A. Chatterjee. 2017. On-line diagnosis and compensation for parametric failures in linear state variable circuits and systems using time-domain checksum observers. In *2017 IEEE 35th VLSI Test Symposium (VTS)*. 1–6. DOI: https://doi.org/10.1109/VTS.2017.7928961

[33] S. Nahavandi. 2017. Trusted autonomy between humans and robots: Toward human-on-the-loop in robotics and autonomous systems. *IEEE Systems, Man, and Cybernetics Magazine* 3, 1 (Jan 2017), 10–17. DOI: https://doi.org/10.1109/MSMC.2016.2623867

[34] Jan Peters and Stefan Schaal. 2008. Natural actor-critic. *Neurocomputing* 71, 7–9 (2008), 1180–1190. DOI: https://doi.org/10.1016/j.neucom.2007.11.026

[35] Pololu Robotics and Electronics. [n.d.]. Balboa 32U4 balancing robot kit. ([n.d.]). https://goo.gl/FUJeEW.

[36] I. E. Skoulakis and M. G. Lagoudakis. 2012. Efficient reinforcement learning in adversarial games. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, Vol. 1. 704–711. DOI: https://doi.org/10.1109/ICTAI.2012.100

[37] R. Song, F. L. Lewis, Q. Wei, and H. Zhang. 2016. Off-policy actor-critic structure for optimal control of unknown systems with disturbances. *IEEE Transactions on Cybernetics* 46, 5 (May 2016), 1041–1050. DOI: https://doi.org/10.1109/TCYB.2015.2421338

[38] Richard S. Sutton and Andrew G. Barto. 1998. *Generalization and Function Approximation*. MIT Press, 193–226. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6282963.

[39] Onder Tutsoy, Duygun Erol Barkana, and Sule Colak. 2017. Learning to balance an NAO robot using reinforcement learning with symbolic inverse kinematic. *Transactions of the Institute of Measurement and Control* 39, 11 (2017), 1735–1748. DOI: https://doi.org/10.1177/0142331216645176

[40] Onder Tutsoy and Martin Brown. 2016. Reinforcement learning analysis for a minimum time balance problem. *Transactions of the Institute of Measurement and Control* 38, 10 (2016), 1186–1200. DOI: https://doi.org/10.1177/0142331215581638

[41] B. Wang, D. Zhao, C. Li, and Y. Dai. 2015. Design and implementation of an adaptive cruise control system based on supervised actor-critic learning. In *2015 5th International Conference on Information Science and Technology (ICIST)*. 243–248. DOI: https://doi.org/10.1109/ICIST.2015.7288976

[42] D. Wang, D. Liu, Q. Zhang, and D. Zhao. 2016. Data-based adaptive critic designs for nonlinear robust optimal control with uncertain dynamics. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46, 11 (Nov 2016), 1544–1555. DOI: https://doi.org/10.1109/TSMC.2015.2492941

[43] Z. Wang, L. Liu, H. Zhang, and G. Xiao. 2016. Fault-tolerant controller design for a class of nonlinear MIMO discrete-time systems via online reinforcement learning algorithm. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46, 5 (May 2016), 611–622. DOI: https://doi.org/10.1109/TSMC.2015.2478885

[44] K. Yoshida. 1999. Swing-up control of an inverted pendulum by energy-based methods. In *1999 American Control Conference.*, Vol. 6. 4045–4047. DOI: https://doi.org/10.1109/ACC.1999.786297

[45] H. Zhang, H. Jiang, Y. Luo, and G. Xiao. 2017. Data-driven optimal consensus control for discrete-time multi-agent systems with unknown dynamics using reinforcement learning method. *IEEE Transactions on Industrial Electronics* 64, 5 (May 2017), 4091–4100. DOI: https://doi.org/10.1109/TIE.2016.2542134

[46] Y. Zhu and D. Zhao. 2014. A data-based online reinforcement learning algorithm with high-efficient exploration. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 1–6. DOI: https://doi.org/10.1109/ADPRL.2014.7010631

[47] Yuanheng Zhu, Dongbin Zhao, and Haibo He. 2014. An high-efficient online reinforcement learning algorithm for continuous-state systems. In *11th World Congress on Intelligent Control and Automation*. 581–586. DOI: https://doi.org/10.1109/WCICA.2014.7052778

Received January 2019; revised May 2019; accepted May 2019