Proceedings of IMECE07
2007 ASME International Mechanical Engineering Congress and Exposition
November 11-15, 2007, Seattle, Washington, USA

IMECE2007-41258

A STATE-SPACE REPRESENTATION MODEL AND LEARNING ALGORITHM FOR REAL-TIME DECISION-MAKING UNDER UNCERTAINTY

Andreas A. Malikopoulos
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48109, U.S.A
amaliko@umich.edu

Panos Y. Papalambros
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48109, U.S.A
pyp@umich.edu

Dennis N. Assanis
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48109, U.S.A
assanis@umich.edu

ABSTRACT

Modeling dynamic systems incurring stochastic disturbances for deriving a control policy is a ubiquitous task in engineering. However, in some instances obtaining a model of a system may be impractical or impossible. Alternative approaches have been developed using a simulation-based stochastic framework, in which the system interacts with its environment in real time and obtains information that can be processed to produce an optimal control policy. In this context, the problem of developing a policy for controlling the system's behavior is formulated as a sequential decision-making problem under uncertainty. This paper considers real-time sequential decision-making under uncertainty modeled as a Markov Decision Process (MDP). A state-space representation model is constructed through a learning mechanism and is used to improve system performance over time. The model allows decision making based on gradually enhanced knowledge of system response as it transitions from one state to another, in conjunction with actions taken at each state. A learning algorithm is implemented realizing in real time the optimal control policy associated with the state transitions. The proposed method is demonstrated on the single cart-pole balancing problem and a vehicle cruise control problem.

Keywords: sequential decision-making under uncertainty, Markov Decision Process (MDP), reinforcement learning, learning algorithms, inverted pendulum, vehicle cruise control

1. INTRODUCTION

Deriving a control policy for dynamic systems is an off-line process in which various methods from control theory are utilized iteratively. These methods aim to determine the policy that satisfies the system's physical constraints while optimizing specific performance criteria. A challenging task in this process is to derive a mathematical model of the system's dynamics that can adequately predict the response of the physical system to all anticipated inputs. Exact modeling of complex engineering systems, however, may be infeasible or expensive. Viable alternative methods have been developed enabling the real-time implementation of control policies for systems when an accurate model is not available. In this framework, the system interacts with its environment, and obtains information enabling it to improve its future performance by means of rewards associated with control actions taken. This interaction portrays the learning process conveyed by the progressive enhancement of the system's "knowledge" regarding the course of action (control policy) that maximizes the accumulated rewards with respect to the system's operating point (state). The environment is assumed to be non-deterministic; namely, taking the same action in the same state on two different stages, the system may transit to a different state and receive a dissimilar reward in the subsequent stage. Consequently, the problem of developing a policy for controlling the system's behavior is formulated as a sequential decision-making problem under uncertainty.

Dynamic programming (DP) has been widely employed as the principal method for analysis of sequential decision-making problems [1]. Algorithms, such as value iteration, and policy iteration, have been extensively utilized in solving deterministic and stochastic optimal control problems, Markov and semi-Markov decision problems, min-max control problems, and sequential games. However, the computational complexity of these algorithms in some occasions may be prohibitive and can grow intractably with the size of the

problem and its related data, referred to as the DP "curse of dimensionality" [2]. In addition, DP algorithms require the realization of the conditional probabilities of state transitions and the associated rewards, implying *a priori* knowledge of the system dynamics.

Alternative approaches for solving sequential decision-making problems under uncertainty have been primarily developed in the field of Reinforcement Learning (RL) [3, 4]. RL has aimed to provide simulation-based algorithms, founded on DP, for learning control policies of complex systems, where exact modeling is infeasible or expensive [5]. A major influence on research leading to current RL algorithms has been Samuel's method [6, 7], used to modify a heuristic evaluation function for deriving optimal board positions in the game of checkers. In this algorithm, Samuel represented the evaluation function as a weighted sum of numerical features and adjusted the weights based on an error derived from comparing evaluations of current and predicted board positions. This approach was refined and extended by Sutton [8, 9] to introduce a class of incremental learning algorithms, Temporal Difference (TD). TD algorithms are specialized for deriving optimal policies for incompletely known systems, using past experience to predict their future behavior. Watkins [10, 11] extended Sutton's TD algorithms and developed an algorithm for systems to learn how to act optimally in controlled Markov domains by explicitly utilizing the theory of DP. A strong condition implicit in the convergence of Q-learning to an optimal control policy is that the sequence of stages that forms the basis of learning must include an infinite number of stages for each initial state and action. However, Q-learning is considered the most popular and efficient model-free learning algorithm in deriving optimal control policies in Markov domains [12]. Schwartz [13] explored the potential of adapting Q-learning to an average-reward framework with his R-learning algorithm; Bertsekas and Tsitsiklis [3] presented a similar to Q-learning average-reward algorithm. Mahadevan [14] surveyed reinforcement-learning average-reward algorithms and showed that these algorithms do not always produce bias-optimal control policies.

Although many of these algorithms are eventually guaranteed to find optimal policies in sequential decision-making problems under uncertainty, their use of the accumulated data acquired over the learning process is inefficient, and they require a significant amount of experience to achieve good performance [12]. This requirement arises due to the formation of these algorithms in deriving optimal policies without learning the system models *en route*. Algorithms for computing optimal policies by learning the models are especially important in applications in which real-world experience is considered expensive. Sutton's Dyna architecture [15, 16] exploits strategies which simultaneously utilize experience in building the model and adjust the derived policy. Prioritized sweeping [17] and Queue-Dyna [18] are similar methods concentrating on the interesting subspaces of the state-action space. Barto *et al.* [19] developed another method, called Real-Time Dynamic Programming (RTDP), referring to the cases in which concurrently executing DP and control processes influence one another. RTDP focuses the computational effort on the state-subspace that the system is most likely to occupy. This method is specific to problems in which the system needs to achieve particular goal states and the initial cost of every goal state is zero.

This paper considers the problem of deriving optimal policies in real-time sequential decision-making under uncertainty that can be modeled as a Markov Decision Process (MDP). A state-space representation model is constructed through a learning mechanism and is used to improve system performance over time. The model accumulates gradually enhanced knowledge of system response as it transitions from one state to another, in conjunction with actions taken at each state. A learning algorithm is implemented realizing in real time the optimal course of action (control policy) associated with the state transitions. The major difference between the proposed method and the existing RL algorithms is that the latter consists of evaluation functions attempting to successively approximate the Bellman equation [20]. The proposed real-time learning method, on the contrary, utilizes an evaluation function which considers the expected reward that can be achieved by state transitions forward in time. This approach is especially appealing to learning engineering systems in which the initial state is not fixed, and recursive updates of the evaluation functions to approximate the Bellman equation would demand significant amount of experience to achieve the desired system performance.

In the following section the mathematical framework for modeling sequential decision-making problems under uncertainty is presented and the predictive optimal decision-making learning method is introduced. The performance of the proposed method is demonstrated on the single cart-pole balancing problem, in Section 3 and, on a vehicle cruise-control problem, in Section 4. Conclusions are presented in Section 5.

2. PROBLEM FORMULATION

A large class of sequential decision-making problems under uncertainty can be modeled as a Markov Decision Process (MDP). MDP, extensively covered by Puterman [21], provides the mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the decision maker. Decisions are made at points of time referred to as decision epochs, and the time domain can be either discrete or continuous. The focus of this paper is on discrete-time decision-making problems.

The Markov decision process model consists of five elements: (a) decision epochs; (b) states; (c) actions; (d) the transition probability matrix; and (e) the transition reward matrix. In this framework, the decision maker is faced with the problem of influencing system behavior as it evolves over time, by making decisions (choosing actions). The objective of the decision maker is to select the course of action (control policy)

which causes the system to perform optimally with respect to some predetermined optimization criterion. Decisions must anticipate rewards (or costs) associated with future system states-actions.

At each decision epoch, the system occupies a state s_i from the finite set of all possible system states $\mathcal{S}$

$$\mathcal{S} = \{s_i \mid i = 1,2,...,N\},\ N \in \mathcal{N}. \quad (1)$$

In this state $s_i \in \mathcal{S}$, the decision maker has available a set of allowable actions, $\alpha \in A(s_i)$, $A(s_i) \subseteq \mathcal{A}$, where $\mathcal{A}$ is the finite action space

$$\mathcal{A} = \bigcup_{s_i \in \mathcal{S}} A(s_i). \quad (2)$$

The decision-making process occurs at each of a sequence of decision epochs $\mathcal{T} = 0,1,2,...,M$, $M \in \mathcal{N}$. At each epoch, the decision maker observes a system's state $s_i \in \mathcal{S}$, and executes an action $\alpha \in A(s_i)$, from the feasible set of actions $A(s_i) \subseteq \mathcal{A}$ at this state. At the next epoch, the system transits to the state $s_j \in \mathcal{S}$ imposed by the conditional probabilities $p(s_j \mid s_i, \alpha)$, designated by the transition probability matrix $\mathbf{P}(\cdot,\cdot)$. The conditional probabilities of $\mathbf{P}(\cdot,\cdot)$, $p : \mathcal{S} \times \mathcal{A} \to [0,1]$, satisfy the constraint

$$\sum_{j=1}^{N} p(s_j \mid s_i, \alpha) = 1. \quad (3)$$

Following this state transition, the decision maker receives a reward associated with the action α, $R(s_j \mid s_i, \alpha), R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ as imposed by the transition reward matrix $\mathbf{R}(\cdot,\cdot)$. The states of an MDP possess the Markov property, stating that the conditional probability distribution of future states of the process depends only upon the current state and not on any past states, i.e., it is conditionally independent of the past states (the path of the process) given the present state. Mathematically, the Markov property states that

$$\begin{aligned} &p(X_{n+1} = s_j \mid X_n = s_i, X_{n-1} = s_{i-1}, ..., X_0 = s_0) = \\ &= p(X_{n+1} = s_j \mid X_n = s_i). \end{aligned} \quad (4)$$

2.1 Optimal Policies and Performance Criteria

The solution to an MDP can be expressed as an admissible control policy such that a given performance criterion is optimized over all admissible policies Π. An admissible policy consists of a sequence of functions

$$\pi = \{\mu_0, \mu_1, ..., \mu_M\}, \quad (5)$$

where μ_T maps states s_i into actions $\alpha = \mu_T(s_i)$ and is such that $\mu_T(s_i) \in A(s_i), \forall s_i \in \mathcal{S}$.

Consequently, given an initial state at decision epoch $\mathcal{T} = 0$, s_i, and an admissible policy $\pi = \{\mu_0, \mu_1, ..., \mu_M\}$, the expected accumulated undiscounted value of the rewards of the decision maker is given by the Bellman optimality equation

$$V(s_i) = E\{R(s_N \mid s_{N-1}, \mu(s_{N-1})) + \sum_{T=0}^{M-1} V_T(s_j \mid s_i, \mu(s_i))\}, \quad (6)$$
$$\forall s_i, s_j \in \mathcal{S},\ i, j = 1,2,...,N,\ N \in \mathcal{N}.$$

In the finite-horizon context the decision maker should maximize the accumulated value for the next M epochs. An optimal policy $\pi^* \in \Pi$ is one that maximizes the overall expected accumulated value of the rewards

$$V_{\pi^*}(s_i) = \max_{\pi \in \Pi} V_\pi(s_i). \quad (7)$$

Consequently, the optimal policy $\pi^* = \{\mu_0^*, \mu_1^*, ..., \mu_M^*\}$ for the M-decision epoch sequence is

$$\pi^* = \arg\max_{\pi \in \Pi} V_{\pi^*}(s_i). \quad (8)$$

The finite-horizon model is appropriate when the decision-maker's "lifetime" is known, namely, the terminal epoch of the decision-making sequence. For problems with a very large number of decision epochs, however, the infinite-horizon model is more appropriate. In this context, the overall expected undiscounted reward is:

$$V_{\pi^*}(s_i) = \lim_{M \to \infty} \max_{\pi \in \Pi} V_\pi(s_i). \quad (9)$$

This relation is valuable computationally and analytically, and it holds under certain conditions [22].

2.2 Predictive Optimal Decision-Making State-Space Representation

The predictive optimal decision-making (POD) learning method consists of a new state-space system representation and a learning algorithm. The state-space representation accumulates gradually enhanced knowledge of the system's transition from each state to another in conjunction with actions taken for each state. This knowledge is expressed in terms of an expected evaluation function associated with each state. While the model's knowledge is advanced, the learning algorithm realizes, at each decision epoch, the course of action that guarantees at least a minimum value of the overall reward for both the current state and the subsequent states.

The major difference between the proposed learning method and the existing RL methods is that the latter consists of evaluation functions attempting to successively approximate the Bellman equation, Eq. (6). These evaluation functions assign to each state the total reward expected to accumulate over time starting from a given state when a policy π is employed. The proposed real-time learning method, on the contrary, utilizes an evaluation function which considers the expected reward that can be achieved by state transitions forward in time. This approach is especially appealing to learning engineering systems in which the initial state is not fixed [23, 24], and recursive updates of the evaluation functions to approximate Eq.(6) would demand a huge number of iterations to achieve the desired system performance.

The new state-space representation defines the POD domain $\tilde{\mathcal{S}}$. It is implemented by a mapping H from the

Cartesian product of the finite state space and action space of the MDP:

$$H : (S \times A) \times (S \times A) \to \tilde{S}, \quad (10)$$

where $S = \{s_i \mid i = 1,2,...,N\}$, $N \in \mathcal{N}$ denotes the Markov state space, and $A = \bigcup_{s_i \in S} A(s_i), \forall s_i \in S$ stands for a finite action space. This mapping generates an indexed family of subsets, $\tilde{S}_{s_i}$, for each state $s_i \in S$, defined as Predictive Representation Nodes (PRNs). Each PRN is constituted by a set of POD states, $\tilde{s}_m^i \in \tilde{S}_{s_i}$,

$$\tilde{S}_{s_i} = \{\tilde{s}_m^i \mid i, m = 1,2,...,N,\ N = |S| \in \mathcal{N}\}, \quad (11)$$

Each POD state $\tilde{s}_m^i \in \tilde{S}$ essentially represents a Markov state transition from $s_i \in S$ to $s_m \in S$. PRNs partition the POD domain insofar as the POD underlying structure captures the state transitions in the Markov domain. Consequently, a PRN is defined as

$$\tilde{S}_{s_i} = \{\tilde{s}_m^i \mid \tilde{s}_m^i \equiv s_i \underset{\mu(s_i) \in A(s_i)}{\to} s_m, \sum_{m=1}^{N} p(s_m \mid s_i, \mu(s_i)) = 1, N = |S|\},$$
$$\forall s_i, s_m \in S,\ \forall \mu(s_i) \in A(s_i), \quad (12)$$

the union of which defines the POD domain

$$\tilde{S} = \bigcup_{\tilde{s}_m^i \in \tilde{S}_{s_i}} \tilde{S}_{s_i}, \text{ with} \quad (13)$$

$$\bigcap_{\tilde{s}_m^i \in \tilde{S}_{s_i}} \tilde{S}_{s_i} = \varnothing. \quad (14)$$

Each PRN, $\tilde{S}_{s_i}$, corresponds to a Markov state, $s_i \in S$, and portrays all possible transitions occurring from this state s_i to the other states $s_m \in S$. PRNs, constituting the fundamental aspect of the POD state representation, provide an assessment of the Markov state transitions along with the actions executed at each state. This assessment aims to establish a necessary embedded property of the new state representation so as to consider the potential transitions that can occur in subsequent decision epochs. The assessment is expressed by means of the PRN value, $\bar{R}_{s_i}(\tilde{s}_m^i \mid \mu(s_i))$, which accounts for the maximum average expected reward that can be achieved by transitions occurring inside a PRN. Consequently, the PRN value is defined as

$$\bar{R}_{s_i}(\tilde{s}_m^i \mid \mu(s_i)) = \max_{\mu(s_i) \in A} \left(\frac{\sum_{m=1}^{N} p(s_m \mid s_i, \mu(s_i)) \cdot R(s_m \mid s_i, \mu(s_i))}{N} \right),$$
$$\forall \tilde{s}_m^i \in \tilde{S}, \forall s_i, s_m \in S,\ \forall \mu(s_i) \in A(s_i), \text{and } N = |S|. \quad (15)$$

The PRN value is exploited by POD state representation as an evaluation metric to estimate the subsequent Markov state transitions. The estimation property is founded on the assessment of POD states by means of an expected evaluation function, $R_{PRN}^i(\tilde{s}_m^i, \mu(s_i))$, defined as

$$R_{PRN}^i(\tilde{s}_m^i, \mu(s_i)) = \{ p(s_m \mid s_i, \mu(s_i)) \cdot R(s_m \mid s_i, \mu(s_i)) + \\ + \bar{R}_{s_m}(\tilde{s}_j^m \mid \mu(s_m)) \}, \quad (16)$$
$$\forall \tilde{s}_m^i, \tilde{s}_j^m \in \tilde{S}, \forall s_i, s_m \in S,\ \forall \mu(s_i) \in A(s_i), \forall \mu(s_m) \in A(s_m).$$

Consequently, employing the POD evaluation function through Eq. (16), each POD state, $\tilde{s}_m^i \in \tilde{S}_{s_i}$, is comprised of an overall reward corresponding to: (a) the expected reward of transiting from state s_i to s_m (implying also the transition from the PRN $\tilde{S}_{s_i}$ to $\tilde{S}_{s_m}$); and (b) the maximum average expected reward when transiting from s_m to any other Markov state (transition occurring into $\tilde{S}_{s_m}$).

While the system interacts with its environment, the POD model learns the system dynamics in terms of the Markov state transitions. The POD state representation attempts to provide a process in realizing the sequences of state transitions that occurred in the Markov domain, as infused in PRNs. The different sequences of the Markov state transitions are captured by the POD states and evaluated through the expected evaluation functions given in Eq. (16). Consequently, the highest value of the expected evaluation function at each POD state essentially estimates the subsequent Markov state transitions with respect to the actions taken. As the process is stochastic, however, it is still necessary for the real-time learning method to build a decision-making mechanism of how to select actions.

The learning performance is closely related to the exploration-exploitation strategy of the action space. More precisely, the decision maker has to exploit what is already known regarding the correlation involving the admissible state-action pairs that maximize the rewards, and also to explore those actions that have not yet been tried for these pairs to assess whether these actions may result in higher rewards. A balance between an exhaustive exploration of the environment and the exploitation of the learned policy is fundamental to reach nearly optimal solutions in few decision epochs and, thus, enhancing the learning performance. This exploration-exploitation dilemma has been extensively reported in the literature. Iwata *et al.* [25] proposed a model-based learning method extending Q-learning and introducing two separated functions based on statistics and on information by applying exploration and exploitation strategies. Ishii *et al.* [26] developed a model-based reinforcement learning method utilizing a balance parameter, controlled through variation of action rewards and perception of environmental change. Chan-Geon *et al.* [27] proposed an exploration-exploitation policy in Q-learning consisting of an auxiliary Markov process and the original Markov process. Miyazaki *et al.* [28] developed a unified learning system realizing the tradeoff between exploration and exploitation. Hernandez-Aguirre *et al.* [29] analyzed the problem of exploration-exploitation in the context of the probably approximately correct framework and studied whether it is possible to give bounds on the complexity of the

exploration needed to achieve a fixed approximation error over the action value function with a given probability.

An exhaustive exploration of the environment is necessary to evade prematurely convergence on a sub-optimal solution even if this may result in both scarifying the system's performance in the short run and increasing the learning time. In our case it is assumed that, for any state $s_i \in \mathcal{S}$, all actions of the feasible action set $\mu(s_i) \in A(s_i)$ are selected by the decision maker at least once. At the early decision epochs and until full exploration of the action set $A(s_i), \forall s_i \in \mathcal{S}$ occurs, the mapping from the states to probabilities of selecting the actions is constant; namely, the actions for each state are selected randomly with the same probability

$$p(\mu(s_i) \mid s_i) = \frac{1}{|A(s_i)|}, \forall \mu(s_i) \in A(s_i), \forall s_i \in \mathcal{S}. \quad (17)$$

When the exploration phase is complete, the POD learning algorithm is utilized to build up the decision-making mechanism.

2.3 The Predictive Optimal Decision-Making Learning Algorithm

The POD state representation attempts to provide an efficient process in realizing the state transitions that occurred in the Markov domain. The different sequences of the state transitions are captured by the POD states and evaluated through the expected evaluation functions given in Eq. (16). Consequently, the highest value of the expected evaluation function at each PRN essentially predicts the Markov state transition that will occur in the future. As the process is stochastic, however, it is still necessary for the decision maker to build a decision-making mechanism of how to make decisions (select actions). The POD learning algorithm aims to provide this mechanism.

The principle of the POD learning algorithm is founded on the theory of stochastic control problems with unknown disturbance distribution- also known as games against nature. The decision-making mechanism is modeled as a zero-sum stochastic game between the decision maker (controller) and an "opponent" (environment). The solution of this game is derived utilizing the mini-max theorem. Each POD state, $\tilde{s}_m^i \in \tilde{\mathcal{S}}_{s_i}$, corresponds to a completed game that started at the Markov state $s_i \in \mathcal{S}$ and ended up at $s_m \in \mathcal{S}$. At the state s_i, the decision maker has a set of strategies (actions) $\mu(s_i) \in A(s_i)$ available to play. Similarly, the environment's set of strategies are the Markov states $\mathcal{S} = \{s_i \mid i = 1, 2, ..., N\}$, $N \in \mathcal{N}$. During the learning process, this game has been played insofar as the decision maker forms a belief about the environment's behavior by fully exploring all available strategies, $\mu(s_i) \in A(s_i)$. Consequently, at the state s_i, the decision maker is able to predict the subsequent states to be selected by the environment by means of the PRN expected evaluation functions, $R_{PRN}^i(\tilde{s}_m^i, \mu(s_i))$. However, to handle the uncertainty of this prediction, the decision maker selects his/her strategy by means of maximizing the minimum expected accumulated reward related to both immediate state transition and subsequent transitions, namely,

$$\pi^*(s_i) = \arg\max_{\mu(s_i) \in A(s_i)} \left\{ \min_{s_m \in \mathcal{S}} \left(R_{PRN}^i(\tilde{s}_m^i, \mu(s_i)) \right) \right\}, \quad \forall \tilde{s}_m^i \in \tilde{\mathcal{S}}, \forall s_i \in \mathcal{S}, \forall \mu(s_i) \in A(s_i). \quad (18)$$

Consequently, the decision maker seeks for the max-min policy $\pi^* \in \Pi$, which guarantees the best performance in the worst possible situation.

3. CASE STUDY ONE

3.1 The Single Cart-Pole Balancing Problem

The overall performance of the POD real-time learning model is evaluated on the basis of its application to the inverted pendulum balancing problem. The inverted pendulum involves a pendulum hinged to the top of a wheeled cart as illustrated in Figure 1. The objective of POD is to balance the pendulum having no prior knowledge about the system dynamics utilizing only real-time measurements.

Realizing the balance control policy of a single inverted pendulum without *a priori* knowledge of the system's model has been extensively reported in the literature for the evaluation of learning algorithms. Anderson [30] implemented a neural network reinforcement-learning method to generate successful action sequences. Two neural networks having a similar structure were employed to learn two functions: (a) an action function mapping the current state into control actions, and (b) an evaluation action mapping the current state into an evaluation of that state. These two networks were trained utilizing reinforcement learning by evaluating the performance of the network and compared to real-time measurements. Williams *et al.* [31] proposed a learning architecture for training a neural network controller to provide the appropriate control force to balance the inverted pendulum. One network for the identification of the plant dynamics and one for the controller were employed. Zhidong *et al.* [32] implemented a

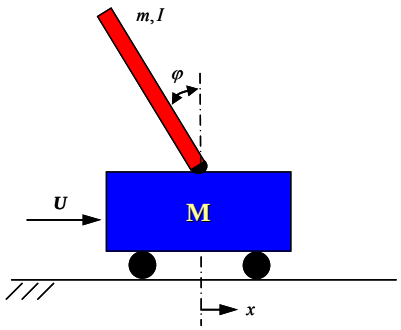

Figure 1. The inverted pendulum.

"neural-fuzzy BOXES" control system by neural networks and utilized reinforcement learning for the training. Jeen-Shing *et al.* [33] proposed a defuzzification method incorporating a genetic algorithm to learn the defuzzification factors. Mustapha *et al.* [34] developed an actor-critic reinforcement learning algorithm represented by two adaptive neural-fuzzy systems. Si *et al.* [35] proposed a generic on-line learning control system similar to Anderson's utilizing neural networks and evaluated it through its application to both a single and double cart-pole balancing problem. The system utilizes two neural networks, and employs the action- dependent heuristic dynamic programming to adapt the weights of the networks.

In the implementation of the POD on the single inverted pendulum presented here, two major variations are considered: (a) a single look-up table-based representation is employed for the controller to develop the mapping from the system's Markov states to optimal actions, and (b) two system's state variables are selected to represent the Markov state. The latter introduces uncertainty and thus a conditional probability distribution associating the state transitions with respect to the actions taken. Consequently, the POD method is evaluated in deriving the optimal policy (balance control policy) in a sequential decision making problem under uncertainty.

The governing equations, derived from the free body diagram of the system, shown in Figure 2, are:

$$(M+m)\ddot{x} + b\dot{x} + mL\ddot{\varphi}\cos\varphi - mL\dot{\varphi}^2\sin\varphi = U, \qquad (19)$$

$$mL\ddot{x}\cos\varphi + (I+mL^2)\ddot{\varphi} + mgL\sin\varphi = 0, \qquad (20)$$

where $M = 0.5$ kg, $m = 0.2$ kg, $b = 0.1$ $\frac{\text{N sec}}{\text{m}}$

$I = 0.006$ kg m^2, $g = 9.81$ $\frac{\text{m}}{\text{sec}^2}$, and $L = 0.3$ m.

The goal of the learning controller is to realize in real time the force, U, of a fixed magnitude to be applied either to the right or the left direction so that the pendulum stands balanced when released from any angle, φ, between 3° and -3°. The system is simulated by numerically solving the nonlinear differential equations (19) and (20) employing the explicit Runge-Kutta method with a time step of τ =0.02 sec. The simulation is conducted by observing the system's states and executing actions (control force U) with a sample rate T =0.02 sec (50 Hz). This sample rate defines a sequence of decision-making epochs, $\mathcal{T} = 0,1,2,...,M$, $M \in \mathcal{N}$.

The system is fully specified by four state variables: (a) the position of the cart on the track, $x(t)$; (b) the cart velocity, $\dot{x}(t)$; (c) the pendulum's angle with respect to the vertical position, $\varphi(t)$; and (d) the angular velocity, $\dot{\varphi}(t)$. However, to incorporate uncertainty, the Markov states are selected to be only the pair of the pendulum's angle and angular velocity, namely, the finite state space $\mathcal{S}$, in Eq. (1) is defined as

$$\mathcal{S} = \{s_i \mid s_i = (\varphi, \dot{\varphi})\}. \qquad (21)$$

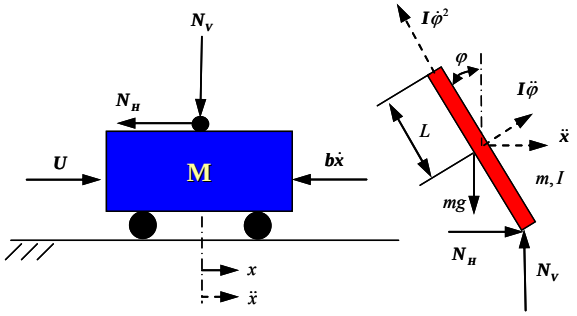

Figure 2. Free body diagram of the system.

Consequently, at state $s_i \in \mathcal{S}$ and executing a control force value, $U(s_i)$, the system will end up at state $s_j \in \mathcal{S}$ with a conditional probably $p(s_j \mid s_i, U(s_i))$. The control force, $U(s_i)$, selects values from the finite set $\mathcal{A}$, defined as

$$\mathcal{A} = A(s_i) = [-3N, 3N], \text{ where } i = 1,2,...,N,\ N = |\mathcal{S}|. \qquad (22)$$

The decision-making process occurs at each of a sequence of epochs $\mathcal{T} = 0,1,2,...,M$, $M \in \mathcal{N}$. At each decision epoch, the learning controller observes the system's state $s_i \in \mathcal{S}$, and executes a control force value $U(s_i) \in A(s_i)$. At the next decision epoch, the system transits to another state $s_j \in \mathcal{S}$ imposed by the conditional probabilities $p(s_j \mid s_i, U(s_i))$, and receives a numerical reward (the pendulum's angle φ).

The inverted pendulum is simulated repeatedly for different initial angles, φ, between 3° and -3° utilizing the POD learning method. The simulation lasts for 50 sec and each complete simulation defines one iteration. If at any instant during the simulation, the pendulum's angle, φ, becomes greater than 3° or less than -3°, this constitutes a failure, denoted by stating that there was one iteration associated with a failure. If, however, no failure occurs during the simulation, this is denoted by stating that there was one iteration associated with no failure.

3.2 Simulation Results

After completing the learning process, the controller employing the POD learning method realizes the balance control policy of the pendulum, as illustrated in Figure 3. In some instances, however, the system's response demonstrates some overshoots or delays during the transient period, shown in Figure 4. This can be handled by a denser parameterization of the state-space or adding a penalty in long transient responses. The efficiency of the POD learning method in deriving the optimal balance control policy that stabilizes the system is illustrated in Figure 5. It is noted that after POD realizes the optimal policy in 749 failures and, afterwards, as the number of iterations continues no further failures occur.

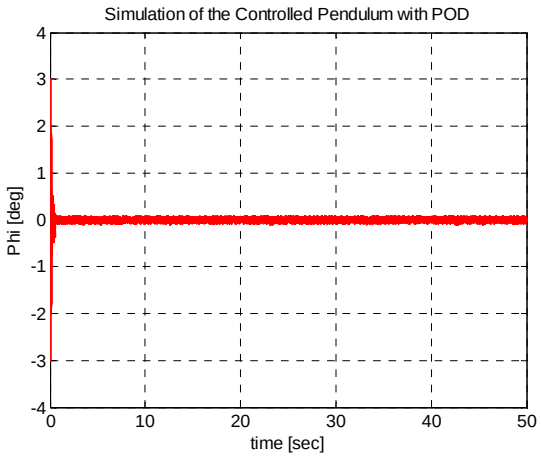

Figure 3. Simulation of the system after learning the balance control policy with POD for different initial conditions.

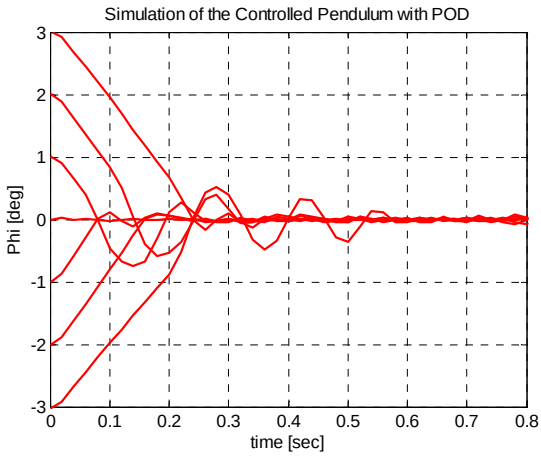

Figure 4. Simulation of the system after learning the balance control policy with POD for different initial conditions (zoom in).

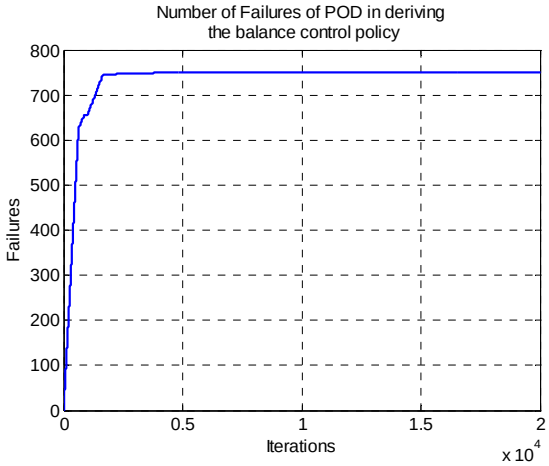

Figure 5. Number of failures until POD derives the balance control policy.

4. CASE STUDY TWO

4.1 Vehicle Cruise Control

The POD real-time learning method introduced in the previous sections is now applied to a vehicle cruise-control problem. Cruise control automatically regulates the vehicle's longitudinal velocity by suitably adjusting the gas pedal position. A vehicle cruise-control system is activated by the driver who desires to maintain a constant speed in long highway driving. The driver activates the cruise controller while driving at a particular speed, which is then recorded as the desired or set-point speed to be maintained by the controller. The main goal in designing a cruise control algorithm is to maintain vehicle speed smoothly but accurately, even under large variation of plant parameters (e.g., the vehicle's varying mass in terms of the number of passengers) and road grade. In the case of passenger cars, however, vehicle mass may change noticeably but is within a small range. Therefore, powertrain behavior might not vary significantly.

The objective of the POD learning cruise controller is to realize in real time the control policy (gas pedal position) that maintains the vehicle speed as set by the driver under a great range of different road grades. Implementing learning vehicle cruise controllers has been addressed previously employing learning and active control approaches. Zhang *et al.* [36] implemented learning control based on pattern recognition to regulate in real time the parameters of an PID cruise controller. Shahdi *et al.* [37] proposed an active learning method to extract the driver's behavior and to derive control rules for a cruise control system. However, no attempt has been reported in implementing a learning automotive vehicle cruise controller utilizing the principle of reinforcement learning, i.e., enabling the controller to improve its performance over time by learning from its own failures through a reinforcement signal from the external environment, and thus, attempting to improve future performance.

The software package enDYNA by TESIS [38], suitable for real-time simulation of internal combustion engines, is used to evaluate the performance of the POD learning cruise controller. The software simulates the longitudinal vehicle dynamics with a highly variable drive train including the modules of starter, brake, clutch, converter, and transmission. In the driving mode the engine is operated by means of the usual vehicle control elements just as a driver would do. In addition, a mechanical parking lock and the uphill grade can be set. The driver model is designed to operate the vehicle at given speed profiles (driving cycles). It actuates the starter, accelerator, clutch and brake pedals according to the profile specification, and also shifts gears. In this example, an existing vehicle model is selected representing a midsize passenger car carrying an 1.9L turbocharged diesel engine.

When activated, the learning cruise controller bypasses the driver model and takes over the vehicle's cruising. The Markov states are defined to be the pair of the transmission gear and the

difference between the desired and actual vehicle speed, ΔV, namely,

$$\mathcal{S} = \{s_i \mid s_i = (gear, \Delta V)\}. \quad (23)$$

The actions, α, correspond to the gas pedal position and can take values from the feasible set $\mathcal{A}$, defined as

$$\mathcal{A} = A(s_i) = [0, 0.7], \text{ where } i = 1, 2, ..., N, \ N = |\mathcal{S}|. \quad (24)$$

To incorporate uncertainty the vehicle is simulated in a great range of different road grades from 0° to 10°. Consequently, at state $s_i \in \mathcal{S}$ and executing a control action (gas pedal position), the system transits to another state $s_j \in \mathcal{S}$ with a conditional probably $p(s_j \mid s_i, \alpha(s_i))$, since the acceleration capability of a vehicle varies at different road grades. As a consequence of this state transition the system receives a numerical reward (difference between the desired and actual vehicle speed).

4.2 Simulation Results

After completing four simulations of each road grade, the POD cruise controller realizes the control policy (gas pedal position) to maintain the vehicle's speed at the desired set point. The vehicle model was initiated from zero speed. The driver model, following the driving cycle, accelerated the vehicle up to 40mph and at 10sec activated the POD cruise controller. The desired and actual vehicle speeds for three different road grades as well as the gas pedal rates of the POD controller are illustrated in Figure 6. The small discrepancy between the desired and actual vehicle speed before the cruise controller activation is due to the steady-state error of the driver's model. However, since the desired driving cycle set the vehicle's speed to be at 40mph, when the POD cruise controller is activated helps to correct this error and, afterwards, maintains the vehicle's actual speed at the set point. The accelerator pedal position is at different values because in the case of road grades 2º and 6º the selected transmission gear is 2, shown in Figure 7, while in case of road grade 10º the selected transmission gear is 1. So, at different selected gears, the accelerator pedal varies to maintain constant vehicle's speed. In Figure 8, the performance of POD cruise controller is evaluated in a severe driving scenario where the road grade changes from 0° to 10°, while the POD cruise controller is active. In this scenario, the POD is activated again at 10sec when the road grade is 0°, and at 14 sec the road grade becomes 10°. The engine speed and the selected transmission gear for this scenario are shown in Figure 9. While the vehicle is cruising at constant speed and the road grade changes from 0º to 10º, the vehicle's speed starts decreasing after some time. Once this occurs, the self-learning cruise controller senses the discrepancy between the desired and actual vehicle speed and commands the accelerator pedal so as to correct the error. Consequently, there is a small time delay in the acceleration pedal command, illustrated in Figure 8, which depends on vehicle inertia.

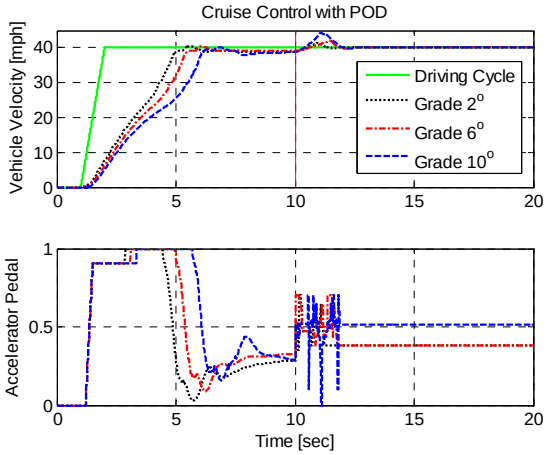

Figure 6. Vehicle speed and accelerator pedal rate for different road grades by self-learning cruise control with POD.

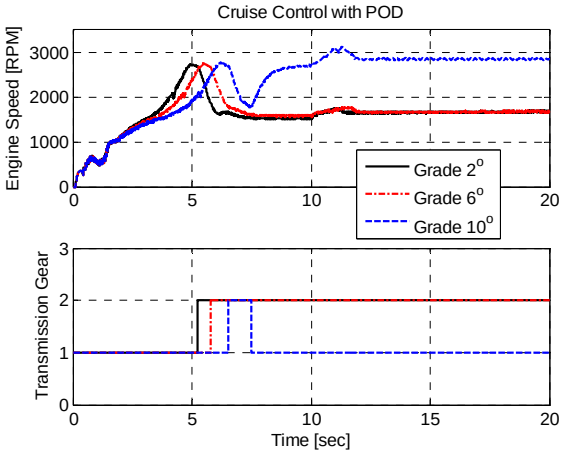

Figure 7. Engine speed and transmission gear selection for different road grades by self-learning cruise control with POD.

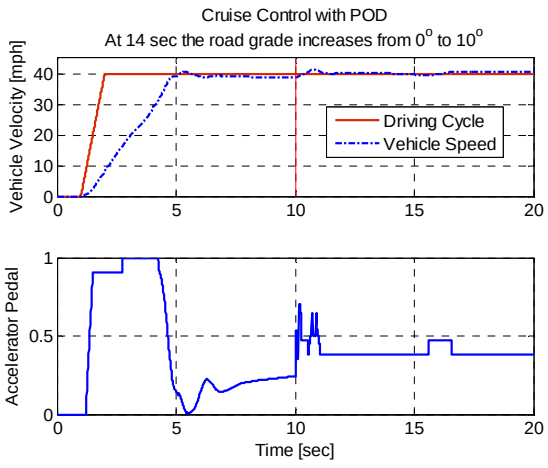

Figure 8. Vehicle speed and accelerator pedal rate for a road grade increase from 0° to 10°.

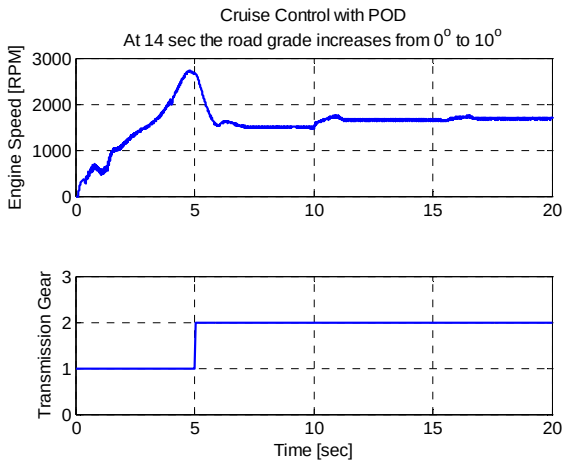

Figure 9. Engine speed and transmission gear selection for a road grade increase from 0° to 10°.

5. CONCLUDING REMARKS

This paper has presented a real-time learning method, consisting of a new state-space representation model and a learning algorithm, suited to solve sequential decision-making problems under uncertainty in real time. The method aims to build autonomous engineering systems that can learn to improve their performance over time in stochastic environments. Such systems must be able to sense their environments, estimate the consequences of their actions, and learn in real time the course of action (control policy) that optimizes a specified objective. The major difference between the POD learning method presented in this paper and the existing reinforcement learning methods is that the first employs an evaluation function which does not require recursive iterations to approximate the Bellman equation. This approach has been shown to be especially appealing to systems in which the initial state is not fixed, and recursive updates of the evaluation functions to approximate the Bellman equation would demand significant amount of experience to achieve the desired system performance.

The overall performance of the POD method in deriving an optimal control policy was evaluated by its application to two examples: (a) the cart-pole balancing problem, and (b) a real-time vehicle cruise-controller development. In the first problem, POD realized the balancing control policy for an inverted pendulum when the pendulum was released from any angle between 3° and -3°. In implementing the real-time cruise controller, the POD was able to maintain the desired vehicle's speed at any road grade between 0° and 10°. Future research should investigate the potential of advancing the POD method to accommodate more than one decision maker in sequential decision-making problems under uncertainty, known as multi-agent systems [39]. These problems are found in systems in which many intelligent decision makers (agents) interact with each other. The agents are considered to be autonomous entities. Their interactions can be either cooperative or selfish, i.e., the agents can share a common goal, e.g., control of vehicles operating in platoons to improve throughput on congested highways by allowing groups of vehicles to travel together in a tightly spaced platoon at high speeds. Alternatively, the agents can pursue their own interests.

ACKNOWLEDGMENTS

This research was partially supported by the Automotive Research Center (ARC), a U.S. Army Center of Excellence in Modeling and Simulation of Ground Vehicles at the University of Michigan. The engine simulation package enDYNA was provided by TESIS DYNAware GmbH. This support is gratefully acknowledged.

REFERENCES

[1] Bertsekas, D. P. and Shreve, S. E., *Stochastic Optimal Control: The Discrete-Time Case*, 1st edition, Athena Scientific, February 2007.

[2] Gosavi, A., "Reinforcement Learning for Long-Run Average Cost," *European Journal of Operational Research*, vol. 155, pp. 654-74, 2004.

[3] Bertsekas, D. P. and Tsitsiklis, J. N., *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3)*, 1st edition, Athena Scientific, May 1996.

[4] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, The MIT Press, March 1998.

[5] Borkar, V. S., "A Learning Algorithm for Discrete-Time Stochastic Control," *Probability in the Engineering and Information Sciences*, vol. 14, pp. 243-258, 2000.

[6] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210-229, 1959.

[7] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers. II -Recent progress," *IBM Journal of Research and Development*, vol. 11, pp. 601-617, 1967.

[8] Sutton, R. S., Temporal Credit Assignment in Reinforcement Learning, PhD Thesis, University of Massachusetts, Amherst, MA, 1984.

[9] Sutton, R. S., "Learning to Predict by the Methods of Temporal Difference," *Machine Learning*, vol. 3, pp. 9-44, 1988.

[10] Watkins, C. J., Learning from Delayed Rewards, PhD Thesis, Kings College, Cambridge, England, May 1989.

[11] Watkins, C. J. C. H. and Dayan, P., "Q-learning," *Machine Learning*, vol. 8, pp. 279-92, 1992.

[12] Kaelbling, L. P., Littman, M. L., and Moore, A. W., "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4.

[13] Schwartz, A., "A Reinforcement Learning Method for Maximizing Undiscounted Rewards," Proceedings of the Tenth International Conference on Machine Learning, pp. 298-305, Amherst, Massachusetts, 1993,

[14] Mahadevan, S., "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results," *Machine Learning*, vol. 22, pp. 159-195, 1996.

[15] Sutton, R. S., "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming," Proceedings of the Seventh International Conference on Machine Learning, pp. 216-224, Austin, TX, USA, 1990,
[16] Sutton, R. S., "Planning by Incremental Dynamic Programming," Proceedings of the Eighth International Workshop on Machine Learning (ML91), pp. 353-357, Evanston, IL, USA, 1991,
[17] Moore, A. W. and Atkinson, C. G., "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time," *Machine Learning*, vol. 13, pp. 103-30, 1993.
[18] Peng, J. and Williams, R. J., "Efficient Learning and Planning Within the Dyna Framework," Proceedings of the IEEE International Conference on Neural Networks (Cat. No.93CH3274-8), pp. 168-74, San Francisco, CA, USA, 1993,
[19] Barto, A. G., Bradtke, S. J., and Singh, S. P., "Learning to Act Using Real-Time Dynamic Programming," *Artificial Intelligence*, vol. 72, pp. 81-138, 1995.
[20] Bellman, R., *Dynamic Programming*. Princeton, NJ, Princeton University Press, 1957.
[21] Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 2nd Rev. edition, Wiley-Interscience, 2005.
[22] Bertsekas, D. P., *Dynamic Programming and Optimal Control (Volumes 1 and 2)*, Athena Scientific, September 2001.
[23] Malikopoulos, A. A., Papalambros, P. Y., and Assanis, D. N., "A Learning Algorithm for Optimal Internal Combustion Engine Calibration in Real Time," to be presented in the ASME 2007 International Design Engineering Technical Conferences Computers and Information in Engineering Conference, Las Vegas, Nevada, September 4-7, 2007, DETC2007-34718.
[24] Malikopoulos, A. A., Assanis, D. N., and Papalambros, P. Y., "Real-Time, Self-Learning Optimization of Diesel Engine Calibration," to be presented in the 2007 Fall Technical Conference of the ASME Internal Combustion Engine Division, Charleston, South Carolina, October 14-17, 2007, ICEF2007-1603.
[25] Iwata, K., Ito, N., Yamauchi, K., and Ishii, N., "Combining Exploitation-Based and Exploration-Based Approach in Reinforcement Learning," Proceedings of the Intelligent Data Engineering and Automated - IDEAL 2000, pp. 326-31, Hong Kong, China, 2000,
[26] Ishii, S., Yoshida, W., and Yoshimoto, J., "Control of Exploitation-Exploration Meta-Parameter in Reinforcement Learning," *Journal of Neural Networks*, vol. 15, pp. 665-87, 2002.
[27] Chan-Geon, P. and Sung-Bong, Y., "Implementation of the Agent Using Universal On-Line Q-learning by Balancing Exploration and Exploitation in Reinforcement Learning," *Journal of KISS: Software and Applications*, vol. 30, pp. 672-80, 2003.
[28] Miyazaki, K. and Yamamura, M., "Marco Polo: A Reinforcement Learning System Considering Tradeoff Exploitation and Exploration under Markovian Environments," *Journal of Japanese Society for Artificial Intelligence*, vol. 12, pp. 78-89, 1997.
[29] Hernandez-Aguirre, A., Buckles, B. P., and Martinez-Alcantara, A., "The Probably Approximately Correct (PAC) Population Size of a Genetic Algorithm," Proceedings of the 12th IEEE Internationals Conference on Tools with Artificial Intelligence, pp. 199-202, Vancouver, BC, Canada, 2000,
[30] Anderson, C. W., "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control Systems Magazine*, vol. 9, pp. 31-7, 1989.
[31] Williams, V. and Matsuoka, K., "Learning to Balance the Inverted Pendulum Using Neural Networks," Proceedings of the 1991 IEEE International Joint Conference on Neural Networks (Cat. No.91CH3065-0), pp. 214-19, Singapore, 1991,
[32] Zhidong, D., Zaixing, Z., and Peifa, J., "A Neural-Fuzzy BOXES Control System with Reinforcement Learning and its Applications to Inverted Pendulum," Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century (Cat. No.95CH3576-7), pp. 1250-4, Vancouver, BC, Canada, 1995,
[33] Jeen-Shing, W. and McLaren, R., "A Modified Defuzzifier for Control of the Inverted Pendulum Using Learning," Proceedings of the 1997 Annual Meeting of the North American Fuzzy Information Processing Society - NAFIPS (Cat. No.97TH8297), pp. 118-23, Syracuse, NY, USA, 1997,
[34] Mustapha, S. M. and Lachiver, G., "A Modified Actor-Critic Reinforcement Learning Algorithm," Proceedings of the 2000 Canadian Conference on Electrical and Computer Engineering, pp. 605-9, Halifax, NS, Canada, 2000,
[35] Si, J. and Wang, Y. T., "On-line Learning Control by Association and Reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, pp. 264-276, 2001.
[36] Zhang, B. S., Leigh, I., and Leigh, J. R., "Learning Control Based on Pattern Recognition Applied to Vehicle Cruise Control Systems," Proceedings of the the American Control Conference, pp. 3101-3105, Seattle, WA, USA, 1995,
[37] Shahdi, S. A. and Shouraki, S. B., "Use of Active Learning Method to Develop an Intelligent Stop and Go Cruise Control," Proceedings of the the IASTED International Conference on Intelligent Systems and Control, pp. 87-90, Salzburg, Austria, 2003,
[38] TESIS, <http://www.tesis.de/en/>.
[39] Panait, L. and Luke, S., "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387-434, 2005.