Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 1015 – 1034

doi:10.3906/elk-1106-40

Actor-critic-based ink drop spread as an intelligent controller

Hesam SAGHA,* Iman Esmaili Paeen AFRAKOTI, Saeed BAGHERISHOURAKI
Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran

Received: 25.06.2011 • **Accepted:** 06.01.2012 • **Published Online:** 03.06.2013 • **Printed:** 24.06.2013

Abstract: This paper introduces an innovative adaptive controller based on the actor-critic method. The proposed approach employs the ink drop spread (IDS) method as its main engine. The IDS method is a new trend in soft-computing approaches that is a universal fuzzy modeling technique and has been also used as a supervised controller. Its process is very similar to the processing system of the human brain. The proposed actor-critic method uses an IDS structure as an actor and a 2-dimensional plane, representing control variable states, as a critic that estimates the lifetime goodness of each state. This method is fast, simple, and away from mathematical complexity. The proposed method uses the temporal differences (TD) method to update both the actor and the critic. Our system: 1) learns to produce real-valued control actions in a continuous space without regarding the Markov decision process, 2) can adaptively improve performance during the lifetime, and 3) can scale well to high-dimensional problems. To show the effectiveness of the method, we conduct experiments on 3 systems: an inverted pendulum, a ball and beam, and a 2-wheel balancing robot. In each of these systems, the method converges to a pertinent fuzzy system with a significant improvement in terms of the rise time and overshoot compared to other fuzzy controllers.

Key words: Active learning method, actor-critic method, adaptive control, fuzzy inference system, ink drop spread, reinforcement learning

1. Introduction

Although man-made machines are trying to be perfect in their environments, they may not adapt themselves with dynamics easily. This difficulty is the motivation for designing adaptive controllers. There are 2 categories of this kind of controller: the model-based controllers, which need an explicit model of the plant under control, and model-free controllers, which do not need this model. Although system models may ease the controlling process, they are mostly approximated via linearization, relaxing variables, or ignoring variables. Even in nonlinear control methods like back-stepping and sliding mode control, it is mostly difficult or even impossible to model a complex system. To overcome these problems we should migrate from model-based controllers to model-free intelligent controllers.

The prominent control strategies of intelligent controllers are based on learning, and especially reinforcement learning (RL) methods [1].

Beyond the agent and the environment, we can identify 4 main subelements of a RL system: a policy, a reward function, a value function, and, optionally, a model of the environment. A reward function defines the goal in a RL problem. A policy defines the learning agent's way of behaving at a given time. A RL agent's sole objective is to maximize the total reward it receives in the long term [2]. The reward function scores the actions

*Correspondence: hesam.sagha@epfl.ch

done by the agent, explaining how promising they are to achieve the goal. They are the immediate and defining features of the problem faced by the agent. As such, the reward function must necessarily be unalterable by the agent. It may, however, serve as a basis for altering the policy. For example, if an action selected by the policy is followed by a low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward functions may be stochastic [3].

Mostly, the description of the environment states and the actions of the agent are discrete in conventional RL methods by assuming the task as a Markov decision process (MDP), with a definite transition probability for all of the actions at each state. Real-world problems that are large-scale or have continuous spaces are a challenge for RL methods. In these cases, an admissible solution is to reduce the continuous states into numerable discrete ones. Nonetheless, this can cause the curse of the dimensionality problem. Another solution is to use a generalization function to approximate a mapping from a state space into an action space for an unseen state.

Value function and policy function approximation are 2 well-known approaches in the generalization of RL. Algorithms like Q-learning, temporal differences (TD), and state-action-reward-state-action (SARSA) learning all estimate the value function of the MDP. Barto et al. [3] proposed an adaptive heuristic critic (AHC) algorithm, which approximates the value and policy functions simultaneously under the actor-critic architecture. In the actor-critic algorithm, the actor approximates the policy function and the critic predicts the approximation of the value function. Actor-critic methods are on-policy temporal-difference–based methods that have a separate memory structure to explicitly represent the policy independent of the value function.

In [4], an original variable gain proportional-integral (VGPI) controller was presented for speed control of a direct torque neuro fuzzy-controlled (DTNFC) induction motor drive. The results showed the effectiveness of the method for solving speed control problems of an induction motor as an intelligence control.

Our proposed actor-critic RL agent uses an ink drop spread (IDS) method [5] as an actor. This method has been proposed as a new approach to soft computing. The IDS method is a fuzzy modeling technique in the framework of the active learning method (ALM) [5–7]. The main idea of the ALM is to approximate a multiple inputs-single output system with several single input-single output subsystems. Each subsystem shows the behavior of an output with respect to one input. This kind of behavior is depicted on a plane (called the IDS plane or IDS unit).

The IDS method applies a 2-dimensional (2D) membership function on each datum to diffuse information in the problem space and combines the diffused data to obtain a narrow path on each IDS plane. The IDS method does not store data in the process of learning, like instance-based learning methods [5], but estimates it via a representation of a narrow path that has a short constant memory usage.

The ALM was used as a supervised controller in [5–8]. It creates the model of the plant under control during the operation of the system and uses the inverse model of the plant to control it simultaneously. The ALM's hardware implementation was described in [9] and other versions were also proposed in [10] as supervised and unsupervised controllers. Sakurai [11] proposed an ALM-based nonlinear modeling method that can be applied to a learning controller and verifies the effectiveness of this method even for a plant with nonholonomic characteristics. In [12], the ability of the IDS method in modeling was investigated. In this paper, we propose a control algorithm based on the IDS modeling and RL learning algorithms. Actually, IDS is 1 of the 2 significant engines of the proposed algorithm.

The rest of this paper is organized as follows: we discuss RL and actor-critic methods extensively in Section 2. The concept of the ALM and the IDS method and its mathematics are provided in Section 3 and the

proposed method is explained in Section 4. It uses an actor-critic scheme to model the best actions on the IDS planes. The actor is an IDS system and the critic is a plane that stores the goodness of the control variable for all of the possible states in a compact way. Finally, the results of the algorithm implemented on 3 simulated models are provided in Section 5.

2. Reinforcement learning

There are 3 classes of machine learning methods: supervised, unsupervised, and RL.

In supervised learning, a teacher provides the desired control objective to the learning system at each time step. However, in unsupervised learning, a bunch of data enter the system and the system must cluster them into some classes. In the case of RL, the teacher response is not as direct, immediate, and informative as in supervised learning. The system must learn to do its best through observations and rewards.

A RL agent observes a state and recommends an action. The environment rewards the action and the RL agent tries to maximize this value over time. Figure 1 displays the interaction between the RL agent and the environment, where β is the reward that the environment gives to the system, α is the system's stimulation to the environment, and $s(t)$ is the state of the environment at time t.

Among the proposed RL methods, the prominent ones are dynamic programming (DP) [13], Monte Carlo (MC), and TD-based methods [14], e.g., Q-learning, SARSA, and actor-critic methods [14]. DP methods use the exact model of the environment as a MDP to obtain the optimal policy. MC methods do not assume the existence of an exact model of the environment and just act based on experiments represented as state, action, and reward triples. In MC methods, it is necessary to finish each episode to update the estimates. TD methods are combinations of DP and MC methods. Similar to MC methods, TD methods use experiments without an exact definition of the environment, and, similar to DP methods, they update the estimates based on other estimates without the need to finish an episode.

The difference between Q-learning and the SARSA is in their action selection policy. Q-learning is off-policy, i.e. the selected action may be different from what is estimated. However, the SARSA is on-policy, which means that it improves the recently done action. Actor-critic methods are on-policy methods that use separate memory structures to represent the explicit policy independent of the value function.

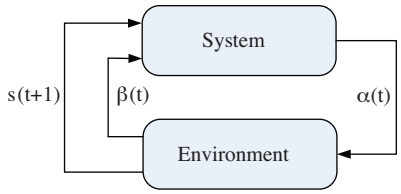

Figure 1. Schematic of implementing a RL agent in an environment: β is the reward, α is the action that the system recommends, and $s(t)$ is the state at time t.

2.1. Temporal difference

TD learning is a predictive method. It assumes that consecutive predictions are correlated in some sense. In predictive learning, the process of learning is done by observing the most recent reaction; a prediction is made, and when an observation is done, the prediction parameters are changed to be more adapted to the observation. The main idea of TD is to adjust predictions to be more adapted in the future. TD updates the state value by:

$$\bar{V}_t = \lambda_t + \gamma \bar{V}_{t+1}, \tag{1}$$

where λ_t is the reinforcement at time t, γ is the discounting factor, and $\bar{V}_t$ is the predicted sum of the discounted rewards at time t. Note that with $\lambda = 1$, TD resembles the MC algorithm.

2.2. Actor-critic method

Actor-critic methods are TD processes that have separate memory structures for policy and value functions. The policy structure is called "actor", because it selects an action. The value estimator function is called "critic" because it criticizes the recently done action. Learning is on-policy, i.e. the critic must learn about and criticize whatever policy is currently being followed by the actor. The critique takes the form of a TD error. This scalar signal is the sole output of the critic that drives all of the learning in both the actor and critic, as shown in Figure 2.

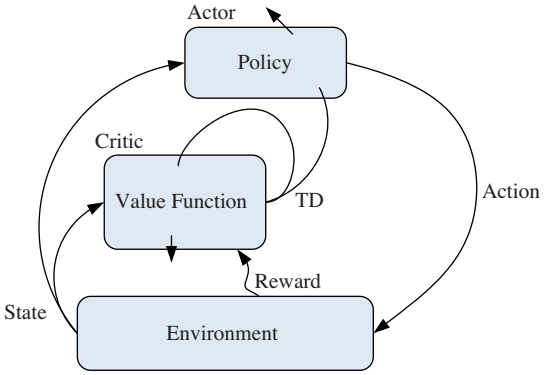

Figure 2. Schematic of an actor-critic–based system

Although actor-critic methods have received less attention in comparison to methods that learn action-value functions and determine a policy exclusively from the estimated values (such as SARSA and Q-learning), they have 2 advantages over these methods:

- Actor-critic methods require minimal computation in order to select actions and they consider a continuous-valued action or continuous environment.
- Actor-critic methods can learn explicit stochastic policy; that is, they can learn the optimal probabilities of selecting various actions. This ability turns out to be useful in competitive and non-Markov cases.

In the actor-critic method, a value is assigned to each state (which can be estimated) and then, by observing the reward or the penalty of an action, the last state value is verged to the current state value. Finally, the actor updates itself according to these values. In fact, this is not supervised learning, because the critic (supervisor) itself is being formed during the working procedure of the system. The value of each state relates to the reward and the next state value according to:

$$\delta_t = R_{t+1} + \lambda V(s_{t+1}) - V(s_t)$$

$$V(s_t) = V(s_t) + \alpha\delta_t$$

$$P(s_{t+1}, a_{t+1}) = P(s_t, a_t) + \beta\delta_t, \tag{2}$$

where δ_t is the TD error, $V(s_t)$ is the state value of state s_t, R_{t+1} is the reward of arriving at state s_{t+1}, and $P(s_t, a_t)$ is the value of the policy at time t, which represents the tendency to select action a at state s. The action a is conducted at time t. α and β are constants and $\lambda$is the discounting factor.

Various fuzzy actor-critic RL methods have been proposed. Generalized approximate-reasoning–based intelligent control (GARIC) [15] uses a neural network as the critic. The fuzzy actor-critic reinforcement learning network (FACRLN) [16] is realized by a 4-layer fuzzy radial basis function neural network that is used to approximate both the action-value function of the actor and the state-value function of the critic simultaneously. Continuous actor-critic learning-automaton (CACLA) [17] can handle continuous states and actions. QV (lambda)-learning and the actor-critic learning automaton (ACLA) [18] both use the TD (lambda)-method as the state-value function learning process.

Due to the existence of similarities between our proposed algorithm architecture and the GARIC architecture, we describe it in the following subsection briefly.

2.3. GARIC

The GARIC architecture is shown in Figure 3.

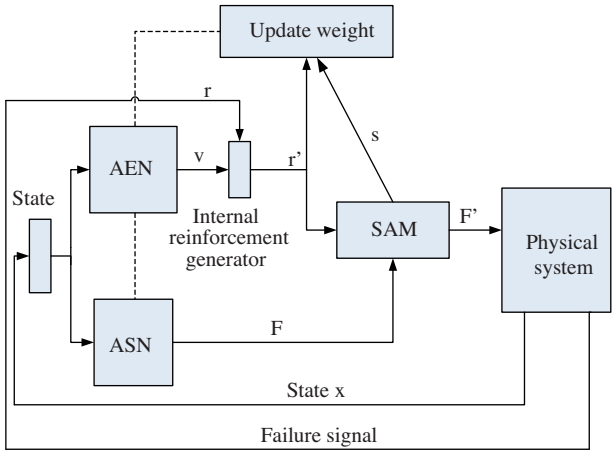

Figure 3. GARIC architecture.

It has 3 main parts:

I. Action evaluation network (AEN): This unit acts as the critic. It evaluates the goodness of a state. Moreover, it produces internal reinforcement r'. The AEN can be a neural network [15], fuzzy inference system, or other evaluators.

II. Action selection network (ASN): It maps the current state into an action F. This is a fuzzy inference system.

III. Stochastic action modifier (SAM): It uses F and r' to produce action F', which will be applied to the environment. $F^{'}$ is a Gaussian stochastic variable with the average F and the variance $\sigma(r^{'}(t-1))$. $\sigma(.)$ is a monotonically decreasing and nonnegative function like $exp\left(-r^{'}\right)$. This means that if the internal reinforcement is high, the recommended action will be less modified and we must rely on the ASN more. The SAM produces perturbation s, which is a measure for the magnitude of the difference between F and F'. The SAM provides the ability of exploration (getting new knowledge) and exploitation (using previous knowledge).

3. Active learning method

The ALM is a recursive fuzzy modeling technique that avoids mathematical complexity. It converts a multiple inputs-single output system into several single input-single output subsystems. The flowchart of the ALM is shown in Figure 4. As stated in [11], the ALM was developed based on the following set of hypotheses.

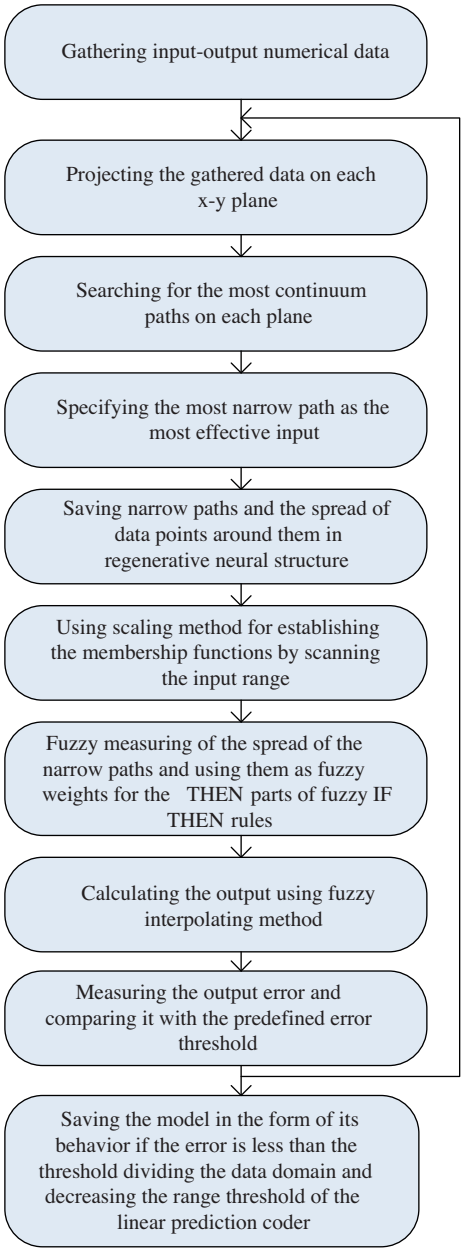

Figure 4. Flowchart of the ALM.

When humans model a complex system, they carry out the following steps:

1. They derive the system features by breaking down the system into simpler aspects and translating the information into a more readily comprehensible form.

2. The information obtained at this stage is of an outline nature and is in the form of images.
3. They link multiple images together and strive to obtain an understanding of the entire system.
4. If the information is inadequate, an effort is made to acquire additional information from specific parts of the system by active manipulation. This process is repeated through trial and error.

Each subsystem is a plane (called the IDS plane or IDS unit [12]) that depicts the output behavior with respect to one input. The behavior is obtained by the IDS engine. The IDS method diffuses the data information on each plane and extracts a behavior by combining that information. The IDS method is explained later in this section. To get the final output of the system, which can be a function output, the model output, or an input to a control system, the IDS method combines all of the behaviors using the weighted sum. For a 2-input system, where each input is broken into 2 membership functions (partitions), there would be 4 rules. The consequence part of each rule is demonstrated by an IDS plane. Its inference process is shown in Figure 5.

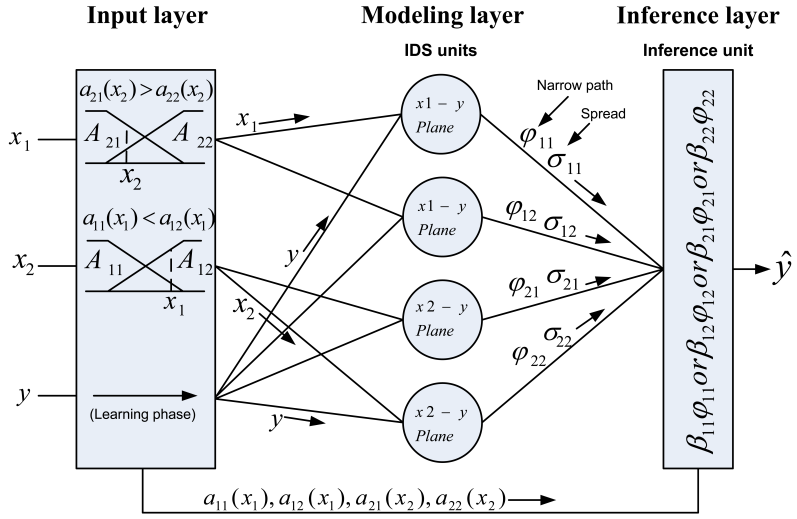

Figure 5. Diagram of a 2-input system; each input is broken into 2 membership functions (courtesy of Murakami [12]).

The IDS planes show 2 kinds of information: a) the narrow path that depicts the output behavior with respect to 1 input and b) the variance of data around each narrow path, which illustrates the efficacy of the relevant input on the output.

When a datum enters the ALM, for each rule, first the data dependency to each membership function of that rule will be obtained and then the degree of belief for that rule will be determined regarding the fuzzy intersection of these values. Up to now, the inference process is like what Takagi–Sugeno and Mamdani inference systems do. The consequent part is different, such that instead of a linear combination of inputs or a membership function, the information on the IDS plane is used.

To combine the behaviors, they can be useful as a weighted union:

$$y \text{ is } \beta_{11}\varphi_{11} \text{ or } \ldots \text{ or } \beta_{ik}\varphi_{ik} \text{ or } \ldots \text{ or } \beta_{Nl_N}\varphi_{Nl_N}, \tag{3}$$

where y is the output, *or* is the union operator, and β_{ij} is the weight of the jth narrow line of the ith input, which has an inverse relation to the variance of the data around the narrow path, i.e. while the variance of the data increases, the efficacy decreases.

If we have an IDS system with N inputs, $X_1 \dots X_N$, and each input is partitioned into $m_1 \dots m_N$ membership functions, the number of IDS units, L, which is equivalent to the number of rules, is:

$$L = \sum_{i=1}^{N} l_i, \tag{4}$$

wherel_i is the number of IDS units for the ith input variable, which is obtained by:

$$l_i = \prod_{i'=1}^{N} m_{i'}, i' \neq i \tag{5}$$

The rules are:

$$\text{if } x_2 \text{ is } A_{2j_2} \wedge x_3 \text{ is } A_{3j_3} \wedge \dots x_N \text{ is } A_{Nj_N}, \text{ Then is } \phi_{1k} i = 1$$

$$\text{if } x_1 \text{ is } A_{1j_1} \wedge \dots x_{i'} \text{ is } A_{i'j_i} \wedge \dots x_N is A_{Nj_N}, \text{ Then is } \phi_{ik} i' \neq i, 1 < i < N$$

$$\text{if } x_1 \text{ is } A_{1j_1} \wedge x_2 \text{ is } A_{2j_2} \wedge \dots x_{N-1} \text{ is } A_{N-1j_{N-1}},$$

$$\text{Then is } \phi_{Nk} i = N, 1 \leq j_1 \leq m_1, 1 \leq j_2 \leq m_2, \dots, 1 \leq j_N \leq m_N, \tag{6}$$

where x_i is the ith input, A_{kj} is the jth membership function of the kth input, $\wedge$ is the and operator, and φ_{ik} is the estimated narrow path for the kth IDS unit of the ith input. To store narrow paths, a look-up table, regenerative neural network [18], or fuzzy curve expresser can be used.

Now we are going to describe the IDS method. As can be inferred from the name, the diffusion of information has the main rule in this concept, i.e. by observing a data sample as a point in the problem space, not only can we extract the information at that point, but we can also elicit the information around that data point in the problem space. By getting away from that point, we have less confidence about the extracted knowledge. It can be done by putting a 2-dimensional pyramid-shape fuzzy membership function on each datum.

This kind of membership function gives the full trust for a datum and it assumes lower trusts for its neighbors. The value of the overlapped area is the summation of the degrees of trusts for that area (Figure 6). Figure 7 shows an IDS plane on which 5 pieces of data have been diffused.

To extract the behavior of each input with respect to the output, the IDS method combines the information on the inputs. This is done by calculating the center of gravity (COG) of the IDS plane on each perpendicular path to the input axis. The output of the combination is a narrow path that shows the behavior of data on that plane.

As an example, Figure 8 shows a 2-input function. Projecting data on each $x_1 - y$ plane results in Figure 9a and on the $x_2 - y$ plane results in Figure 9b. By diffusing the data and getting the COG on each vertical line of an IDS plane, a narrow path would be extracted. Figure 10a shows the diffused data on the $x_2 - y$ plane and the extracted narrow path, and the spread is shown in Figure 10b.

This path shows the behavior of y with respect to x_2. Furthermore, other information can be obtained in this plane. As can be seen on the left side of this IDS plane, the variance of the data is very low. It illustrates that the input variable x_2 has more effect on the output than x_1 when x_2 is small.

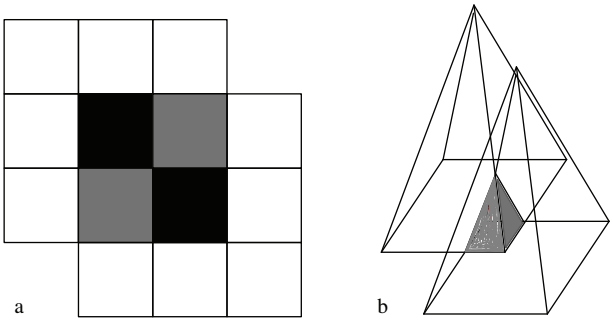

Figure 6. IDS and 2D fuzzy membership functions.

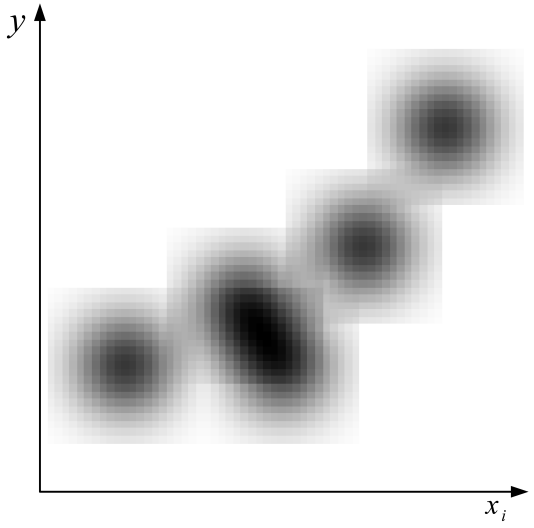

Figure 7. The IDS plane after spreading 5 data on it.

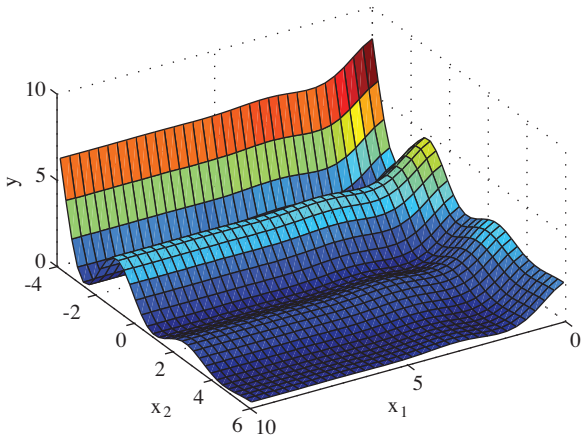

Figure 8. A 2-input, 1-output function surface.

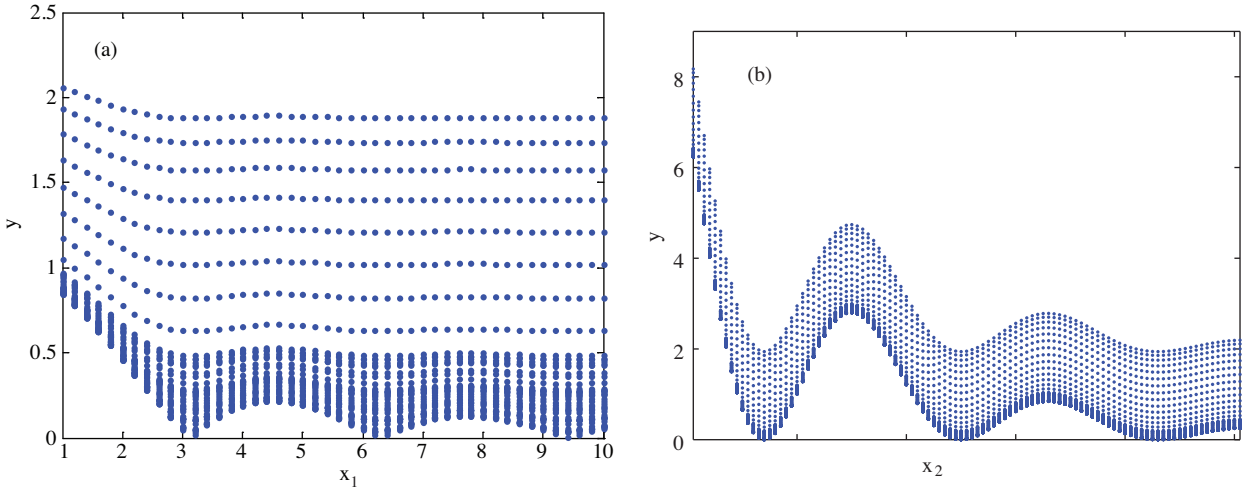

Figure 9. Projected data on 2 IDS planes: a) $x_1 - y$ plane and b) $x_2 - y$ plane.

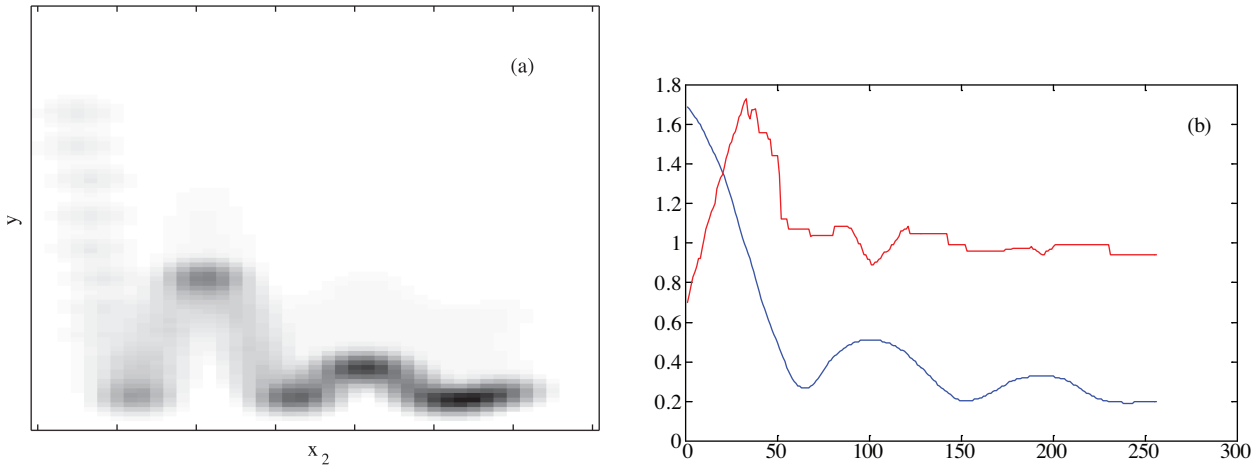

Figure 10. a) Diffused data on the $x_2 - y$ plane and b) extracted narrow path (blue line) and spread (red line).

The IDS method is actually a concept. It can be implemented in many ways. This section explains the mathematical implementation of the IDS method that was proposed by Murakami and Honda [12]. To better match our new method and to be more robust against noise, some formulas have been changed.

Assume an $N-$inputs, 1-output system, such that $(X_1, X_2 \ldots, X_N) \rightarrow Y$; a point on the $X_i - Y$ plane is $p(x,y)\, x \in X_i y \in Y$; and $d(x,y)$ depicts the brightness of $p(x,y)$. When diffusion is done on point $p(x_s, y_s)$, the brightness of its neighbors is increased similar to an ink drop. Here, an ink drop is a 2D Gaussian function with a variance proportional to the spread of the ink drop. In the center, the drop has a maximum value. The drop in 2 sizes, 5 and 9, is shown in Figures 11a and 11b, respectively. Experimentally, the variance is set to 15% of the ink spread.

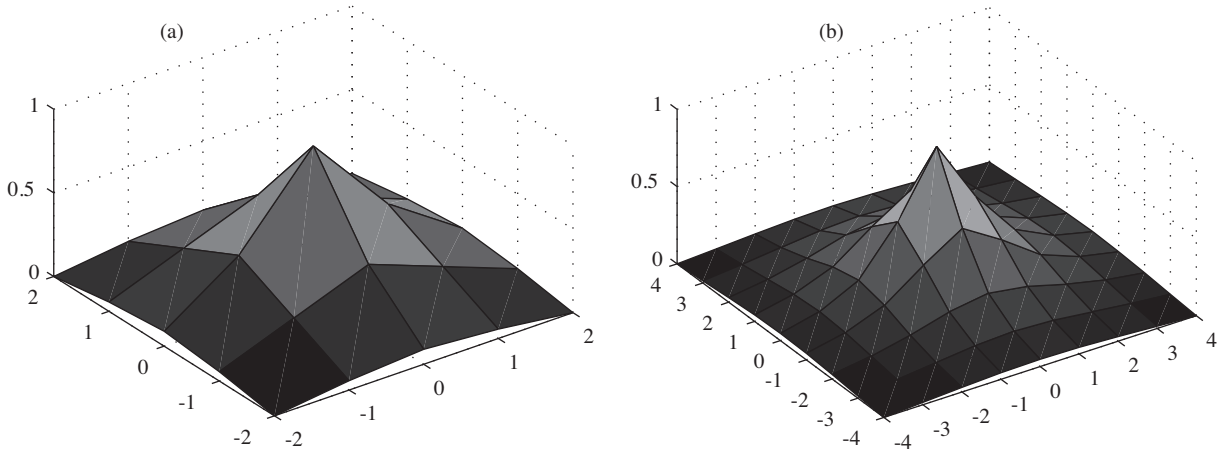

Figure 11. Gaussian ink drop of size 5 (a) and 9 (b).

There are different methods for extracting narrow paths. One is computing the COG of the data on each path perpendicular to the input axis. In this case, the COG for the point x_i is defined by:

$$\varphi(x_i) = COG_i = \sum_y y d(x_i, y) \Big/ \sum_y d(x_i, y). \tag{7}$$

Another method is a rectilinear move on each vertical line from both sides simultaneously. The selected point is where the cumulative sums of the values on each move for both directions are equal. It is actually the bisector of the area.

$$\varphi(x_i) = \left\{ b | \sum_{y=1}^{b} d(x_i,y) \approx \sum_{b}^{y_{\max}} d(x_i,y), b \in Y \right\} \tag{8}$$

We used a new method for obtaining the variance around a narrow path. Where the variance has a reciprocal relation with the value of $d(x_i, y)$ and has a straight relation with the distance from the narrow path, the following formula is used:

$$\sigma^{-1}(x_i) = \frac{\sum_{y} d(x_i,y) / (y - \varphi(x_i))}{\sum_{y} d(x_i,y)}. \tag{9}$$

To limit this value between 0 and 1, we used:

$$f(x_i) = 1 - \exp\left(-\alpha \sigma^{-1}(x_i)\right), \tag{10}$$

where α is set to be between 0.05 and 0.12 of the output range, experimentally.

The inference process is a combination of each IDS unit. This combination is the weighted sum of the narrow paths.

$$\beta_{ik} = \frac{\omega_{ik}\Gamma_{ik}}{\omega_{11}\Gamma_{11} + \ldots + \omega_{ik}\Gamma_{ik} + \ldots + \omega_{Nl_N}\Gamma_{Nl_N}}, \tag{11}$$

where

$$\begin{aligned}
\Gamma_{11} &= a_{21}(x_2) \bigwedge a_{31}(x_3) \bigwedge \ldots \bigwedge a_{Nl}(x_N) \\
\Gamma_{ij} &= a_{1j}(x_1) \bigwedge \ldots \bigwedge a_{i'j}(x_{i'}) \bigwedge \ldots \bigwedge a_{Nj}(x_N), i' \neq i \\
\Gamma_{Nl_N} &= a_{1l_N}(x_1) \bigwedge a_{2l_N}(x_2) \bigwedge \ldots \bigwedge a_{N-1l_N}(x_{N-1}),
\end{aligned} \tag{12}$$

where a_{ij} is the jth membership function of the ith input, $\bigwedge$ is the union operator, $\omega_{ik} = f_{ik}(x_i)$ illustrates the variance around the narrow path for the kth IDS unit of the ith input variable, and $f_{ik}(.)$ represents the conversion function, Eq. (7).

4. Proposed method

In this section, a new method called Reinforcement Learning Ink Drop Spread (RLIDS) is proposed, which uses the IDS method as an actor and a new plane called the reward-penalty plane (RPP) as a critic, the RPP surface demonstrating the goodness of the states. The RPP is a plane made up of the control variable. One axis is the variable (error) and the other is the change in the variable (change in the error). Each point on the plane illustrates how much we can trust in the recommended action of the IDS method. A stochastic action modifier (SAM) is used to change the action according to the output of critic.

Since the method is going to control a system, and we know that the final goal is to keep the control variable at a set point of 0, the RPP is divided into 3 parts:

i. Reward area: This is the area where we want the control variable to stay (around 0). The value of this area is 1.

ii. Penalty area: This is the undesirable area; the system is not supposed to be here and the system may be unstable.

iii. Play area: This is the other part of the RPP, which is set to 0 initially.

An initial RPP is shown in Figure 12, where the control variable is the angle of an inverted pendulum.

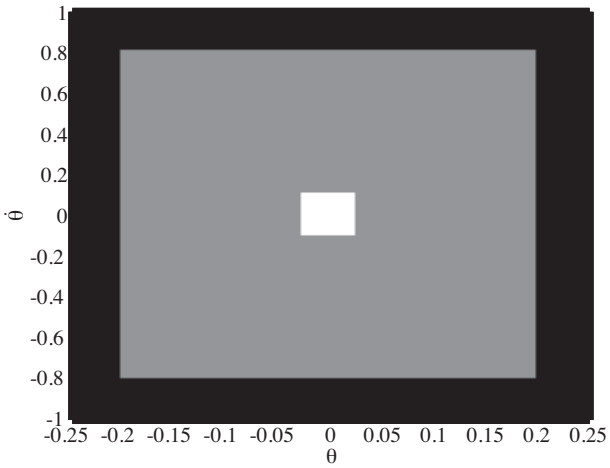

Figure 12. An initial RPP. The white area in the center has the value 1. The border areas have a negative value. The other area has a value of 0 initially.

In fact, since this is a control problem and the set point is 0, we embed the rewards and punishments into the critic part. If we do not know where we should locate the reward and the penalty areas, it is better to set the RPP to 0 in the all of the points and get the advantages of the external reinforcement.

During the run-time, when an observation is made, the value of the RPP for a state and its neighbors in the previous time step verge to the value of the RPP for the current state (as is done in TD).

$$delta{=}win\lambda\left(RPP\left(e\left(t\right),ce\left(t\right)\right)-RPP\left(e\left(t{-}1\right),ce\left(t{-}1\right)\right)\right) \tag{13}$$

$$Nei\left(RPP\left(e\left(t{-}1\right),ce\left(t{-}1\right)\right)\right){=}Nei\left(RPP\left(e\left(t{-}1\right),ce\left(t{-}1\right)\right)\right){+}delta \tag{14}$$

Here *delta* is the change in the RPP; $e(t)$ is the value of the control variable at time t; *ce* is the change in the error; *win* is the IDS window (ink), which demonstrates how many of the neighbors must be affected and by how much; λ is the reduction rate that is a constant between 0 and 1; and *Nei(s)* demonstrates the neighbors of the state position s on the RPP.

The IDS planes are updated using:

$$IDS_{ij}\left(in_i\left(t{-}1\right),out\left(t{-}1\right)\right){=}IDS_{ij}\left(in_i\left(t{-}1\right),out\left(t{-}1\right)\right){+}delta, \tag{15}$$

where IDS_{ij} is the jth IDS unit of the ith input and in_i is the ith input variable.

To apply an action to the environment and to not miss the exploration and exploitation capabilities, it is necessary to change the recommended action F. This is done by:

$$F^{'}\left(t\right){=}F\left(t\right){+}N\left(0,Var\right)\times\left(\exp\left(-\alpha{\times}RPP\left(e\left(t\right),ce\left(t\right)\right)\right)-\exp\left(-\alpha\right)\right), \tag{16}$$

where *N(0,Var)* is a normal distributed random variable with the variance Var and the average 0 (α is a constant). When the RPP scores a state, the best IDS-offered action, which is 1, will not be changed.

When the system lies in the reward area, it may have some oscillation. This is because it has not learned how to act in this area. To cope with this problem, a fuzzy scaling method is used. In this method, the input range of the variables is increased proportional to the reward area:

$$in_i^{'}=in_i\times\frac{Range\,(in_i)}{Range\,(Reward\,(in_i))}, \tag{17}$$

where in_i is the ith input variable, which constructs the RPP, and the output range decreases:

$$Out=\frac{Out}{Max_i\left(\frac{Range(in_i)}{Range(Reward(in_i))}\right)}. \tag{18}$$

Hence, there is no need for another fuzzy system, and it is sufficient to scale the input and the output ranges using the current IDS.

RLIDS has some advantages over other IDS-based controllers such as modified IDS, RIDS, and modified RIDS; it covers delayed rewards, the reward area is explicitly defined, and it is not necessary to define the trajectory to reach the goal.

5. Results

The experiments were conducted on 3 simulated systems: an inverted pendulum, a ball and beam, and a 2-wheeled balancing robot (TWBR). They are described in the following 3 subsections.

5.1. Inverted pendulum

In this model, a controller must apply a force to a cart on which an inverted pendulum is linked. Figure 13 shows this system. The model of the inverted pendulum is depicted below. The input variables are θ and the velocity of theta ($\dot{\theta}$). Since the proposed algorithm is designed to control one variable, the goal of the controller is to stabilize the pendulum regardless of the location of the cart.

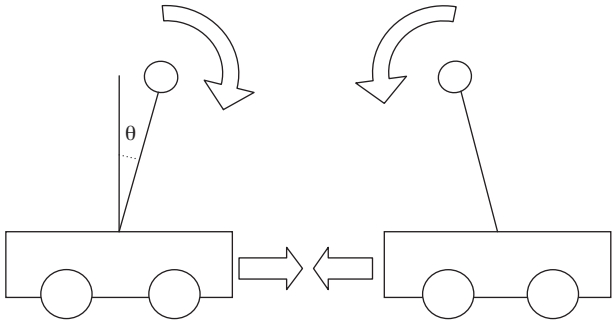

Figure 13. An inverted pendulum and force direction to the cart to stabilize a pendulum.

The dynamic model of the cart–pole system is described by the following nonlinear equations [19]:

$$x=\left[\theta,\dot{\theta}\right],$$

$$\dot{x} = f(x,u),$$

$$y = [x_1, x_2],$$

$$f(x,u) = \begin{bmatrix} x_2 \\ \dfrac{g\sin(x_1)+\cos(x_1)\left(\frac{-u-mlx_2^2\sin(x_1)}{m+M}\right)}{l\left(\frac{4}{3}-\frac{mcos(x_1^2)}{m+M}\right)} \end{bmatrix}, \tag{19}$$

where M and m are the mass of the cart and the pendulum, respectively. L is the length of the pendulum and u is the applied force to the cart. We set $M = 1$, $m = 0.1$, $l = 0.5$, and $g = 9.8$.

The operation (play) area is set to $\theta = [-0.23, 0.23]$ rad and $\dot{\theta} = [-0.98, 0.98]$ rad/s. The output range is [–25, 25]. The reward area is in 5% of the operation area and the penalty area is outside 90% of the operation area. The λ for the reward is 0.9 and for the penalties it is 0.05. The penalty area is set to –0.5 and the time step is set to 0.02. Each input is broken into 2 membership functions.

We applied our method in 2 ways: episodic and continuous. In episodic learning, when the system states lie in the reward area, it is reset to a new random state to begin the new episode. In continuous learning, there is no reset while the states lie in the reward area.

Episodic learning: Rewards and penalties during the learning process are shown in Figure 14 for one experiment. As is shown, the number of rewards grows and the number of punishments decreases. In this

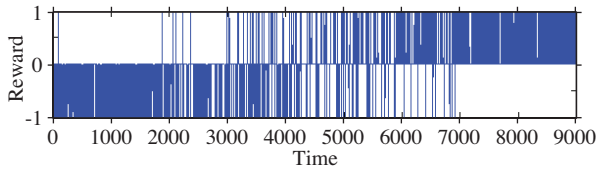

Figure 14. Rewards (1) and penalties (–1) during episodic learning for the inverted pendulum system.

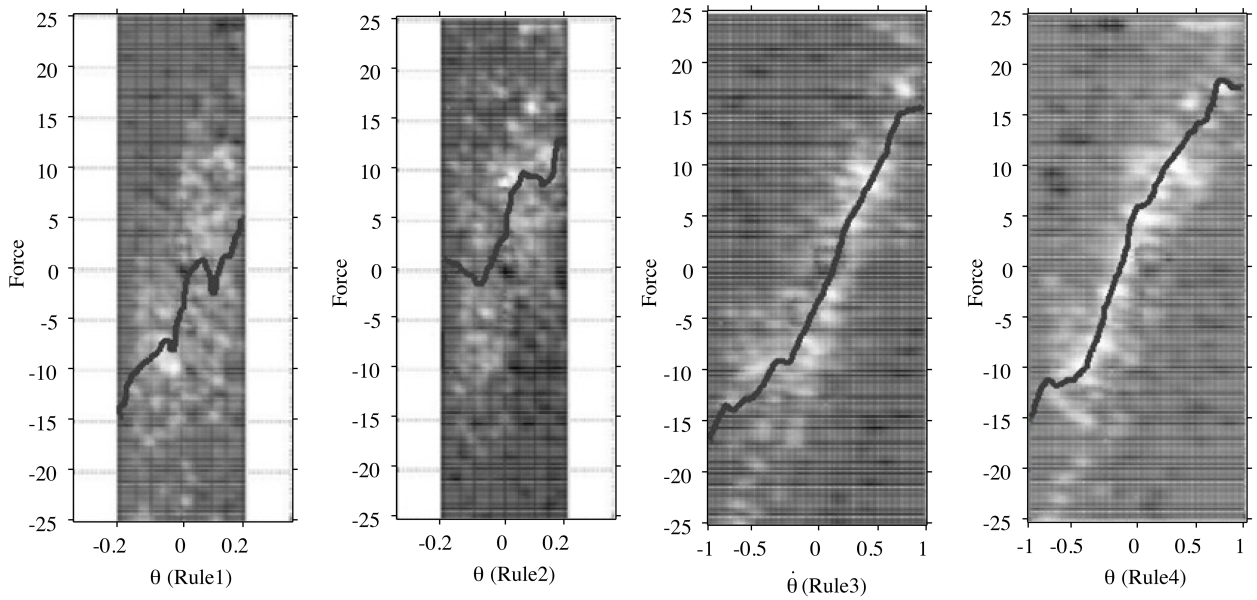

Figure 15. The IDS plane after 10,000 time steps. Dark paths are narrow paths.

experiment, when the system reached the penalty or reward area, it was reset to a new state to begin a new episode. Figure 15 shows the IDS planes after 10,000 time steps. The system states in the first 1000 time steps and after 7200 time steps are shown in Figures 16a and 16b, respectively.

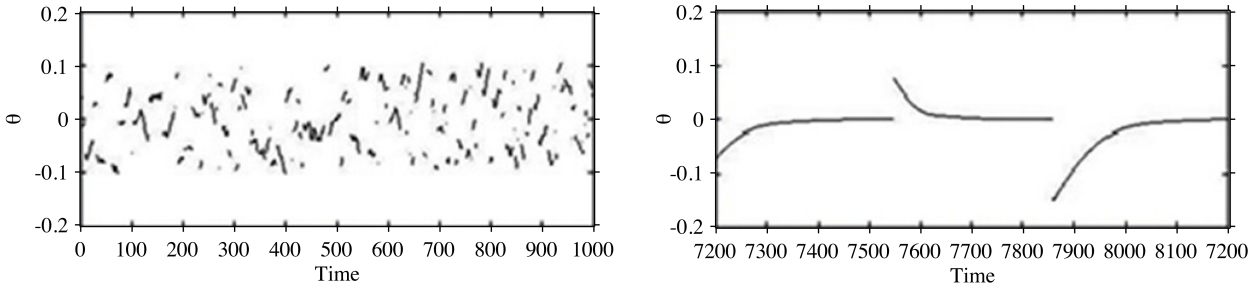

Figure 16. System states before and after learning: a) first 1000 time steps and b) time steps between 7200 and 8200.

Continuous learning: In this case, for 10 experiments, it takes an average of 83 trials to achieve the goal. Here, the goal is to not fall in an interval of 50,000 time steps. We ran the system 24 times. The average number of trials was 83 for 4 rules.

The convergence rate depends on how many times the system meets the reward area during the learning process. When the system does not meet the reward area, it cannot create pertinent narrow paths. Table 1 compares the proposed method with the other methods. Although the number of trials is more than some others, the least number of rules shows the advantages over the others.

Table 1. Comparing the proposed system (RLIDS) with the other RL systems that are provided in [15].

	Continuous states	Continuous actions	A priori knowledge	Number of fuzzy rules	Structure learning	Number of trials
AHC	No	No	No	-	No	35
Anderson	Yes	No	No	-	No	8000
GARIC	Yes	Yes	Yes	13	No	300
RNN-FLCS	Yes	Yes	Yes	35	Yes	10
FALCON-RL	Yes	Yes	Yes	10	Yes	15
FACL	Yes	Yes	No	54	Yes	13
FACRLN	Yes	Yes	No	11	Yes	8.68
RLIDS	Yes	Yes	No	4	No	83

5.2. Ball and beam

This problem tries to stabilize a ball on a beam at position zero of the beam. Figure 17 shows this system.

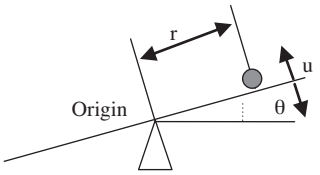

Figure 17. Ball and beam system.

The model of this system is [20]:

$$x = \left(r, \dot{r}, \theta, \dot{\theta}\right)$$

$$\dot{x} = f(x) + g(x)\,u$$
$$y = x_1$$
$$f(x) = \begin{bmatrix} x_2 \\ B\left(x_1 x_4^2 - G sin x_3\right) \\ x_4 \\ 0 \end{bmatrix} g(x) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{20}$$

where r is the position of the ball on the beam, $\dot{r}$ is the velocity of the ball, θ and $\dot{\theta}$ are the angles of the beam and its velocity, respectively, and B and G are constant values. Figure 18 shows the system state after 10,000 time steps.

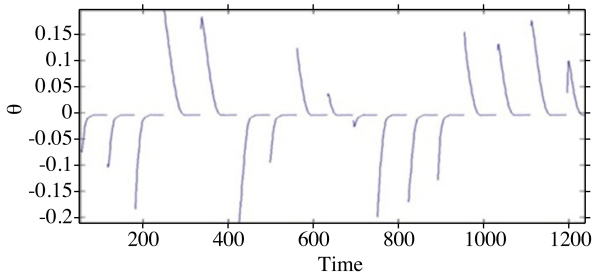

Figure 18. Output of the plant (angle) after 10,000 time steps.

Table 2 compares the control parameters for the 5 controlling methods. The unsupervised ALM and the supervised ALM were proposed in [10]. RALM is an off-line RL method that was proposed in [11]. The results show that the proposed algorithm achieves less rise time and overshoot with the same number of rules compared to the other ALM-based learning methods.

Table 2. Comparing the control parameters of the 5 controlling methods.

	Rule numbers	Overshoot	Rise time
FALCON-ART	28	23.5%	2.11
Unsupervised ALM	4	14.3%	1.87
Supervised ALM	4	0%	1.85
RALM	4	0%	1.61
RLIDS	4	0%	1.31
Gaussian	4	0%	1.28

5.3. Two-wheel balancing robot

The TWBR is a robot that must be stable on its 2 wheels [3]. The model is shown in Figure 19. This is a sample of a nonholonomic system.

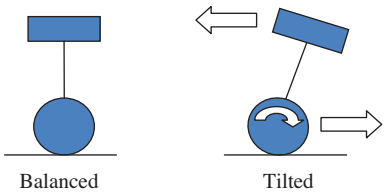

Figure 19. The 2-wheeled balancing robot.

The model of this system is:

$$3\left(M_w + M_p\right)\ddot{x} - M_p L\ddot{\theta}cos\theta + M_p L\dot{\theta}^2 sin\theta = -\left(\frac{C_L + C_R}{R}\right),$$

$$M_p L\ddot{x}cos\theta + \left(-M_p L^2 - J\right)\ddot{\theta} + M_p L^2\dot{\theta}^2 sin\theta cos\theta + M_p gLsin\theta = C_L + C_R \quad (21)\ , \quad (21)$$

where M_p and M_w are the masses of the chaises and wheel, respectively; x and θ are the location and the angle of the robot, respectively; L is the length of the chaises; R is the radius of the wheels; J is the inertia of the robot; and C_L and C_R are the torques that are applied to the left and right wheels.

We choose vector z as $z = (\theta\theta, \dot{x}, \dot{x})$ and have:

$$\dot{z}{=}f\left(z{,}u\right)u = C_u = C_R. \quad (21)$$

We simulated it using the Runge–Kutta method. The parameters are chosen to be $R = 0.106$ m, $L = 0.4$ m, $J = 1.12$ kg m^2, $g = 9.8$ m s^{-2}, $M_p = 21$ kg, and $M_r = 0.420$ kg. The input ranges are θ = [–0.2, 0.2] rad and $\dot{\theta}$ = [–4, 4] rad/s, and the output range is u = [–15, 15]. Each input was broken into 3 membership functions. Simulation was done up to time step 35,000. The rewards, penalties, and TD error are shown in Figure 20.

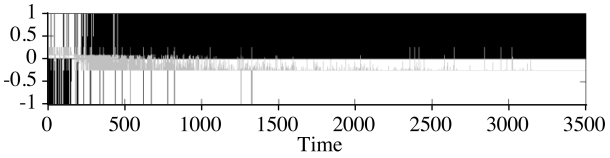

Figure 20. Rewards and penalties during the learning process (in black); gray marks demonstrate the TD error (–0.25 is the origin).

The numbers of rewards and penalties in the interval of 500 time steps are shown in Figure 21. The IDS planes, the narrow paths, and the trust value of each rule for each point of input variables are shown in Figure 22. The system state, θ, is shown in Figure 23 after 35,000 time steps.

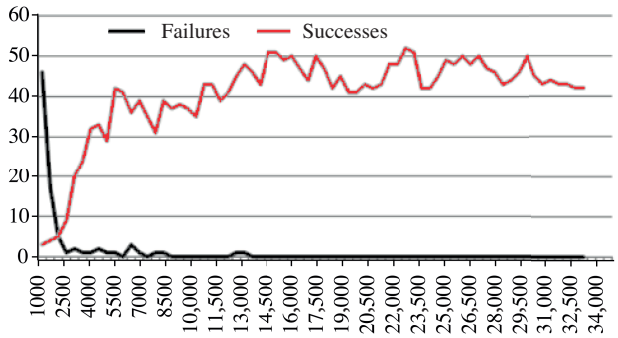

Figure 21. Number of rewards and penalties in each of 500 time steps.

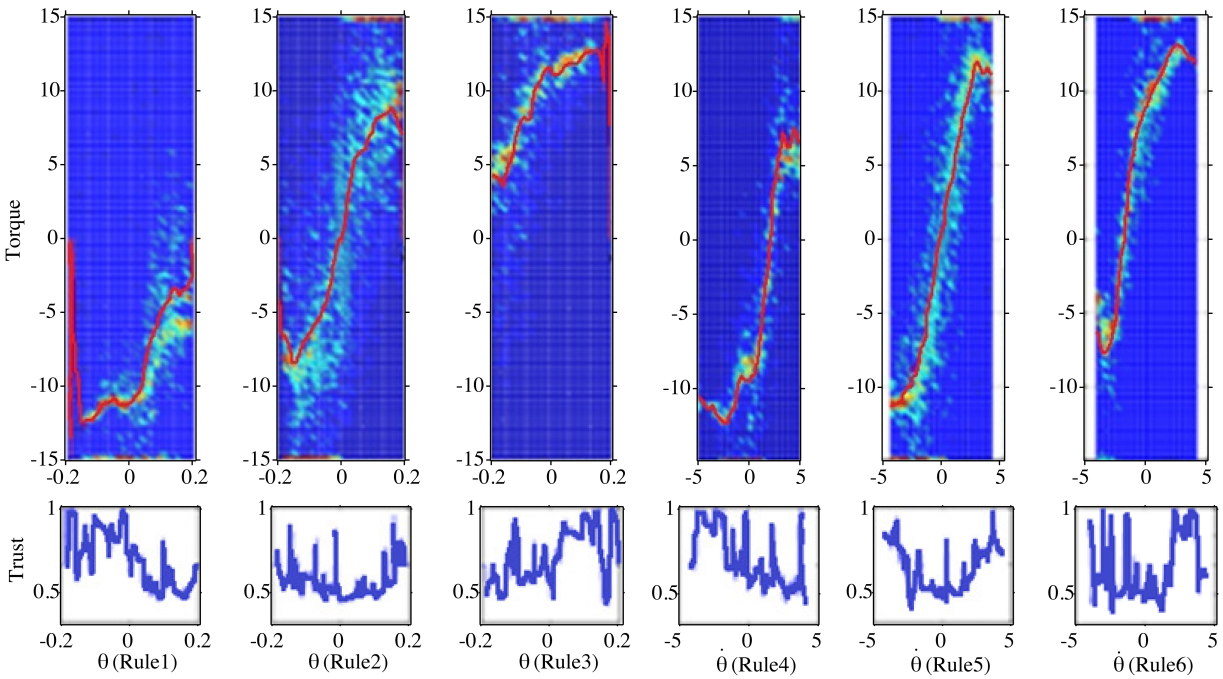

Figure 22. Narrow paths and the degrees of trust for each rule.

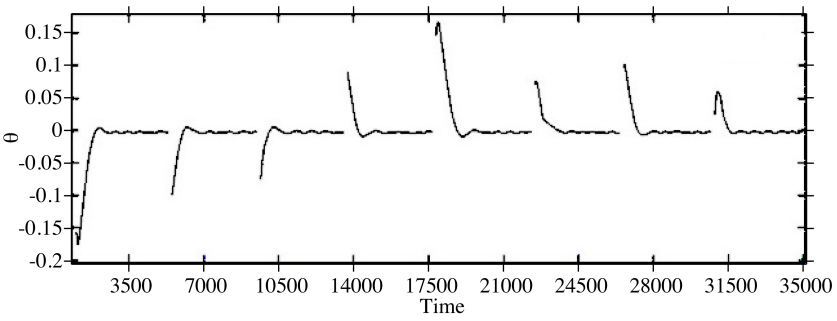

Figure 23. System states after 35,000 time steps.

Table 3 shows the overshoot and rise time values with respect to the number of rules. As can be seen, increasing the number of rules makes for a better behavior of the system. Figures 24a–24f show the evolution of the RPP during the learning process for each of the 8000 iterations.

Table 1. Overshoots and rise times with respect to the number of rules; (-) means that convergence was not obtainable.

Number of rules	Overshoot	Rise time
2	-	-
4	-	-
6	4.5%	0.066
8	1.5%	0.062

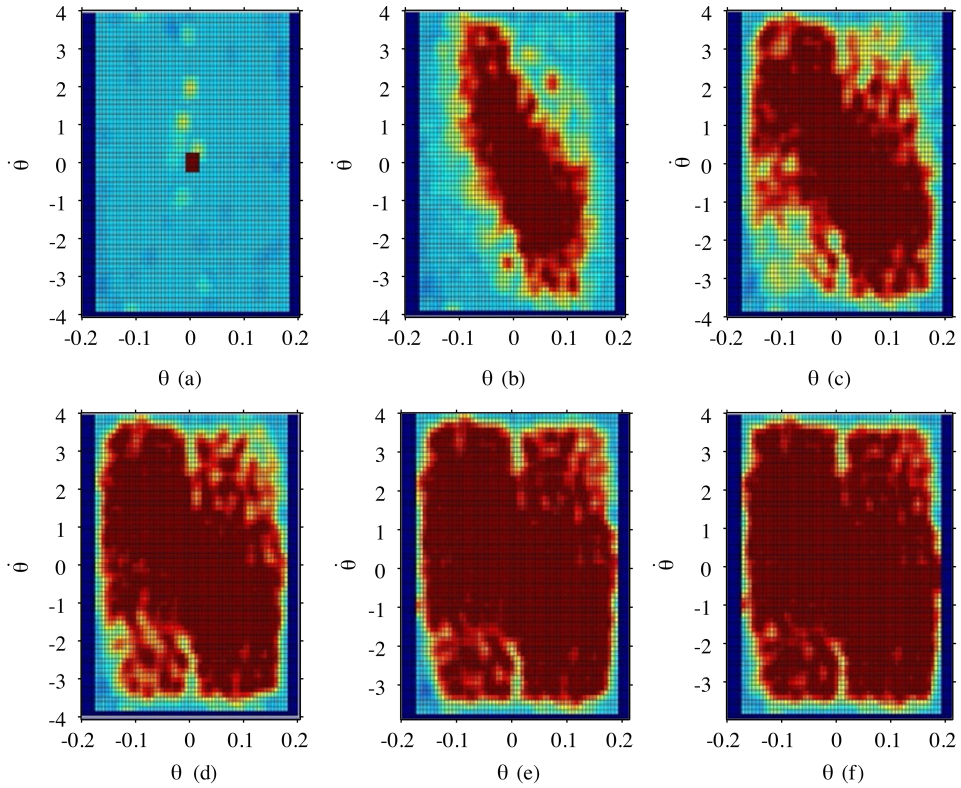

Figure 24. Reward-penalty plane after a) 8000, b) 16,000, c) 24,000, d) 32,000, e) 40,000, and f) 48,000 iterations. The $x-$ and y-axes represent θ and $\dot{\theta}$, respectively.

6. Conclusion

In this paper, we proposed an intelligent control method based on RL called RLIDS. The main engine of this algorithm is the IDS modeling method. To add reinforcement capability to the IDS method, we take advantage of the actor-critic method. An IDS system acts as an actor and a new plane called the RPP acts as a critic. The IDS planes and the RPP are updated using the TD error, which is obtained by the RPP. The IDS method models rewards and penalties to converge to the best action.

We applied the RLIDS on 3 systems: inverted pendulum, ball and beam, and TWBR. In all of these systems, our method learned to do an acceptable action at each state better than the other ALM-based controllers.

Acknowledgment

The financial support of the Iran National Science Foundation is gratefully acknowledged.

References

[1] G. Feng, "A survey on analysis and design of model-based fuzzy control systems", IEEE Transactions on Fuzzy Systems, Vol. 14, pp. 676–697, 2006.

[2] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction. Cambridge, MIT Press, 1998.

[3] A.G. Barto, R.S. Sutton, C.W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems", IEEE Transactions on Systems, Man and Cybernetics, Vol. 13, pp. 834–846, 1983.

[4] A. Miloudi, A. Al-Radadi, A.D. Draou, "A variable gain PI controller used for speed control of a direct torque neuro fuzzy controlled induction machine drive", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 15, pp. 37–49, 2007.

[5] S.B. Shouraki, N. Honda, "Outlines of a soft computer for brain simulation", International Conference on Soft Computing Information/Intelligence Systems, 1998.

[6] D. Aha, D. Kibler, M. Albert, "Instance-based learning algorithms", Journal of Machine Learning, Vol. 6, pp. 37–66, 1991.

[7] H.T. Shahraiyni, S.B. Shouraki, F. Fell, M. Schaale, J. Fischer, A. Tavakoli, R. Preusker, M. Tajrishy, M. Vatandoust, H. Khodaparast, "Application of the active learning method to the retrieval of pigment from spectral remote sensing reflectance data", International Journal of Remote Sensing, Vol. 30, pp. 1045–1065, 2009.

[8] S.B. Shouraki, N. Honda, "Fuzzy controller design by an active Learning method", 31st Symposium of Intelligent Control, 1998.

[9] S.B. Shouraki, N. Honda, "Hardware simulation of brain simulation process", 15th Symposium of Fuzzy System, 1999.

[10] S.A. Shahdi, S.B. Shouraki, "Supervised active learning method as an intelligent linguistic controller and its hardware implementation", Proceedings of the 2nd IASTEAD International Conference on Artificial Intelligence and Applications, 2002.

[11] Y. Sakurai, "A study of the learning control method using PBALM-a nonlinear modeling method", PhD, The University of Electro-Communications, Tokyo, 2005.

[12] M. Murakami, N. Honda, "A study on the modeling ability of the IDS method: a soft computing technique using pattern-based information processing", International Journal of Approximate Reasoning, Vol. 45, pp. 470–487, 2007.

[13] A.G. Barto, S.J. Bradtke, S.P. Singh, "Learning to act using real-time dynamic programming", Artificial Intelligence, Vol. 72, pp. 81–138, 1995.

[14] R.S. Sutton, "Learning to predict by the methods of temporal differences", Machine Learning, Vol. 3, pp. 9–44, 1988.

[15] H. Berenji, P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcement", IEEE Transaction on Neural Networks, Vol. 3, pp. 724–740, 1992.

[16] X.S. Wang, Y.H. Cheng, J.Q. Yi, "A fuzzy actor–critic reinforcement learning network", Information Sciences, Vol. 177, pp. 3764–3781, 2007.

[17] H.V. Hasselt, M.A. Wiering, "Reinforcement learning in continuous action spaces", Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007.

[18] M.A. Wiering, H.V. Hasselt, "Two novel on-policy reinforcement learning algorithms based on TD(λ)-methods", Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007.

[19] F. Grasser, A. D'Arrigo, S. Colombi, A. Rufer, "JOE: A mobile, inverted pendulum", IEEE Transactions on Industrial Electronics, Vol. 49, pp. 107–114, 2002.

[20] J. Hauser, S. Sastry, P. Kokotovix, "Nonlinear control via approximate input-output linearization: the ball and beam example", IEEE Transaction on Automatic Control, Vol. 37, pp. 392–398, 1992.