52nd IEEE Conference on Decision and Control
December 10-13, 2013. Florence, Italy

Spike-Based Indirect Training of a Spiking Neural Network-Controlled Virtual Insect

X. Zhang*, Z. Xu*, C. Henriquez*, S. Ferrari*

***Abstract*— Spiking neural networks (SNNs) have been shown capable of replicating the spike patterns observed in biological neuronal networks, and of learning via biologically-plausible mechanisms, such as synaptic time-dependent plasticity (STDP). As result, they are commonly used to model cultured neural network, and memristor-based neuromorphic computer chips that aim at replicating the scalability and functionalities of biological circuitries. These examples of SNNs, however, do not allow for the direct manipulation of the synaptic strengths (or weights) as required by existing training algorithms. Therefore, this paper presents an indirect training algorithm that, instead, is designed to manipulate input spike trains (stimuli) that can be implemented by patterns of blue light, or controlled input voltages, to induce the desired synaptic weights changes via STDP. The approach is demonstrated by training an SNN to control a virtual insect that seeks to reach a target location in an obstacle populated environment, without any prior control or navigation knowledge. The simulation results illustrate the feasibility and efficiency of the proposed indirect training algorithm for a biologically-plausible sensorimotor system.**

I. Introduction

This paper presents a spike-based indirect training approach that is applicable to spiking neural networks (SNNs), such as *in vitro* biological neuronal networks, and memristor-based neuromorphic computer chips, that do not allow for the direct manipulation of the synaptic strengths, or weights, that is instead required by existing SNN training algorithms [1]–[5]. The proposed training approach is said to be spike-based, because instead of manipulating the synaptic weights, it manipulates input spike trains that can be implemented by patterns of blue light, or controlled input voltages, to induce the desired the synaptic weights changes via synaptic time-dependent plasticity (STDP).

The spike-based indirect training approach is demonstrated by training the simulated sensorimotor system of a virtual insect in a path planning and control problem. The virtual insect is controlled by an SNN that receives sensory inputs from terrain and vision sensors (antennas), and that seeks to reach a random target located in a terrain with varied roughness. Unlike typical implementations of feedback control in path planning problems, which require models of the vehicle (insect) dynamics and environment, and can suffer from inefficiencies due to online obstacle detection, the indirect training algorithm presented in this paper always produces feasible trajectories under dynamic constraints. Because of its ability to control the insect without any prior modeling, control, or navigation knowledge, the SNN controller presented in this paper is also applicable to mobile sensors and autonomous robots [6]–[10].

Recent work has shown that SNNs are capable of solving nonlinear function approximation problems in few dimensions [1], [2], [11]. Furthermore, due to their ability to simulate biological neuronal networks, they are used in a variety of neuroscience and neurobiology applications, such as the study of multi-cortical computational models [12], *in vitro* biological neuronal networks [13], and CMOS/memristor devices [14], [15]. The effectiveness of SNN training algorithms to date remains very limited compared to artificial neural networks, and is yet to be demonstrated on challenging control problems. One of the main challenges to be overcome is that the response of an SNN is not available in closed-form and, typically, must be obtained numerically by solving a system of differential equations. Another challenge is that the SNN response consists of complex spike patterns that need to be decoded into reduced-order continuous signals to be utilized for control, or to assess the system-level performance of the SNN [16], [17].

One line of research, reviewed in [18], has focused on modifying backpropagation algorithms for SNNs, which are, however, unsupported in biological neuronal networks [2]. Another line of research has explored biologically-plausible learning mechanisms, such as Hebbian plasticity and STDP, both of which are supported by significant experimental evidence [1]–[5]. However, the latter class of algorithms relies on the direct manipulation of the synaptic weights and, thus, implements incremental weight changes computed by a Hebbian/STDP learning rule.

A recent study [19] demonstrates the computational efficiency of spike driven synaptic plasticity (SDSP) for pattern recognition with the assumption that the synaptic weights are known. However, it is not within the realm of current technological abilities to measure the synaptic weights and synaptic connections of *in vivo* neural networks. Therefore, in this paper, the values and changes in synaptic weights are assumed unknown *a priori*. Weight adjustments are induced by the STDP mechanism, and can be controlled indirectly by implementing a square-pulse function obtained from the RBF spike model. The square-pulse function is obtained by integrating the RBF spike model against a suitable averaging function in a leaky integrator, and by comparing it to a positive threshold, by means of a so-called IF sampler. As a result, a precise square-pulse function is obtained with widths, intensities, and timings determined by the parameters of the RBF model. This paper is structured as follows: the path planning and control problem is formulated in Section II and the fundamental mathematical models of spiking neurons and STDP are reviewed in Section III-A. Then the SNN architecture and indirect training algorithm presented in Section IV are used to solve the problem described in Section II. The simulation results in Section V demonstrated the efficiency and feasibility of the proposed spike-based indirect training algorithm.

II. Problem Formulation and Assumptions

The spike-based indirect training approach presented in this paper is demonstrated on a path planning and control problem in which a virtual insect equipped with target and terrain sensors, processes environmental autonomously via SNNs. The physical characteristics of the virtual insect can be described by a rigid object $\mathcal{A}$ that is a compact subset of a workspace $\mathcal{W} \subset \mathbb{R}^2$. The workspace $\mathcal{W}$ can either contain smooth or rugged territory, with roughness and solid obstacles, denoted by $\mathcal{B}_1, ..., \mathcal{B}_N$, that must be avoided by $\mathcal{A}$. The virtual insect uses two antennas, shown in Fig. 1, where the dashed circle $\mathcal{S}$ of radius r represents the field of view of the left (target) antenna, which depends on the terrain roughness, and the red dot at the top of the red antenna is the field of view of the right (terrain) antenna. Using sensory information obtained by the antennas, the virtual insect must process the sensory inputs and adjust its current state according to the corresponding desired movement. The position and orientation of the insect with respect

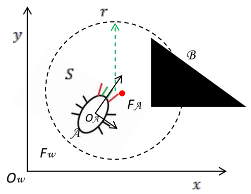

Fig. 1. Insect geometry and workspace coordinates.

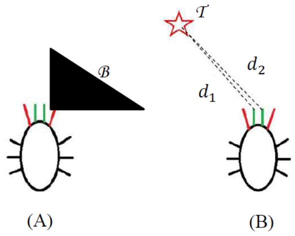

Fig. 2. Insect terrain sensors (red antennas) (A), and target sensors (green antennas) (B).

to $\mathcal{W}$ are defined with respect to an inertial reference frame $F_{\mathcal{W}}$ with origin $O_{\mathcal{W}}$ in $\mathcal{W}$. The insect's sensory inputs are defined with respect to a moving reference frame $F_{\mathcal{A}}$, embedded in $\mathcal{A}$, and with origin $O_{\mathcal{A}}$. It is assumed that both $\mathcal{A}$ and $\mathcal{S}$ are rigid objects, and that $\mathcal{S}$ has a fixed orientation and position with respect to $\mathcal{A}$. Let $q \in \mathbb{R}^3$ denote the configuration of the virtual insect, such that $q = [x \ y \ \theta]^T$, where x and y are the Cartesian coordinates in $F_{\mathcal{W}}$, and θ is the heading angle of the virtual insect. Then, $\mathcal{A}(q)$ denotes the compact subset of $\mathcal{W}$ that is occupied by $\mathcal{A}$ when the insect is at a configuration $q \in \mathcal{C}$, where $\mathcal{C}$ is the insect's configuration space. Similarly, the subset of $\mathcal{W}$ occupied by $\mathcal{S}$ at q can be denoted by $\mathcal{S}(q)$, and is the set of all the accessible sensor information that can be obtained by the insect when $\mathcal{T}_{obj} \cap \mathcal{S}(q) = \emptyset$, where $\mathcal{T}_{obj}$ is the geometry of the target.

The motion of the virtual insect is simulated using an adaptation of the unicycle robot that can more closely represent insect locomotion [20],

$$\begin{cases} \dot{x} = v \ cos\theta \\ \dot{y} = v \ sin\theta \\ v = \frac{v_1+v_2}{2} \\ \dot{\theta} = \frac{v_2-v_1}{L} \\ \dot{v_s} = -\frac{v_s}{\tau_{motor}} + \eta \cdot H(t-t_i^f), s \in [1,2] \end{cases} \tag{1}$$

where v is the linear velocity, v_s is the s_{th} motor speed, $\dot{\theta}$ is the angular velocity, L is the distance between two motors, τ_{motor} is the time constant that results in a gradual decay of the motor speed following activation of the motor, t_i^f is the firing time of the output neuron i and $H(\cdot)$ is the heaviside function which is scaled by a constant η.

The objective of the virtual insect is to reach the target $\mathcal{T}_{obj}$, while avoiding rugged terrain in $\mathcal{W}$ by using visual and terrain information obtained from its antennas. Although the problem formulation and equations presented in this section are used to describe the insect motion, they are not used to design the insect SNN controller. Instead the SNN, described in the next section, is trained using the spike-based indirect algorithm presented in Section IV, based on sensory inputs to which the insect responds at any time $t > t_0$.

III. Mathematical Models

A. Modeling of Neuron and Alpha Synapse

Spiking neuron models are mathematical representation of spike pattern dynamics observed in biological neurons, such as action-potential generation, refractory periods, and post-synaptic potential shaping. The leaky integrate-and-fire (LIF)-SNN is adopted in this paper because it is the simplest model of spiking neuron, and is amenable to mathematical analysis. The LIF-SNN also provides the highest computational efficiency compared to other neuron models [21]. The LIF membrane potential can be modeled by the differential equation [21],

$$\tau_m \frac{dV(t)}{dt} = -V(t) + R_m[I_{stim}(t) + I_{syn}(t)] \quad (2)$$

where I_{stim} is the stimulus current (e.g. from an external stimulus such as blue light or a controlled input voltage), and I_{syn} is the synaptic current from the presynaptic neurons. τ_m is the passive membrane time constant, and R_m is the membrane resistance of the neuron, both of which can be assumed constant. When the membrane potential V reaches a threshold value V_{th}, the neuron fires (spikes), and the membrane potential returns to a resting potential E_{rest}. The synaptic current from the presynaptic neurons can be modeled as,

$$I_{syn}(t) = g_{syn}(t)[V(t) - E_{syn}] \quad (3)$$

where $g_{syn}(t)$ is the synaptic conductance, and E_{syn} is the reversal potential for the synapse. The $g_{syn}(t)$ in this paper are modeled using this two-parameter alpha function for representing the rising and decay phases [22],

$$g_{syn}(t) = \bar{g}_{syn} h(e^{-\frac{t}{\tau_{delay}}} - e^{-\frac{t}{\tau_{rise}}}) \quad (4)$$

where, to ensure that the amplitude equals $\bar{g}_{syn}$, the normalization factor in (4) is defined as,

$$h = (-e^{-\frac{t_{peak}}{\tau_{rise}}} + e^{-\frac{t_{peak}}{\tau_{decay}}})^{-1} \quad (5)$$

and the conductance peaks at a time

$$t_{peak} = \frac{\tau_{decay}\ \tau_{rise}}{(\tau_{decay} - \tau_{rise})} \ln\left(\frac{\tau_{decay}}{\tau_{rise}}\right) \quad (6)$$

The time constants of synapse conductance vary widely among synapse types. In our paper, excitatory synapses are modeled with AMPA receptors, which are found in many parts of the brain and are the most commonly found receptors in the nervous system [23], and inhibitory synapses are modeled using the fast inhibition GABA$_A$ receptor.

B. Spike-Timing Dependent Plasticity (STDP)

In this paper, it is assumed that all of the SNN synapses change over time only by virtue of the STDP rule. The STDP can be implemented in a LIF-SNN, during every time step Δt, to adapt the synaptic weight w_{ij} between a pre-synaptic neuron j and a neuron i based on the relative timing of the pre- and post-synaptic spikes, denoted by $\hat{t}_j$ and $\hat{t}_i$, respectively, such that [21]:

$$w_{ij}(t + \delta t) = w_{ij}(t)(1 + \Delta w_{ij}(t)) \quad (7)$$

$$\Delta w_{ij}(t_i^f) = D_+(w_{ij}) \cdot x_j(t_i^f) \quad (8)$$

$$\Delta w_{ij}(t_j^f) = -D_-(w_{ij}) \cdot y_i(t_j^f) \quad (9)$$

where the functions,

$$D_+(w_{ij}) \triangleq \lambda(1 - w_{ij})^\mu, \quad D_-(w_{ij}) \triangleq \xi\lambda w_{ij}^\mu \quad (10)$$

describe how the weight changes depend on the current weights. if $\mu = 0$, the rule becomes additive STDP. λ is the learning rate, and ξ is the asymmetry parameter. As observed in biological neurons, if the presynaptic neuron fires before the postsynaptic neuron, the synapse is strengthened, and if it fires after, the synapse is weakened. The pair-based STDP rule can be numerically implemented in the LIF-SNN using two local variables, x_j and y_i for low-pass filtered version of the presynaptic spike train and the postsynaptic spike train [21]. Let us consider the synapse between neuron j and neuron i. Suppose that each spike from presynaptic neuron j contributes to a trace x_j at the synapse,

$$\frac{dx_j}{dt} = -\frac{x_j}{\tau_x} + \sum_{t_j^f} \delta(t - t_j^f) \quad (11)$$

where t_j^f denotes the firing times of the presynaptic neuron. In other words, the trace x_j is increased by an amount of one at t_j^f and decays with time constant τ_x afterwards. Similarly, each spike from postsynaptic neuron i contributes to a trace y_i,

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_y} + \sum_{t_i^f} \delta(t - t_i^f) \quad (12)$$

where t_i^f denotes the firing times of the postsynaptic neuron. When a presynaptic spike occurs, the weight decreases proportionally to the momentary value of the postsynaptic trace y_i. Similarly, when a postsynaptic spike occurs, a potentiation of the weight is induced.

C. Sensor Models

The virtual insect uses terrain and target sensors, represented by the antennas in Fig. 2 in order to detect the roughness of terrain, and the distances, d_1 and d_2, between the antennas and the target. The roughness of the terrain is represented by grayscale values, where 255 (white) is the flat region and 0 (black) is the roughest region. Thus, the terrain sensor has an input,

$$S_m = \gamma|M(x, y) - 255| \quad (13)$$

where γ is a scaling constant for the intensity of sensor inputs, and $M(x, y)$ is the grayscale value at (x, y). The two target (e.g. vision) sensors determine the position of the target $\mathcal{T}$ by calculating the Euclidean distance between each of the two (green) antennas, and the target (star). Then, the sensory input from the target sensors is defined as,

$$S_t = \mu\|P(x, y) - T(x, y)\| \quad (14)$$

where μ is a scaling constant for the intensity of sensor inputs, and $P(x,y)$ and $T(x,y)$ are the coordinates of the sensor and target, respectively.

IV. Spike-Based Indirect Training Approach

The indirect spike-based training approach is illustrated on a seven-node SNN, shown in Fig. 3, that includes both excitatory and inhibitory synapses, $n = 4$ input neurons that receive information from each of the insect antennas, and $r = 2$ output neurons that control the two motors of the insect. During every iteration cycle

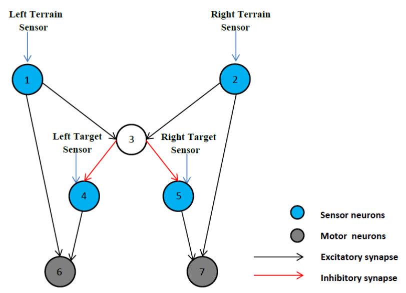

Fig. 3. SNN architecture

l of the training algorithm, the desired SNN input-output (unknown) mapping g_l: $R^{n\times 1} \rightarrow R^{r\times 1}$ is assumed to be stationary. Let $W_l(t_k) = w_{ij}(t_k)$ represents a matrix containing the SNN synaptic weights at time t_k. For any set of weights, the SNN is provided with an observation of the input, s^l, which is encoded as n constant current inputs over a time interval $[t_k, t_k + \tau]$. The decoded output of the SNN matches the output $y^l = g_l[s^l]$. Because of $\tau \ll (t_{k+1} - t_k)$, the amplitude of the constant current ω can be used to encode, at any t_k, instantaneous values of s^l. Then, the indirect training algorithm modifies the synaptic weights of the SNN by injecting the training currents into the training neurons of the SNN, such that the weight matrix W are optimized in (15),

$$y^l(t_k) = g_l[s^l(t_k)] \leftarrow SNN[s^l(t_k), W_l(t_k)], \qquad (15)$$

where s^l is the input of the SNN at time t_k, y^l is the desired output of the SNN, $g_l(x)$ is the desired mapping between the inputs and outputs of the SNN, and $W_l(t_k)$ is the weights matrix of the SNN at time t_k, which contains the weight w_{ij} between neuron j and i.

The distance between y^l and the decoded SNN's output is optimized with respect to the parameters of a new deterministic spike train model, as the individual synaptic weights w_{ij}, can not be set directly. In order to change W_l in a manner that improves the SNN performance of the mapping in (15), the programming voltages are generated based on an optimized spiking sequence $T_i^l = \{\hat{t}_i^l : i = 1, ..., N\}$ during the l^{th} time interval $[t_l, t_{l+1}]$, where N is the total number of neurons in the SNN. Existing stochastic spike models [21] cannot be used to generate T_i^l. As a result, they do not allow for the precise timing of pre- and post-synaptic firings, which can lead to undesirable changes in synaptic weights by virtue of the STDP rule in III-B. This paper utilizes a new radial basis function (RBF) spike model shown in Fig. 4 from previous work [24] ,

$$s_i^l(t) = \sum_{k=1}^{M_i^l} \omega_{i,k} exp[-\beta_{i,k}(\|t - c_{i,k}\|)^2], \qquad (16)$$
$$i = 1, ..., q, \; t \in [t_l, t_{l+1}] \subset [t_k, t_k + T]$$

where $c_{i,k}$ represent the centers, the biases $\beta_{i,k}$ are widths, and the weights $\omega_{i,k}$ correspond to the heights of the RBF spike k, M_i^l is the total number of RBFs during $[t_l, t_{l+1}]$. The continuous signal $s_i^l(t)$ in (16) is integrated against a suitable averaging function in a leaky integrator and, then, compared to a positive threshold, by means of a so-called IF sampler [25]. As a result, a precise pulse function, with a width that is dependent on the value of $\beta_{i,k}$, and an intensity that depends on $\omega_{i,k}$, is generated with center at $c_{i,k}$, during the interval $[t_l, t_{l+1}]$. Therefore, by using the RBF spike model in (16) with suitable widths and heights, it is possible to induce the neuron to fire shortly before the end of each pulse of the square-pulse function (also considering the refractory period Δ_{abs} of the neuron).

Because of the relationship in (2), the time it takes for a neuron to fire with constant current input should be,

$$T^f = -\tau_m ln[1 - \frac{V_{th} - E_{rest}}{\omega R_m}], \quad (\omega R_m > V_{th} - E_{rest}) \qquad (17)$$

where T_s is the duration for the neuron to fire, when it is given the constant current stimulus ω. Accordingly, the fire time of the neuron can be expressed as,

$$t_{i,k}^f = c_{i,k} - \frac{\beta_{i,k}}{2} + T^f \qquad (18)$$

For simplicity, in this paper, it is assumed that for all k, the heights $\omega_{i,k}$ and widths $\beta_{i,k}$ are known positive constants of equal magnitudes. Then, the centers of the RBF comprise the set of adjustable parameters, $P_i = \{c_{i,k} \mid k = 1, \ldots, M_i^l\}$, to be optimized. The same approach can be easily extended to a case where all of the RBF parameters are adapted. The optimal RBF parameters P_i^* used to generate s^l are determined from minimization of the distance between the decoded output of the SNN, y_N, and the desired output, y^l.

The square pulses from the output of IF sampler cannot overlap with each other. For this reason, the

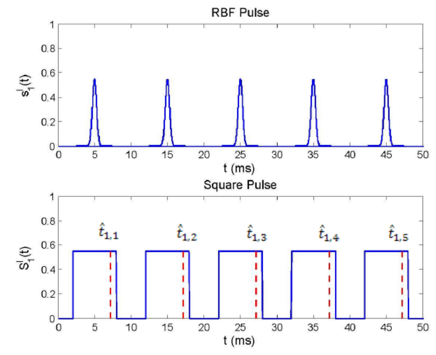

Fig. 4. RBF is converted to square pulses with corresponding parameters. $\hat{t}_{i,k}$ is the k_{th} firing time of neuron i

constraints on vector P_i are as follows:

$$\begin{aligned} t^f_{i,k} &\le c_{i,k} + \frac{\beta}{2} \\ t^f_{i,k} + \Delta^{abs} &\ge c_{i,k} + \frac{\beta}{2} \end{aligned} \tag{19}$$

A. Optimization of RBF Parameters

If either the left or right terrain sensor detects the rough terrain, the insect adjusts its direction in order to avoid entering the respective location. This desired behavior is assumed linear and can be mathematically formulated as follows,

$$\begin{pmatrix} F_L \\ F_R \end{pmatrix} = \begin{pmatrix} S_{trL} & S_{tL} \\ S_{trR} & S_{tR} \end{pmatrix} \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \tag{20}$$

where F_L and F_R are the desired firing frequencies of the left and right motor neurons; S_{trL}, S_{tL}, S_{trR}, S_{tR} are, respectively the left terrain sensor value, the left target sensor value, the right terrain sensor value, and the right target sensor value. The constants α, γ scale the sensor values, where $\alpha > \gamma$ due to fact that inputs from the terrain sensors are prioritized when the virtual insect approaches rough terrain.

The coding schemes for SNNs include both rate coding, which is the frequency of neuron spikes, and temporal coding, which is correlated with the exact firing times of the neurons. This paper employs the rate coding to decode the output spikes of the neurons. The average firing rate of neuron i over time interval $[t_{l-1}, t_l]$ can be calculated by (21), in which l is the index of the epoch. During our simulation, $[t_{l-1}, t_l]$ is the testing epoch, whereas $[t_l, t_{l+1}]$ is the training epoch. The sensor neurons (see Fig. 3) receive inputs from the sensors of the virtual insect. The firing frequencies of the outputs, which connect to the motors, are depicted $f^p_i(t_l)$, where i is the index of the neuron and p is the index of the training samples. The error, δ, at time t_l is defined by (22).

$$f^p_i(t_l) = \frac{\sum_{t_{l-1}}^{t_l} H(v_i(t_{l-1}) > V_{th})}{\Delta t}, \qquad \Delta t = t_l - t_{l-1} \tag{21}$$

$$\delta(t_l) = \frac{\sum_{i \in O} \sum_{p=1}^{P_m} ||F^p_i - f^p_i(t_l)||}{2 * P_m} \tag{22}$$

where O is the set of indices identifying the output neurons in the SNN, P_m is the total number of training samples and F^p_i is the desired firing frequency of the neuron i for training samples p, $f^p_i(t_l)$ is the firing frequency of the neuron i for training samples p during $[t_{l-1}, t_l]$.

During each training epoch $[t_l, t_{l+1}]$, only one square pulse is injected into each neuron, and the time difference between the centers of the RBF input to neuron j and i, d^r_{ji}, is calculated by (23),

$$d^r_{ji} = c^r_j - c^r_i, \qquad i, j \in [1, 2, \cdots, N] \tag{23}$$

where i, j are the index of neurons, r is the index of the training epoch, c^r_j, c^r_i are the centers of the square pulse in training epoch r, and N is the total number of neurons in the SNN. In (24), the d^r_{ji} is calculated by steepest descent method as below,

$$d^{r+1}_{ji} = d^r_{ji} - \lambda \frac{\partial \delta}{\partial w^r_{ji}} \tag{24}$$

where λ is a constant learning rate. The weights w_{ji} is assumed to be unknown, therefore each pair of pre- and postsynaptic firing are separated long time enough to calculate the weights change by (25),

$$\Delta w^r_{ij} = \begin{cases} D_+ \cdot e^{-d^r_{ij}/\tau_+} & d^r_{ij} > 0 \\ -D_- \cdot e^{d^r_{ij}/\tau_-} & d^r_{ij} < 0 \end{cases} \tag{25}$$

where D_+, D_- are the constant amplitudes of the weights change, $d^r_{ij} = c^r_i - c^r_j$ is the time difference between the firing time of post- and presynaptic neuron during training, and τ_+, τ_- are the time constants. By submitting (25) into (24), the steepest descent algorithm works without knowing the synaptic weights during simulation, which is shown in (26)(27).

$$d^{r+1}_{ji} = d^r_{ji} - \lambda \frac{\partial \delta}{D_+ \cdot e^{-d^r_{ij}/\tau_+}} \qquad d^r_{ij} > 0 \tag{26}$$

$$d^{r+1}_{ji} = d^r_{ji} - \lambda \frac{\partial \delta_r}{-D_- \cdot e^{d^r_{ij}/\tau_-}} \qquad d^r_{ij} < 0 \tag{27}$$

At every time step, d^r_{ji} is calculated as described above, and the values of c^r_i, c^r_j are calculated using (23). Then we input the RBF to the neurons during each training epoch. A square pulse is injected into each training neuron, such that the SNN is trained by our

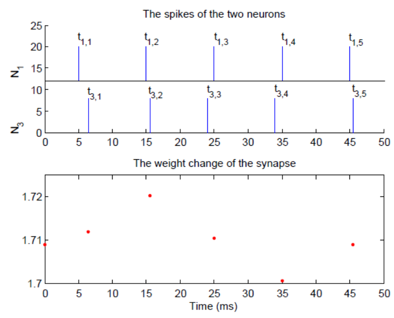

Fig. 5. Optimized input spike train, and indirect weight changes brought about by STDP.

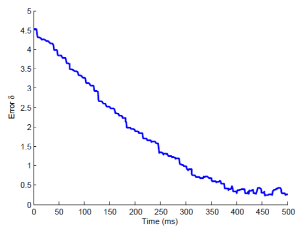

Fig. 6. Error δ during training.

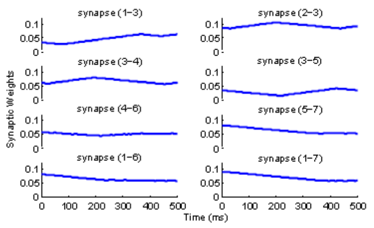

Fig. 7. Evolution of eight synaptic weights subject to STDP during training.The synapse between neuron 1 and 3 is labeled as synapse (1-3). See Fig. 3 for all connections.

indirect training method via the STDP rule discussed in Section III-B. Fig. 5 shows the optimized firing times of neurons 1 and 3 caused by the RBF inputs and the corresponding weight changes due to the STDP rule during training. If the output error is lower than a constant δ_{min}, the training stops and the trained SNN is used on the virtual insect problem.

V. Simulation Results

The architecture and algorithm of the indirect training approach presented in Section IV are used to train a SNN with randomized initialization, in order to perform target detection and terrain navigation. The objective of the simulations presented is to test the effectiveness of this training approach by comparing three trained states of the SNN including naive, partially-trained and fully trained on blank, s-maze, and cloud terrains.

Fig. 6 shows the error defined by (21) and (22) against the training duration where the error converges. Although it reaches at a minimum, the error cannot be further improved with more training. Therefore, the training would be stopped once the error is lower than δ_{min}. Due to the instability of solution caused by continuous strengthening/weakening of synapses, the synaptic weights are fixed after the training process completes. Fig. 7 demonstrates the evolution of synaptic weights with training inputs. The strengthened connections between terrain sensory neurons and neuron 3 (see Fig. 3) ensure the priority of terrain information; meanwhile the weights of inhibitory synapses are updated so that the both sides of SNN structure can be balanced. The synapses connecting with motor neurons can either be strengthened or weakened as long as the values of these synaptic weights are comparable to balance the two motor outputs.

The simulations are conducted in MATLAB® and in order to create the virtual environment, a 600×600 pixels image of the terrain and the target were generated. The initial positions of the virtual insect differ in the three environments. As illustrated by the examples in Section V-A to V-C, the indirect training method is capable of both strengthening and weakening synapses without directly manipulating synaptic weights. It is also capable of integrating information regarding the target location and terrain conditions, and, thus, it can train the virtual insect to avoid rough terrain on its path to the target.The movie clips for these results show a very realistic insect behavior, and can be downloaded from the URL at [26].

A. Blank Terrain

The properties and effectiveness of the indirect training is first tested in a simple environment where the terrain has uniform smoothness and, therefore only target information is relevant. As shown by the trajectory in Fig. 8-10, initially, the virtual insect rotates randomly in place. After partially trained, the virtual insect moves through the workspace, but does not approach the target. Finally, following the completion of the training procedure, the virtual insect approaches to the target using the path of shortest distance.

B. S-maze Terrain

In this scenario, the virtual insect must not only find the target, but integrate information about the terrain. The simulation results for naive, partially trained and fully trained states are illustrated in Fig. 11-13. The naive insect rotates without moving toward the target. In the partially-trained state, the insect initially moves away from the target and due to its capacity of terrain

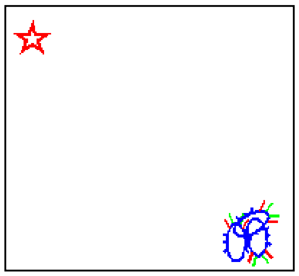
Fig. 8. Trace of naive insect on blank terrain (see movie in [26]).

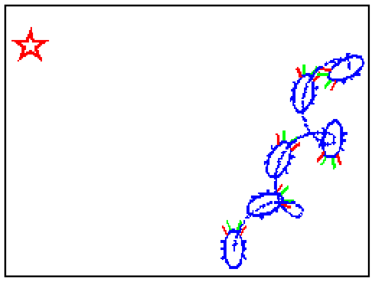
Fig. 9. Trace of partially-trained insect on blank terrain (see movie in [26]).

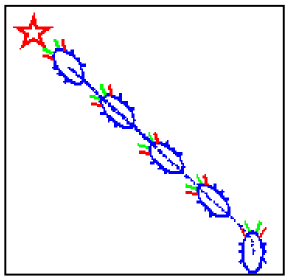
Fig. 10. Trace of fully-trained insect on blank terrain (see movie in [26]).

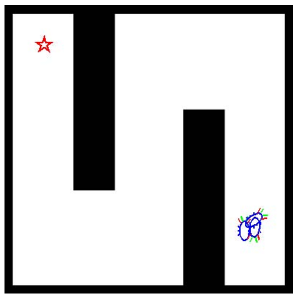
Fig. 11. Trace of naive insect on s-maze terrain (see movie in [26]).

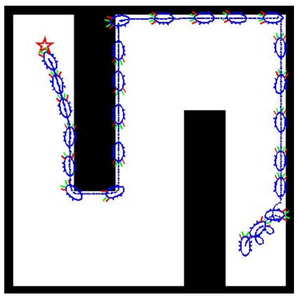
Fig. 12. Trace of partially-trained insect on s-maze terrain (see movie in [26]).

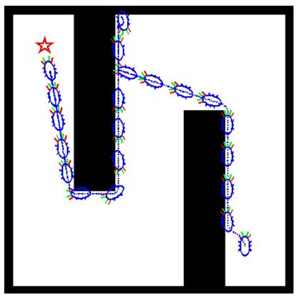
Fig. 13. Trace of fully-trained insect on s-maze terrain (see movie in [26]).

navigation, it successfully accomplishes the task by rambling along the black terrain.

C. Cloud Terrain

The cloud terrain is a heavily obstacle populated maze, created via Photoshop®. This environment introduces rough terrain and narrow channels, creating a complex and difficult landscape for the virtual insect to navigate. In the partially-trained case, the insect fails to acquire the target as it does in the s-maze terrain because of the complexity of cloud terrain. As expected, the fully trained SNN accounts for both target location and terrain roughness and effectively controls the virtual insect along its path.

D. Discussion

The training inputs and the desired outputs in this training algorithm follow the training rules as formulated in (20), which do not contain any knowledge about the locations of obstacles. Therefore once the SNN is fully trained, the virtual insect can navigate any terrain to get a randomly positioned target. Furthermore, the stability of the solution is ensured by fixing synaptic weights following the completion of training. A recent paper by Strauss et. al [27] also demonstrated the learning ability of association task in a bio-inspired SNN model. However, during training, a reward signal is injected into the pre-motor neuron and the training algorithm is path specific, in which only the synapses between input neurons for sample A and pre-motor neuron are strengthened. Therefore, it may not be suitable for the path planning problem described in our paper.

VI. Conclusions

This paper presents a spike based indirect training method that induces changes in the synaptic weights by controlled pulses abiding by the STDP rule, as opposed to the direct weight manipulation. The difficulty in using the indirect training method is that it seeks to optimize a pulse signal, which can be expressed as piece-wise continuous, multi-valued (or many-to-one), and non-differentiable functions that are prohibitive to numerical optimization. Additionally, stimulation patterns created by stochastic spike model can result in imprecise timing of pre- and post-synaptic firings, and thus can induce undesirable changes in the synaptic weights. In order to address the two problems mentioned above, this paper presents a deterministic and adaptive training algorithm by RBFs that can be easily optimized to determine precise spike timings that minimizes a desired objective function. The simulation of the virtual insect path planning shows that this algorithm can train SNN to approximate the mapping between the input and desired output, which enables the SNN to solve control problems like path planning both in biological neuronal networks, and in CMOS/memristor nanoscale chips. For future

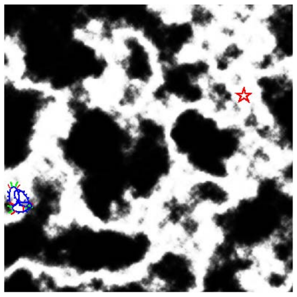
Fig. 14. Trace of naive insect on cloud terrain (see movie in [26]).

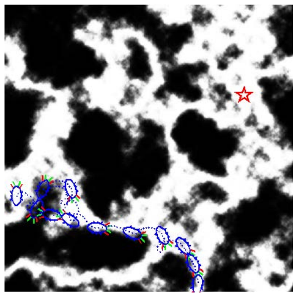
Fig. 15. Trace of half-trained insect on cloud terrain (see movie in [26]).

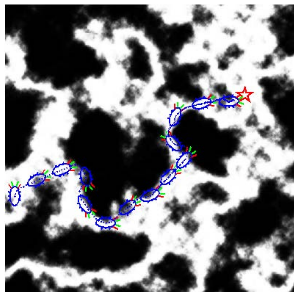
Fig. 16. Trace of fully-trained insect on cloud terrain (see movie in [26]).

investigation, the fully trained synaptic weights may still be able to further evolve rather than being constant and the SNN architecture may not be specially designed.

VII. Acknowledgments

This work was supported by the National Science Foundation, under ECCS Grant 0925407. The authors would like to thank Roy Tangsombavisit and Ryan Peters for developing the virtual insect locomotion model.

References

[1] S. Ferrari, B. Mehta, G. D. Muro, A. M. VanDongen, and C. Henriquez, "Biologically realizable reward-modulated hebbian training for spiking neural networks," *Proc. International Joint Conference on Neural Networks, Hong Kong*, pp. 1781–1787, 2008.

[2] C. M. A. Pennartz, "Reinforcement learning by hebbian synapses with adaptive thresholds," *Neuroscience*, vol. 81, no. 2, pp. 303–319, 1997.

[3] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity," *Neural Computation*, vol. 17, pp. 2337–2382, 2005.

[4] J. P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural Computation*, vol. 18, pp. 1318–1348, 2006.

[5] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.

[6] R. Siegel, "Land mine detection," *Instrumentation Measurement Magazine, IEEE*, vol. 5, no. 4, pp. 22 – 28, dec 2002.

[7] T. Weigel, J.-S. Gutmann, M. Dietl, A. Kleiner, and B. Nebel, "Cs freiburg: coordinating robots for successful soccer playing," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 685 – 699, oct 2002.

[8] S. Ferrari, "Multiobjective algebraic synthesis of neural control systems by implicit model following," *Neural Networks, IEEE Transactions on*, vol. 20, no. 3, pp. 406 –419, march 2009.

[9] D. Culler, D. Estrin, and M. Srivastava, "Guest editors' introduction: Overview of sensor networks," *Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[10] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. shiuan Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," 2002.

[11] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," *Advances in Neural Information Processing Systems*, vol. 9, pp. 211–217, 1997.

[12] G. Hugh, M. Laubach, M. Nicolelis, and C. Henriquez, "A simulator for the analysis of neuronal ensemble activity: application to reaching tasks," *Neurocomputing*, no. 0, pp. 847 – 854, 2002, computational Neuroscience Trends in Research 2002.

[13] N. Maheswaranathan, S. Ferrari, A. M. VanDongen, and C. Henriquez, "Emergentburstingandsynchronyincomputersimulationsofneuronalcultures," *Frontiers in Computational Neuroscience*, vol. 6, no. 15, pp. 1–11, 2012.

[14] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *NanoLetters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[15] I. Ebong and P. Mazumder, "Memristor based stdp learning network for position detection," in *Proc. of the International Conference on Microelectronics*, Cairo, Egypt, 2010.

[16] J. J. B. Jack, D. Nobel, and R. Tsien, *Electric Current Flow in Excitable Cells, 1st ed.* Oxford, UK: Oxford University Press, 1975.

[17] A. L. Hodgkin and A. F. Huxley, "A quantitative description of ion currents and its applications to conductance and excitation in nerve membranes," *Journal of Physiology*, vol. 117, pp. 500–544, 1952.

[18] H. Burgsteiner, "Imitation learning with spiking neural networks and real-world devices," *Engineering Applications of Artificial Intelligence*, vol. 19, no. 7, 2006.

[19] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, "Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition," *Neural Networks*, vol. 41, no. 0, pp. 188 – 201, 2013.

[20] S. M. LaValle, "Planning algorithms," 2004.

[21] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, UK: Cambridge University Press, 2006.

[22] E. D. Schutter, *Computational Modeling Methods for Neuroscientists*, 1st ed. The MIT Press, 2009.

[23] T. Honore, J. Lauridsen, and P. Krogsgaard-Larsen, "The binding of [3h]ampa, a structural analogue of glutamic acid, to rat brain membranes," *Journal of Neurochemistry*, vol. 38, no. 1, pp. 173–178, 1982.

[24] X. Zhang, G. Foderaro, C. Henriquez, A. M. J. VanDongen, and S. Ferrari, "A radial basis function spike model for indirect learning via integrate-and-fire sampling and reconstruction techniques," *Advances in Artificial Neural Systems*, p. 16 pages, 2012.

[25] H. G. Feichtinger, "Approximate reconstruction of bandlimited functions for the integrate and fire sampler," *Advanced Computational Mathematics*, p. 12, 2010.

[26] X. Zhang and Z. Xu, "navigation of virtual insect in different terrains." [Online]. Available: http://fred.mems.duke.edu/silvia.ferrari/downloadables/proposals/

[27] P. Arena, L. Patane, V. Stornanti, P. S. Termini, B. Zapf, and R. Strauss, "Modeling the insect mushroom bodies: Application to a delayed match-to-sample task," *Neural Networks*, vol. 41, no. 0, pp. 202 – 211, 2013.