Proceedings of the 14th IEEE
International Conference on Nanotechnology
Toronto, Canada, August 18-21, 2014

# Digital Implementation of a Spiking Neural Network (SNN) Capable of Spike-Timing-Dependent Plasticity (STDP) Learning

Di Hu, *Student Member, IEEE*, Xu Zhang, Ziye Xu, Silvia Ferrari and Pinaki Mazumder, *Fellow, IEEE*

***Abstract*—The neural network model of computation has been proven to be faster and more energy-efficient than Boolean CMOS computations in numerous real-world applications. As a result, neuromorphic circuits have been garnering growing interest as the integration complexity within chips has reached several billion transistors. This article presents a digital implementation of a re-scalable spiking neural network (SNN) to demonstrate how spike timing-dependent plasticity (STDP) learning can be employed to train a virtual insect to navigate through a terrain with obstacles by processing information from the environment.**

## I. Introduction

The neural network paradigm has recently been drawing increasing interest in the circuit design field because of its massive parallelism and potential scalability. Neuromorphic circuits, which are usually composed of integrate–and–fire (I&F) neuron models [1] - [2], can be applied to solve problems in many areas, including position detection [3], robotic games [4], pattern recognition [5] and path recognition [6]. Although these problems can also be solved using a general purpose processor with software solutions, dedicated circuits have many advantages over general purpose processors. For example, a dedicated circuit is more power-efficient and needs less area for implementation compared with a general purpose processor. This paper presents an indirect training algorithm through which a learning agent can be trained to find targets and avoid obstacles. The training algorithm is based on spike timing-dependent plasticity (STDP), which has been proven suitable as a solution to learning/training problems by previous investigations [7], [8]. According to the STDP rule, weights are adjusted based on the time difference between pre-synaptic and post-synaptic spikes. The main advantages of applying the STDP rule in the design is that the weights do not need to be accessed directly and the learning agent can be trained indirectly. This design allows a learning agent to be trained by receiving signals from its external components without accessing any of its internal components. In addition to indirect training, direct training is widely used in many algorithms. However, under certain circumstances, e.g., when tuning devices implanted in human bodies, direct training is not feasible. In this case, indirect training becomes the only option.

*Resrach supported by NSF under ECCS Grant 1059177.

Di Hu and Pinaki Mazumder are with the University of Michigan, Ann Arbor, MI 48109 USA (phone: 734-763-2107; e-mail: {hudi, mazum}@umich.edu).

Xu Zhang, Silvia Ferrari and Ziye Xu are with Duke University, Durham, NC 27708 USA. (e-mail: {xz70, sferrari}@duke.edu, dec.ziye@gmail.com).

This paper presents a virtual insect model that can be trained by the proposed indirect training algorithm, and can then find a randomly given target on a map. The virtual insect is based on a re-scalable neural network consisting of an input layer, an output layer and a hidden layer. The SNN is re-scalable because the training algorithm has no requirement on the size of each layer or the forms of connections between neurons within one layer or neurons of different layers. In this article, Section II introduces the virtual insect model. Section III introduces the training algorithm used in this research while Section IV discusses digital circuit implementation of the design. Finally, Section V shows the test results of the implemented design.

## II. Virtual Insect Model Based on STDP

In this research, the weights were adjusted through STDP. STDP modifies synaptic weights based on the time difference between the arrival of a pre-synaptic spike and the arrival of a post-synaptic spike. As Fig.1 shows, depending on the time difference, the weight can be either increased or decreased by a certain value, which varies exponentially with the time difference.

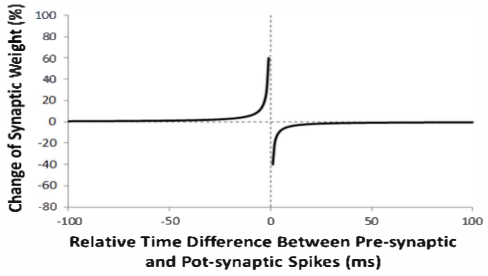


Fig. 1. An Illustration of Spike Timing-Dependent Plasticity

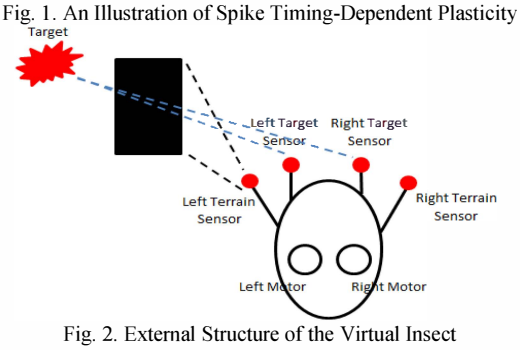


Fig. 2. External Structure of the Virtual Insect

A virtual insect model was constructed to demonstrate the proposed algorithm and design. The virtual insect is initially trained off line. After being well trained, the virtual insect can find a given target on any map with any form of obstacles. Fig. 2 shows the external structure of a virtual insect, which consists of a target sensor, a terrain sensor and a motor on

each of its right and left sides. The target sensor generates a signal $S_{target}$ based on the distance between the sensor and the target. The further the virtual insect is from the target, the stronger $S_{target}$ will be. The terrain sensor generates a signal $S_{terrain}$ based on the roughness of the map, with a rougher map corresponding to a stronger $S_{terrain}$. The two motors control the movement of the insect.

The SNN architecture illustrated in Fig. 3, which includes one hundred input neurons, two hundred and fifty-six hidden neurons and fifty output neurons, was implemented to demonstrate the concept of the indirect training algorithm. The structure of the SNN is flexible (i.e. each layer can have any number of neurons and can be of any shape, and the connections between neurons can be arbitrary) as long as there are input layer, hidden layer and output layer as demonstrated in Fig. 3.

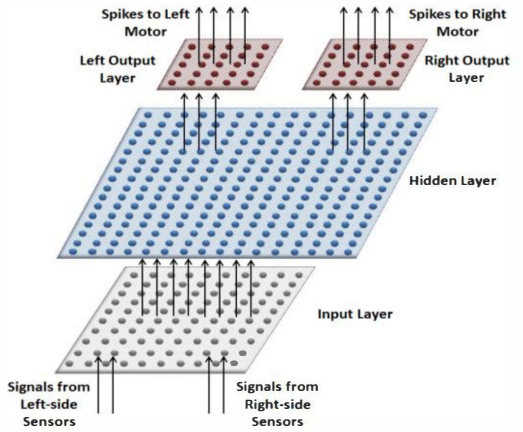


Fig. 3. The Structure of the SNN
Arrows demonstrate the direction of spikes.

The speed of the motors and the speed of the insect are determined by equation (1). $V_x$ and $V_y$ are the speeds of the insect along the x-axis and the y-axis, respectively. V is the linear speed of the insect, and $V_L$ and $V_R$ are the speeds of the left and right motors, respectively. $\theta$ is the angle between the middle axis of the insect and the x-axis. L, $\tau_{motor}$ and $\eta$ are scaling constants, and $t_L^f$ and $t_R^f$ are the firing times of the output neurons. According to equation (1), a motor always decelerates unless its corresponding output neuron fires a spike. Therefore, the more frequently an output neuron fires, the faster the speed of the corresponding motor. Because the firing frequency of an output neuron depends on the weights carried by the synapses connecting it to its corresponding input neurons, training the insect is essentially equivalent to adjusting the weights to the proper values.

$$\begin{cases} V_x = V^* * \cos(\theta) \\ V_y = V * \sin(\theta) \\ V = \frac{V_L + V_R}{2} \\ \Delta\theta = \frac{V_R - V_L}{L} \\ \Delta V_L = -\frac{V_L}{\tau_{motor}} + \eta * (t == t_L^f) \\ \Delta V_R = -\frac{V_R}{\tau_{motor}} + \eta * (t == t_R^f) \end{cases} \quad (1)$$

During the training phase, training signals are injected into the input neurons. Table I shows all of the 12 possible cases of sensor signal combinations. As training signals are generated for each case, there are 12 sets of training signals.

TABLE I. 12 SENSOR SIGNAL CASES

| Case # | L Terrain | R Terrain | L Target | R Target |
|---|---|---|---|---|
| 1 | Plain | Plain | Strong | Weak |
| 2 | Plain | Rough | Strong | Weak |
| 3 | Rough | Plain | Strong | Weak |
| 4 | Rough | Rough | Strong | Weak |
| 5 | Plain | Plain | Equal | Equal |
| 6 | Plain | Rough | Equal | Equal |
| 7 | Rough | Plain | Equal | Equal |
| 8 | Rough | Rough | Equal | Equal |
| 9 | Plain | Plain | Weak | Strong |
| 10 | Plain | Rough | Weak | Strong |
| 11 | Rough | Plain | Weak | Strong |
| 12 | Rough | Rough | Weak | Strong |

## III. TRAINING ALGORITHM

In general, the actual training process can be divided into the following steps.

1. According to each of the 12 cases in Table I, a set of terrain and target signals is generated.

2. In each case, the desired firing frequencies for both the left and right output layers are calculated using equation (2). In equation (2), $\nu_L$ and $\nu_R$ are the desired firing frequencies of the output motor neurons and $S_{trL}, S_{tL}, S_{trR}, S_{tR}$ are the left terrain sensor input, the left target sensor input, the right terrain sensor input and the right target sensor input, respectively. $C_1$ is set to be larger than $C_2$ to prioritize the terrain sensors.

$$\begin{pmatrix} \nu_L \\ \nu_R \end{pmatrix} = \begin{pmatrix} S_{trL} & S_{tL} \\ S_{trR} & S_{tR} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \quad (2)$$

3. In each case, the difference between the actual firing frequency and the desired firing frequency is calculated and an arithmetic average difference for the total 12 cases is calculated using equation (3). In equation (3), $e(t_l)$ is the average difference at time $t_l$, $\nu_i^{p^*}$ and $\nu_i^p$ are the desired and actual firing frequencies in each case, and $P_m$ is the count of total cases.

$$e(t_l) = \sum_{i \in O} \sum_{p=1}^{p_m} \frac{\sqrt{(\nu_i^{p^*} - \nu_i^p)^2}}{2 * P_m} \quad (3)$$

4. Based on the change of the average difference between the actual firing frequencies and desired target firing frequencies, the training signals for the next iteration are generated. In general, when the average difference decreases, the training signals will remain unchanged. If the average difference increases, the order of the training signals will flip.

5. The training signals are injected into the some randomly selected input neuron pairs, and a new iteration is started. When the average difference *e* is sufficiently small, the insect has been well trained.

## IV. CMOS Circuit Implementation

A CMOS circuit was implemented using 130-nm CMOS technology. The full chip has an area of 6.5 mm (width) by 4.8 mm (height) and a frequency of 22.5 MHz. The implementation was accomplished through synthesis and auto placement and routing. Fig. 4 shows the floor plan and data flow of the designed chip. Due to the large amount of internal signals, a signal mux and a training signal generator were added to the chip. As shown in Fig. 4, the top level I/O signals include the target position and the insect position because there are no actual sensors or motors. For a complete design, these pins will be replaced by sensor signals and spikes from output neurons.

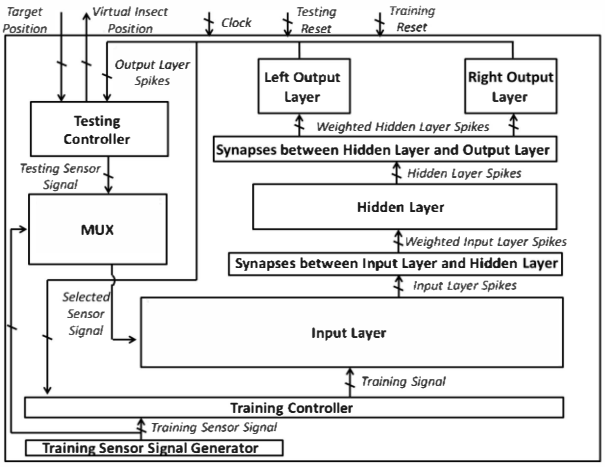


Fig. 4. Floor Plan and Data Flow of the Designed Chip

Table II shows the layout area of each major block and full the chip without pads. As shown in Table II, the input layer is much larger than the other layers. This is because the training (i.e. wrights adjustment) only happens in the input layer. As a result, only neurons and synapses in the input layer need to have the full functionalities. Neurons and synapses at other locations can be simplified and require less area.

TABLE II. Layout Area of the Design

| Block | Area |
|---|---|
| Input Layer | 4600 um * 1600 um |
| Hidden Layer | 3750 um * 500 um |
| Output Layer | 800 um * 200 um * 2 |
| Testing Controller | 900 um * 400 um |
| Training Controller | 4900 um * 200 um |
| Control Signal Mux | 950 um * 800 um |
| Whole Chip without Pads | Approx. 5200 um * 4000 um |

Fig. 5 shows the layout of the complete chip with I/O and power pads. Each circled block corresponds to a block on Fig. 4. The full chip has an area of 6.5 mm by 4.8 mm and a frequency of 22.5 MHz. The peak power consumption is about 13.24 mW and the leakage power is 1.06 uW.

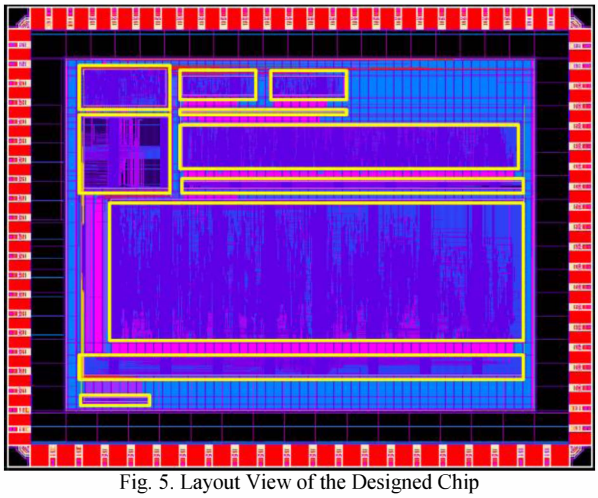

Fig. 5. Layout View of the Designed Chip

## V. Simulations and Results

The indirect training algorithm was implemented in MATLAB, Verilog and CMOS to train the SNN presented in Section II to allow the virtual insect to perform the terrain navigation task. Prior to the hardware implementation, the performance of the trained insect was first evaluated by MATLAB simulations, as demonstrated in Fig. 6, in which the green line and the blue line depict trails of the untrained state and the trained state of the virtual insect, respectively. The results show that after the SNN was fully trained, the virtual bug was capable of avoiding obstacles and obtaining the target position.

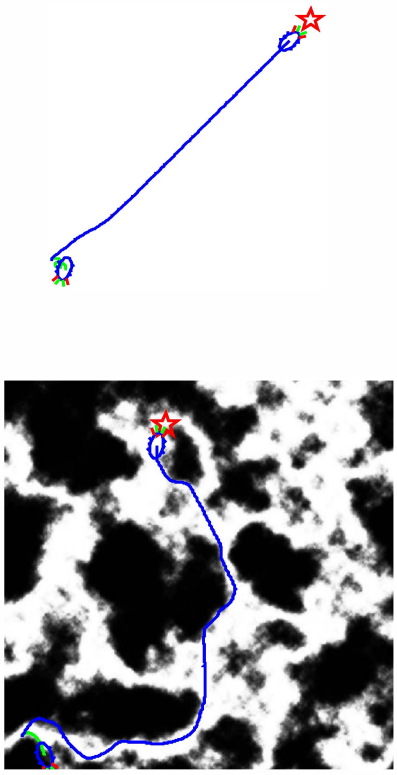

Fig. 6. Trails of the virtual insect on uniform terrain and on terrain populated by obstacles.[9]

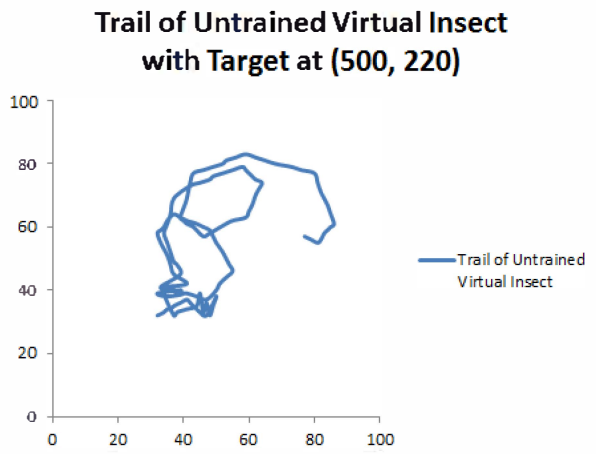


Fig. 7. Trail of an Untrained Virtual Insect

Tests were also performed after implementation. Fig. 7 shows the trail of an untrained virtual insect on a 1000 by 1000 map with a target at (500, 220). As shown by Fig. 7, the untrained virtual insect wandered randomly within a 100 by 100 area, which is consistent with the result shown in Fig. 6.

Fig. 8 shows trails of the virtual insect on a plain map with different targets at (500, 1000), (1000, 500), (1000, 1000), (600, 1400) and (750, 150) on a 1500 by 1500 map. In Fig. 8, the traces represent trails of the virtual insect, and the isolated dots indicate the locations of the targets. As shown in Fig. 8, for any random given target, the well-trained virtual insect was able to move toward the target and stop at a position close to the target. Because the sensor signals were proportional to the distance between the virtual insect and the target, those signals would become zero when the virtual insect was close enough to the target. As a result, the virtual insect would then stop at that location other than the target's position.

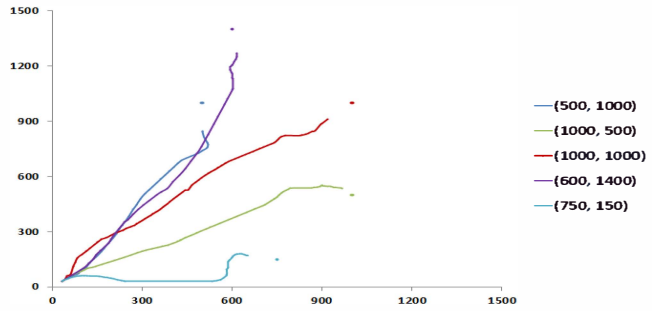


Fig. 8. Trails of the Virtual Insect on a Plain Map

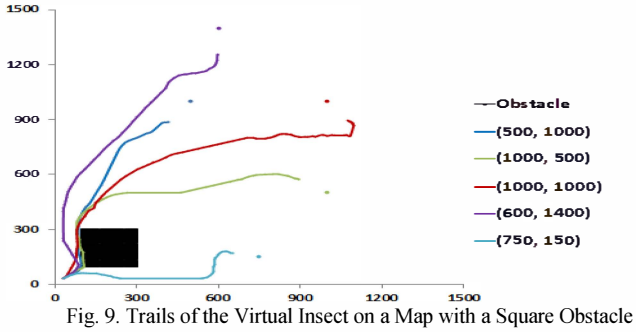


Fig. 9. Trails of the Virtual Insect on a Map with a Square Obstacle

Fig. 9 shows trails of the virtual insect on a map with a square obstacle. Targets were chosen to occur at the same locations as those in Fig. 8 for comparison. As shown in Fig. 9, the well-trained virtual insect changed its path because of the obstacle, even though the target was in the same location. Eventually, the virtual insect was able to stop at a position close to the target. Especially, when target was at (750, 150), the virtual insect did not pass the area of the obstacle, thus the traces are the same in Fig. 8 and Fig. 9.

## VI. Conclusion

This article presents a hardware implementation of an SNN with an indirect training algorithm. A virtual insect model was developed as an example to demonstrate how this approach can be used to solve practical problems. Hardware implementation was achieved using Verilog and CMOS technology. Future work may include building a complete virtual insect with sensors and motors or expanding the virtual insect model to solve other types of problems, such as pattern recognition and robotic games.

## *REFERENCES*


[1] R. J. Vogelstein, U. Mallik, and G. Cauwenberghs, "Silicon spike-based synaptic array and address-event transceiver," in *Proc. IEEE Int. Symp. Circuits and Systems*, 2004, pp. 385–388.

[2] E. Chicca, D. Badoni, V. Dante, M. D'Andreagiovanni, G. Salina, S. Fusi, and P. Del Giudice, "A VLSI recurrent network of integrate–and–fire neurons connected by plastic synapses with long term memory," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1297–1307, Sep. 2003.

[3] Ebong, I.E.; Mazumder, P., "CMOS and Memristor-Based Neural Network Design for Position Detection," *Proceedings of the IEEE* , vol.100, no.6, pp.2050,2060, June 2012

[4] Weigel, T.; Gutmann, J.-S.; Dietl, M.; Kleiner, A.; Nebel, B., "CS Freiburg: coordinating robots for successful soccer playing," *Robotics and Automation, IEEE Transactions on* , vol.18, no.5, pp.685,699, Oct 2002

[5] Le Dung; Mizukawa, M., "A Pattern Recognition Neural Network Using Many Sets of Weights and Biases," *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on* , vol., no., pp.285,290, 20-23 June 2007.

[6] Shinzato, P.Y.; Fernandes, L.C.; Osorio, F.S.; Wolf, D.F., "Path recognition for outdoor navigation using artificial neural networks: Case study," *Industrial Technology (ICIT), 2010 IEEE International Conference on* , vol., no., pp.1457,1462, 14-17 March 2010

[7] Snider, G.S., "Spike-timing-dependent learning in memristive nanodevices," *Nanoscale Architectures, 2008. NANOARCH 2008. IEEE International Symposium on*, vol., no., pp.85,92, 12-13 June 2008.

[8] Arena, P.; Fortuna, L.; Frasca, M.; Patane, L., "Learning Anticipation via Spiking Networks: Application to Navigation Control," *Neural Networks, IEEE Transactions on*, vol.20, no.2, pp.202,216, Feb. 2009.