

Neural Networks for Control of Artificial Life Form

Dan Su

Thesis for degree of MSc by research in Electronics

Department of Electronics

University of York

November 2010

Abstract

An artificial life form under the control of a spiking neural network has been created in a chessboard environment which consists of 60*60 grids using Matlab GUI. The spiking neural network consists of 8 neurons simulated using Izhikevich model which combines the property of both biological plausibility and computational efficiency. The neurons within the network are fully connected with each other. The 'intelligence' of the artificial life form is stored as value of weights in the synaptic connections of neurons. STDP is the learning rule implemented to the network in this project. STDP adjusts the synaptic weights according to the precise timing of pre and postsynaptic spikes.

The artificial life form itself has been designed to complete certain tasks such as avoiding obstacles and catching food in the chessboard. The behavior of the artificial life form under the control of STDP in various situations will be investigated. Experiments will be carried out at the same time trying to improve the behavior of the artificial life form so that the artificial life form can evolve and show some adaption abilities according to the external environments.

Thesis content

1. Introduction	4
1.1 Aim and Objectives	4
1.2 Program Environment	4
1.3 Structure of the thesis	5
2. Background	6
2.1 Generation of action potential inside neurons	6
2.1.1 Elements of neuron system	6
2.1.2 Synaptic integration	6
2.1.3 Action Potential: Generation	7
2.2 Bio-inspired algorithm and learning rules	10
2.2.1 Introduction	10
2.2.2 Direction coding scheme	10
2.2.3 Distance coding scheme	12
2.2.4 Population vector	13
2.3 Noise	14
2.4 Izhikevich model	15
2.4.1 Which model to use?	15
2.4.2 Izhikevich model	15
2.4.3 Regular Spiking and Fast Spiking	18
2.4.4 Chattering	22
2.4.5 Resonator	24
2.5 STDP (Spike Timing Dependent Synaptic Plasticity)	27
2.5.1 What is STDP?	27
2.5.2 Investigation of the behavior of the artificial life form	29
3. Method	30
3.1 Choice of Simulation environment	30
3.1.1 Source of the code	30
3.1.2 Matlab GUI (Graphical User Interface)	30
3.1.3 Program structure	31
3.2 Initial Design	34
3.2.1 Displaying the artificial life form on the screen	34
3.2.2 Making the artificial life form move	35
3.3 Spiking Neural Network Implementation	36
3.3.1 Connections between the neurons	36
3.3.2 Injecting Noise	38
3.3.3 Implementation of the population vector	39
3.4 Environment Implementation	41
3.4.1 Assign different colors to various object	41
3.4.2 Representation of trajectories of the artificial life form	42
3.4.3 Adding signals to food-the implementation of the direction coding scheme	44
3.4.4 Assign field to the signals of the food-implementation of the distance coding scheme	46

3.4.5 Adding 'zero' signal to the obstacles	48
* Testing the ability (efficiency) of the artificial life form overcoming the square obstacle under the different noise conditions (with no STDP)	
* Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under the different noise conditions (with no STDP)	
3.5 Implementation of STDP	52
3.5.1 Value and timing of STDP	52
3.5.2 Potentiation	54
3.5.3 Depression	55
3.5.4 Updating the weight W_{ij}	57
* Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under the control of STDP	
* The weight distribution	
3.5.5 Implementation of Directional Damping	57
*Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under the different noise conditions (with STDP and Directional Damping)	
*Analysis the equilibrium of the weight distribution with Directional Damping	
4. Testing	60
4.1 *Testing the ability (efficiency) of the artificial life form overcoming the square obstacle under different noise conditions (with no STDP)	60
4.2 *Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with no STDP)	62
4.3 *Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with STDP)	64
4.3.1 *Statistical test	65
4.4 *Analysis of the weight distribution	66
4.5 *Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with STDP and Directional Damping)	71
4.5.1 *Analysis of the equilibrium of the weight distribution with Directional Damping	73
4.5.2 *Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with STDP and Directional Damping)	73
5. Discussion	74
6. Future work	77
7. Acknowledgements	79
8. Appendix	80
9. References and Bibliography	120

1. Introduction

1.1 Aim and Objectives

The aim of this project is to study the behaviors of simulated artificial life form under the control of the neural networks. The neural networks will be spiking networks which use sequence of spikes to carry information between neurons. The Izhikevich Model which combines the property of both biological plausibility and computational efficiency has been studied and simulated using Matlab.

This model will be slightly modified and implemented as the basis of the control of the spiking networks. Four key parameters of this model could be used to control the behavior of the spikes in the network; a lot of experience of using this typical model would be gained during this process.

The STDP learning rule which adjusts the weight of the network according to the relative timing of input and output spikes will be implemented into the Izhikevich Model in order to increase the intelligence of the neural network. Investigations on the effect of the learning rule will be carried out with a number of experiments testing the behavior of the artificial life form under the control of STDP.

1.2 Program Environment

The artificial life form is operating in a 2-D chess board like environment; the environment will be simulated using Graphical User Interface (GUI) in Matlab. The program code is based on a web source downloaded from Matlab-Central and modified to fit the project use, so that the implementation time could be saved and emphasis be placed on the neural network part.

The artificial life form is designed to catch food in the chess board while trying to avoid different obstacles when the program starts. A number of rules are added to the simulation environment, and some of them are taken from ideas inspired from the real biological world such as the neural coding of movement in animals. Investigations of the behavior of artificial life form will be carried out simultaneously, enabling the complexity of the network and environment to be increased gradually.

With a combination of internal network learning rules and external environmental rules, the artificial life form should have a high degree of interactions with its external environment, and it should show some adaption abilities under different conditions. The use of STDP (Spike Timing Depend Plasticity) will also improve the performance of the artificial life form so that it can avoid the obstacles and catch the food faster. Investigations will be emphasized on this section and there will be analysis and discussions on the experiment results of the effect of STDP.

The graphical user interface window will also be constructed carefully to facilitate the process of experiment; so that the user could control a number of key parameters in the network and environment allowing more space for testing.

1.3 Structure of the thesis

Detailed background information will be described in Chapter 2 of the thesis, which includes the basis of biological and engineering theory behind the project, the analysis of Izhikevich model and an introduction of noise and the STDP learning rule. In Chapter 3, the thesis will show the complete project method from programming of the simulation environment using Matlab GUI to each part of the program implementation process including STDP. Various Experiments will be carried out with data collected from the program while the implementation goes deeper into the project. The demonstration of all the testing results using plots and diagrams will be made in Chapter 4. Chapter 5 will discuss the results. Any works that can be done in future will be mentioned in Chapter 6.

2. Background

2.1 Generation of action potential inside neurons

2.1.1 Elements of neuron system

It is very important to get some ideas of a real (ideal) neuron in a human brain from a biological point of view before creating its model. A typical ideal spiking neuron structure is shown below in Figure 1

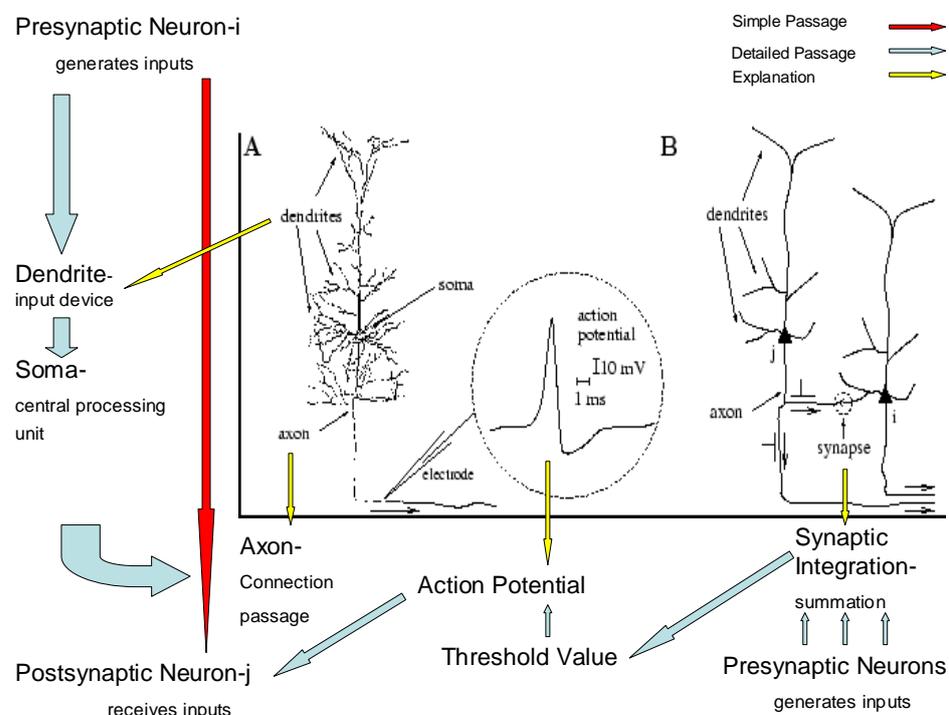


Figure 1 taken from reference [1]

Figure 1 above shows that a real typical neuron can be divided into three functionally distinct parts called dendrites, soma and axon. The dendrite plays the role of the 'input device' that collects signals from other neurons and transmits them to the soma. The soma is the 'central processing unit' that performs an important non-linear processing step. This non-linear processing step is simulated using a spiking neural network model in this project, which will be introduced later in Section 2.4 of the thesis. In general, a neuron can process information received from thousands of other neurons. If the total input exceeds a certain threshold, then an output signal (assumed to be in the form of a spike in the model of this project) is generated. This whole process is represented as an action potential which is shown in the big dashed circle in Figure 1; it is a short voltage pulse of 1-2 ms duration and amplitude of about 100 mV. The biological view of the generation of the action potential will be described in details later in Sections 2.1.3. The output spike generated from neuron i then passes through the 'connection passage' -axon to neuron j . The 'connection passage' is referred to the term 'synaptic integration' which will be introduced in the next subsection.

2.1.2 Synaptic integration

The term *synaptic* refers to the specialized sites for communication between neurons, named *synapse*, where input signals to a neuron are generated [2]. The synapse is marked by the small dashed circle in Figure 1. The neurons that generate input signals are called a *presynaptic neurons* and the neuron that receives this input is called the *postsynaptic neuron*. Integration refers to the way inputs from many presynaptic neurons are processed in order to generate the action potential in the postsynaptic neuron. Synaptic integration, therefore, describes the summation by the postsynaptic neuron of inputs from many presynaptic neurons [1].

The synaptic integration can be represented as a multiple input to a single output system. So the external input generated by presynaptic neurons can be divided into several small input currents which sum up to form the total input currents $I(t)$:

A structural model of the system is shown below:

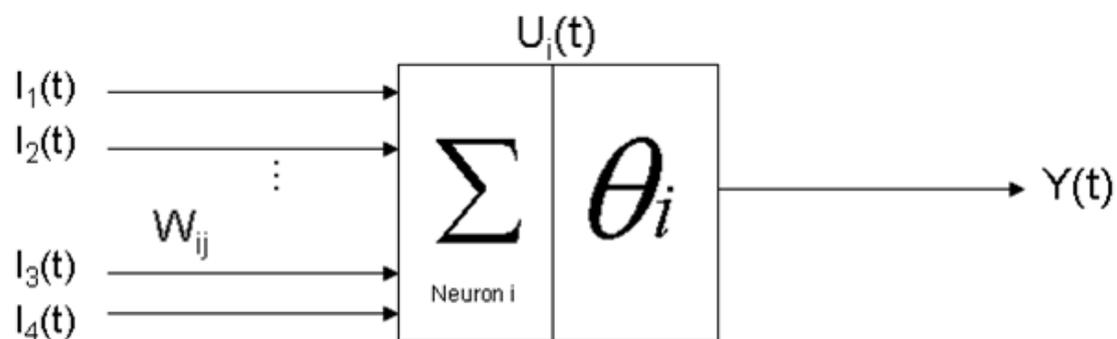


Figure 2

In Figure 2, the total input currents $I(t)$ are the sum of all the external input $I_i(t)$. The factor W_{ij} is a measure of the efficacy of the synapse from neuron j to neuron i . $U_i(t)$ is the internal state of neuron i . θ_i is the threshold needed for the neuron to produce an action potential.

The detailed simulation of synaptic connections and their implementations will be explained later in Section 2.31 of Chapter 3.

2.1.3 Action Potential: Generation

Section 2.1.2 gives an overview on how information is passing through between each distinct part of a real typical neuron from a systematic point of view. Now in this section a more detailed look at how information can pass through the real neurons will be presented--the generation of action potential. The neurons cannot pass any information without an action potential, so how is action potential generated? Depolarizing the membrane potential and passing a critical 'threshold' voltage will result in an action potential. Membrane potential is the electrical potential difference between the inside and outside of a cell [3].

A set of graphs with voltage against time showing a complete process of action potential from beginning to the end are listed in the next page. The relationships between the generation of action potentials and the inward and outward flow mechanism of ions are also described in Figure 3 below.

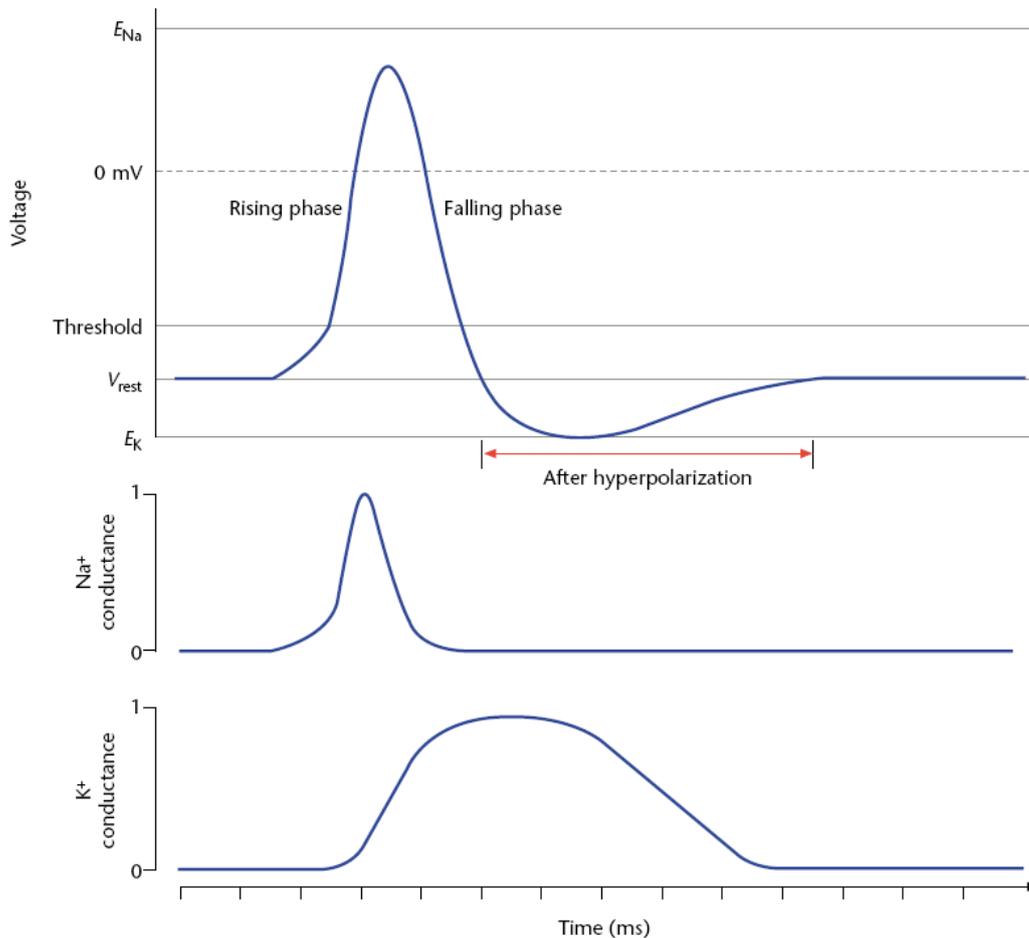


Figure 3 taken from reference [4]

Generally, if a neuron is at rest, its membrane potential is about -65mV . This is the resting potential of a neuron, and it is labeled as V_{rest} on top trace of Figure 3. The resting potential is kept by the total charge of different ions (Mainly sodium Na^+ and potassium K^+) inside and outside of the cell. Roughly speaking, the inward and outward movement of these ions through the membrane is the main reason that causes the depolarization of the membrane potential.

When a neuron receives an “input”, that usually means there will be an increase in the Na^+ influx that changes the membrane potential and activates neighboring Na^+ channels. A threshold value about -50mV (observed from Figure 3) is reached when the amount of Na^+ entering the cell is greater than the resting efflux of K^+ .

As more Na^+ channels open, conductance continues to increase until all available Na^+ channels are open. This is shown on the middle trace of Figure 3. The increase in selective permeability (conductance increase) to Na^+ causes the membrane potential to depolarize toward E_{Na} (The Equilibrium potential for Na^+ in neurons, which is about $+60\text{mV}$), thereby producing the rising phase of the action potential [4]. All of these events occur within duration of less than one millisecond -- the time it takes Na^+ channels to activate.

After channels are open for about a millisecond, they inactivate by the pore-blocking mechanism

leading to a decrease in sodium conductance. Sodium channel inactivation is the first step in action potential termination; the decrease in selective permeability to Na^+ causes the membrane potential to shift away from E_{Na} and towards the normal resting membrane potential [4]. This part of the action potentials is called the 'falling phase'. Although Na^+ channels inactivation could, by itself, terminate the action potential, K^+ channel activation provides a fail-save mechanism to terminate the action potential [4].

The bottom trace on Figure 3 shows that K^+ channel activates with a slower rate and a delay compared to the Na^+ channel activation. E_k (The Equilibrium potential for K^+ in neurons, which is about $-60\sim-70\text{mV}$) is near the resting membrane potential, so it is easy to see that a conductance increase to K^+ will tend to drive the membrane potential towards more negative voltages, thus contributing to the falling phase of the action potential.

The contribution of K^+ channels to action potential termination is also evident by the presence of a period of hyper polarization, during which the membrane potential is briefly more negative than the resting membrane potential. This arises under conditions where E_k is more negative than V_{rest} , so an increase in K^+ conductance hyperpolarizes the membrane potential. The after-hyper polarization also arises because K^+ channel inactivation is slower than Na^+ channels inactivation. As K^+ channels finally inactivate, the after- hyper polarization declines to the resting potential [4].

The detailed simulation of action potential using spiking neural network model--the Izhikevich model will be introduced later in Section 2.4 of this chapter.

2.2 Bio-inspired algorithm and learning rules

2.2.1 Introduction

In academic world, Neural Robotics has two main research applications. One is to create autonomous artifacts or intelligent systems which inherit some or part of the brain's functions in order to help people solve real-world problems. Human brain consists of large-scale of neural networks with complex structures and configurations. The investigation of brain is one of the foremost challenges within neuroscience today, especially within the area of cognitive neural science [5]. There are thousands of research studies on human cognitive function in the context of brain. The relationships between the biomedical engineering, cognitive neural science, and artificial intelligence have become more closely related ever than before. Research in these areas also greatly stimulates the development of Neural Robotics [5].

The other application of Neural Robotics is to construct complex neural models and architectures to investigate the basic principles of the brain and its relationships with behaviors. Various neural network models have been created so far, which always have adopted different algorithms and learning rules being inspired by the biological behavior of living systems. The robot under control of these neural networks can perform vast amount of interesting behaviors. The study of these behaviors can promote the development of artificial neural networks. The applications of Neural Robotics are playing more and more important roles in research of artificial intelligence.

2.2.2 Direction coding scheme

In this project, a 60*60 grid chess board-like environment is simulated using Matlab. Objects within the grid of environment have been set with different colors representing obstacles, food and artificial life form. The artificial life form is designed to avoid obstacles in order to catch food. A set of ideas taken from research studies in cognitive neural science have been applied to generate the algorithms to control this artificial life form. These ideas are inspired from a set of experiments on the physiological analysis of motor pathways of animals, which are used investigate the neural coding of movement.

One example is taken from the research paper "A Model of Spatial Map Formation in the Hippocampus of the Rat" written by K. I. Blum and L. F. Abbott. By using experimental facts about long-term potentiation (LTP) and hippocampal place cells in the rat, they model how a spatial map of the environment can be created in the rat hippocampus [18]. Sequential firing of place cells during exploration induces, in the model, a pattern of LTP between place cells that shift the location coded by their ensemble activity away from the actual location of the animal [18]. These shifts provide a navigational map that, in a simulation of the Morris maze, can guide the animal toward its goal [18]. In their paper, they also mentioned the previous research which suggested the cognitive map of the spatial environment of the rat is stored by potentiated synaptic weights representing both spatial and temporal correlations [18]. Inspired from these ideas, the thesis also uses synaptic weights to represent the spatial information of the artificial life form's external environment.

Another example is learned from the experiments by Apostolos Georgopoulos and his colleagues at Johns Hopkins University to record the activity of cells in various regions of the motor system

of rhesus monkeys [6]. In those experiments, the animals are trained to perform a set of tasks under certain conditions; neurophysiologists investigate the activities of single cells in animal motor cortex while they are moving, and ask what parameters of movement are coded by such cellular activity. They observe a hierarchical coding of movement across the motor system; some cells are directly related to the activation of specific muscles, some cells are coding for movement directions and some cells are even coding for more abstract concepts such as goals of action depending on their hierarchical levels in the central nervous system.

The data from these experiments also indicate that motor cortex cells code movement direction and that movement in a certain direction requires activation of appropriate cells. When the animals perform specific actions, the spikes in the corresponding coding neuron are produced more frequently than when it is at rest. This means that the number of spikes has an influence on the movement of animals.

Inspired by these ideas, this project is going to add a directional coding scheme to this spiking neural network. Eight neurons are used to represent eight different directions which are shown in Figure 4.1

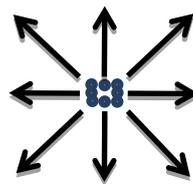


Figure 4.1 Eight Neurons Represent Eight Different Directions

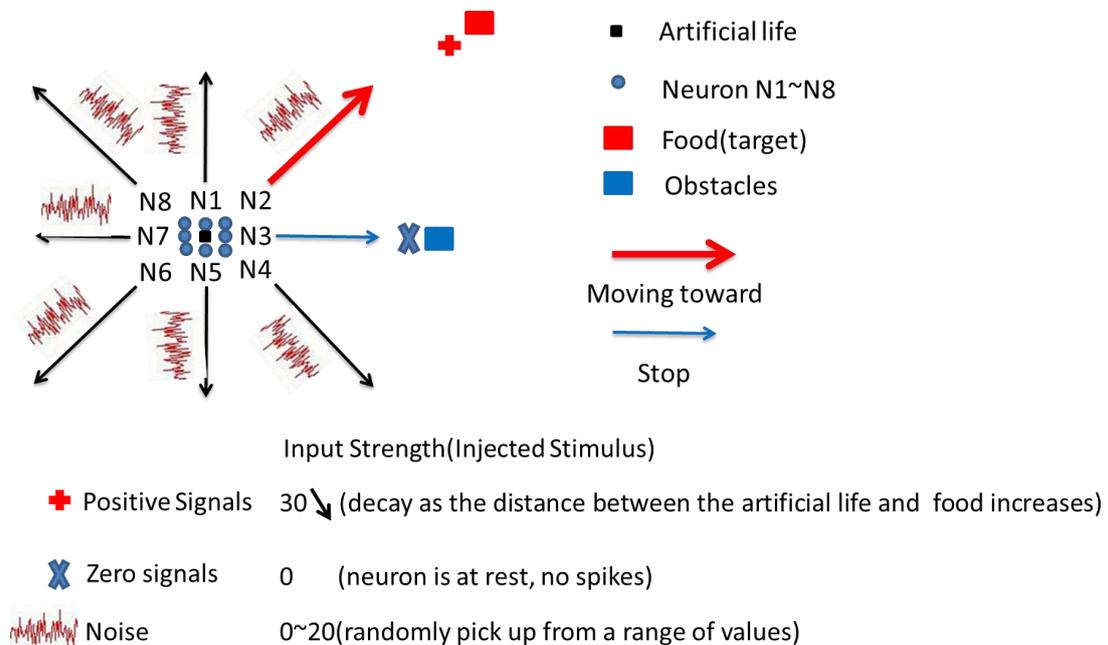


Figure 4.2 Directional coding scheme of the spiking neural network

The artificial life form receives signals from its external environment before each real-time movement. A food target sends positive signals while an obstacle itself represents a 'zero' signal.

A positive signal acts as an electric DC input and triggers additional spikes (Action Potentials) in the neuron which points to the direction of the food target. A 'zero' signal brings an excitatory neuron back to its resting state if this neuron is pointing to the signal's direction and the artificial life form is just about to touch an obstacle. There will be more details about issues of input strength in the later chapters.

The program then realizes the direction coding scheme by counting and comparing the number of spikes that occur within each neuron during a fixed time window. These fixed time windows consist of only simulation time-steps, but it does take a lot of times for computer to execute all the calculation and simulation commands. Unlike the conventional analogue neural networks which compute all the inputs simultaneously and pass the outputs through layers of neurons, the outputs of the spiking neural networks mainly rely on the relative timing of the spikes in pre and post synaptic neurons. In fact, if the time window consists of too many time-steps, the computational efficiency (speed) of the spiking networks will be greatly reduced. However, the length of the time window should also be kept long enough allowing sufficient timing space for spikes so that they could carry sufficient information. Therefore, timing will be a very important issue in spiking neural networks; there should be a compromise between the efficiency and sufficiency. A number of tests are run on this issue, which will be discussed in the first section in Chapter 4. Currently the program takes 400 simulation time steps as the length of the time window. Detailed timing process of the spiking network is shown in Figure 5 below.

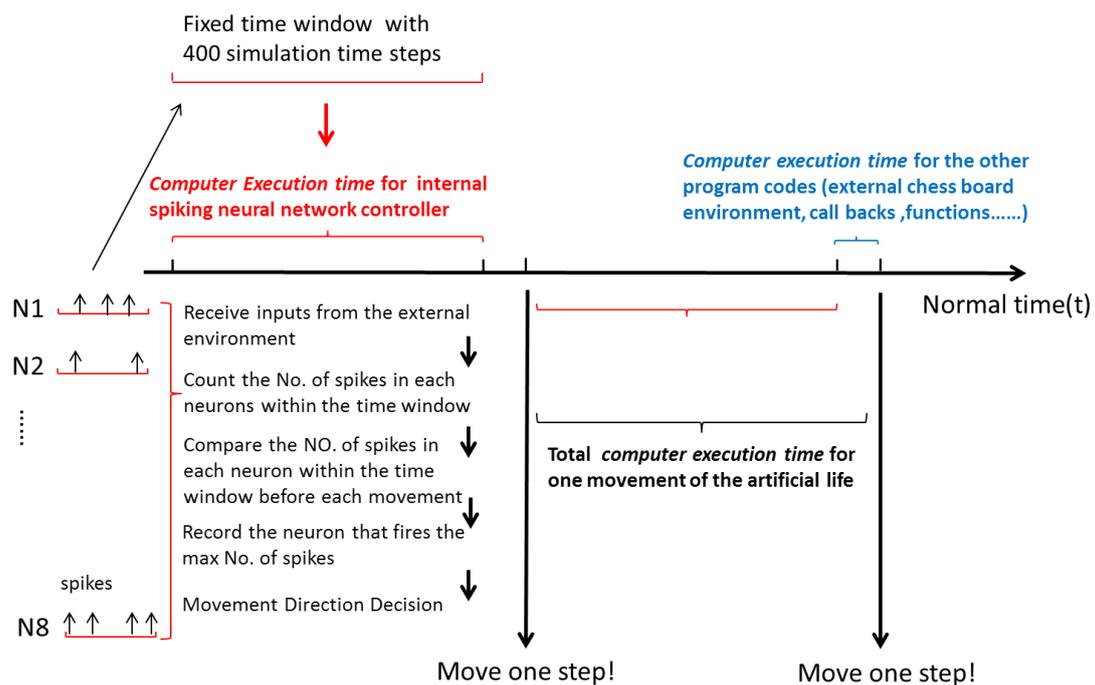


Figure 5 Detail timing process of the spiking network

In addition to the hardware issue such as the speed of the computer CPU, it is very obvious in Figure 5 that the total execution time for one movement of the artificial life form greatly depends on the length of the time window. The less simulation steps it has, the quicker the spiking neural network controller is going to execute the program code. However, as mentioned above, the time window should also be kept long enough allowing sufficient timing space for spikes so that they could carry sufficient information.

By comparing the number of spikes that occur in each neuron during the time window, the spiking neural network controller can know the neuron that fires the maximum number of spikes. The direction that this neuron is pointing to would be the direction that the artificial life form is going to move toward. As a result, a food target sending positive signals will attract the artificial life form since it triggers additional spikes in the neuron that points to its direction. An obstacle will stop the artificial life form from touching the food target because it makes the neuron that points to its direction maintain the resting potential so that no spikes will be triggered in that neuron.

2.2.3 Distance coding scheme

In spite of the direction coding scheme, a distance coding scheme is also necessary for this spiking network. The distance coding scheme is added as a rule of the external environment, which is implemented separately from the central neural controller. The bottom of Figure 4.2 shows that the signal strength of the food is decaying as the distance increases (The equation will be given in Chapter 3), which is the core rule that governs the distance coding scheme. With this distance coding scheme, the artificial life form will move towards the food located closer to it. A full explanation of the distance coding scheme can be found in Chapter 3.

2.2.4 Population vector

When there is more than one food target or with effect of noise (see bottom of Figure 4.2), it is possible that more than one neuron will spike the same maximum number of times within one particular time window. An alternative coding scheme also inspired from the observation of above experimental data is used to solve this problem. The data from the experiment shows that the command of monkey movement in a direction turns out to be the summed activity over all of the cells devoted to a certain limb. This coding scheme has been called the *population vector* because it is a way to see how a global event, a movement in a certain direction, can result from the summed activity of many small elements, each contributing to its own vote [6].

The experiment on hippocampal of rat also demonstrates the concept of population vectors. “We assume that the model network contains a large number of neurons with overlapping place fields. Thus, if the animal is at a location x many neurons with place field center close to x are active at the same time” [19]

By treating all the dominant neurons as a population vector, the network controller combines the vectors and calculates the overall movement direction for the artificial life form to go into the next grid square. For example, in Figure 2, if neuron N8 and N6 fire the same maximum number of spikes within any particular time window, the network will treat it as only N7 fires; If N8, N4, N2 fire the same maximum number of times, the network will treat it as only N2 fires with N8 and N4 cancelling each other out; If N1 and N6 fire the same maximum number of times, the network will treat it as only N7 fires. If N1 and N5 fire, the network will treat it as no neuron fires and the artificial life form will remain still in its original position.

Now, problems also arise with this coding scheme. Firstly, what can be done if neuron N1 and N5

always fire the same maximum number of spikes and the artificial life form remains still all the time? Secondly, what happens if N1 and N2 fire the same maximum number of times? They cannot cancel each other out and the artificial life form can only move one grid each turn for the convenience of collecting data. These problems can be solved by introducing the noise into the spiking network, which will be discussed in the next section.

2.3 Noise

There is noise in all real life systems. In this project, the noise is implemented into the spiking network to give the artificial life form some degree of randomness. There are situations when two possible and equally desirable movements are available. The situation that N1 and N5 always fire the same maximum number of spikes and the artificial life form remains still for long time is one of the problems that may be caused. To solve this problem, an unfixed input stimulus selected from a range of values (see bottom of Figure 4.2) will be randomly injected to any of the 8 neurons before each movement. This project uses the Matlab command 'rand' to generate uniformly distributed pseudorandom numbers from 0 to 20mV. That is one way of introducing noise into the spiking network, which will cause unexpected spikes to occur within neurons (the dynamic details of how injected stimulus causes additional action potentials(spikes) in neuron using Izhikevich models will be illustrated in the next section). As a result, the chance of N1 and N5 always firing the same maximum number of spikes will be greatly reduced. Increasing the strength of noise (the mean and the fluctuations) also increases the number of unexpected spikes and therefore increases the degree of randomness of the artificial life form.

However, the strength of noise should also be limited due to various reasons. The first reason is to prevent the model from saturating. If a model is in saturation, it will not produce any additional spikes no matter how strong the input is applied to the neurons. The second reason is that a very strong noise will also reduce the efficiency of the artificial life form, which means that more movements will be taken by the artificial life form to catch the food. The compromise between the effect of noise and the signal sent from the external environment will be investigated in the first section of Chapter 4.

Under a limited level of noises coupled with a limited degree of randomness, the artificial life form should be able to avoid small and simple obstacles and reach the final food target within a relatively efficient time. However, a larger degree of noises will be required within the network for the artificial life form to overcome complex obstacles, which consequently reduce the efficiency of the artificial life form. STDP is implemented in this situation to increase the performance of the artificial life form under such circumstances. There will be tests conducted in order to observe and investigate the effect of STDP on the behavior of artificial life form under different conditions. The basic principle of STDP will be discussed in the third section of this chapter, and a more detailed explanation as well as test results will be presented in Chapter 3 and 4.

To solve the second problem mentioned in the last section, the network randomly selects one of the neurons if N1 and N2 fire the same maximum number of spikes at a particular time window. That also adds some degree of random to the artificial life form, which acts as another source of noise in this model. In the next section, the thesis will examine a more detailed level of the

neural controller—the spiking neural network models.

2.4 Izhikevich Model

2.4.1 Which model to use?

There is a great shift in the emphasis on the artificial neural network community towards spiking neural networks during last few years[5]. The artificial life form in this project is also controlled by a spiking neural network. However, there is a large number of spiking neural network models that can be simulated using software such as Matlab and C++. So, which model to use and implement will be a very important issue to consider before programming.

The first and most important characteristic is the computational or implementation efficiency of the model. The first section has already mentioned the importance of the computational efficiency (Please refer to Figure 5) because the simulation time window does take a lot of execution time. Therefore it is really important that the model possess a high simulation speed so that the artificial life form can run with sufficient speed in real time even it only consists of eight neurons. The integrate-and-fire model and the Izhikevich Model are the most suitable spiking network models in this case. They both contain a relatively small number of floating operations which are needed to be completed during each simulation time step.

The other important feature is the biological plausibility of the model. This project uses Izhikevich Model because it is also capable of producing rich firing patterns exhibited by real biological neurons such as regular and tonic spiking, bursting or even resonance. This feature of the Izhikevich model gives the user a high degree of control on each single neuron and allows more research space in future while the very simple integrate-and fire model would be more limited.

2.4.2 Izhikevich model

There is a geometrical analysis of neuron's dynamical system in the book “Dynamical Systems in Neuroscience” written by Eugene M.Izhikevich. In the book, he uses a geometrical method—the *phase portraits* to describe four bifurcation mechanisms. These four bifurcations represent all the transitions from a resting neuron's stable equilibrium mode to an excitatory neuron's unstable periodic spiking mode [7]. This thesis will not go into the geometrical analysis of the dynamics in detail since the emphasis is on the algorithms and learning rules of the neural networks. Nevertheless, a relatively brief description of the dynamical system of the Izhikevich model will be provided in this section to show the basic abilities of producing various biological spiking patterns of real neurons using the Izhikevich model.

The neuron should not be seen just in terms of ions and channels, or in terms of an input/output relationship, but also as a dynamical system. A dynamical system consists of a set of variables that describe its state and a law that describes the evolution of the state variables with time [7]. The Izhikevich model which contains two ordinary differential equations is a reduced form of accurate but less-efficient Hodgkin-Huxley-model.

The two equations are shown below [8].

$$v' = 0.04v^2 + 5v + 140 - u + I \quad \text{Eq1}$$

$$u' = a(bv - u) \quad \text{Eq2}$$

With the auxiliary after spike resetting

$$\text{If } v \geq 30\text{mV} \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \quad \text{Eq3}$$

In Eq1 and Eq2, 'v' and 'u' are dimensionless variables and a,b,c,d are dimensionless parameters. The variable 'v' represents the membrane potential of the neuron and 'u' represents a membrane recovery variable, which accounts for the activation of K⁺ ionic currents and inactivation of Na⁺ ionic currents [9]. Membrane recovery variable 'u' provides a feedback to the membrane potential. The relationships between all the variables and parameters in Eq1 and Eq2 are shown in Figure 6.

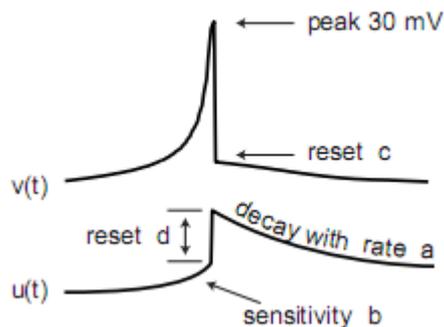


Figure 6 relationships between variables and parameters of the Izhikevich model equations taken from reference [4]

The parameter 'a' describes the time scale of the recovery variable 'u'. The parameter 'b' describes the sensitivity of the recovery variable 'u' to the sub threshold fluctuations of the membrane potential 'v'. The parameter 'c' describes the after-spike reset value of the membrane potential 'v'. The parameter 'd' describes the after-spike reset of the recovery variable 'u'. Like most real neurons, the model does not have a fixed threshold. Depending on the history of the membrane potential prior to the spike, the threshold potential can be as low as -55mV or as high as -40mV.

A typical setting for variables and parameters are a=0.02, b=0.2, c=-65, d=8 [8]. This setting of the Izhikevich model equations can produce the regular spiking pattern which is the most common spiking behavior in an excitatory neuron in all biological systems [8]. This project is going to apply this setting on the neurons in the spiking neural network controller. However, there are also many other settings of the variables and parameters that can allow the model to produce a lot of different spiking patterns (The other types of spiking neurons are not in consideration in this thesis due to the time limit and will be added in future works). A detailed diagram showing the relationships between these settings and different spiking patterns is shown next page.

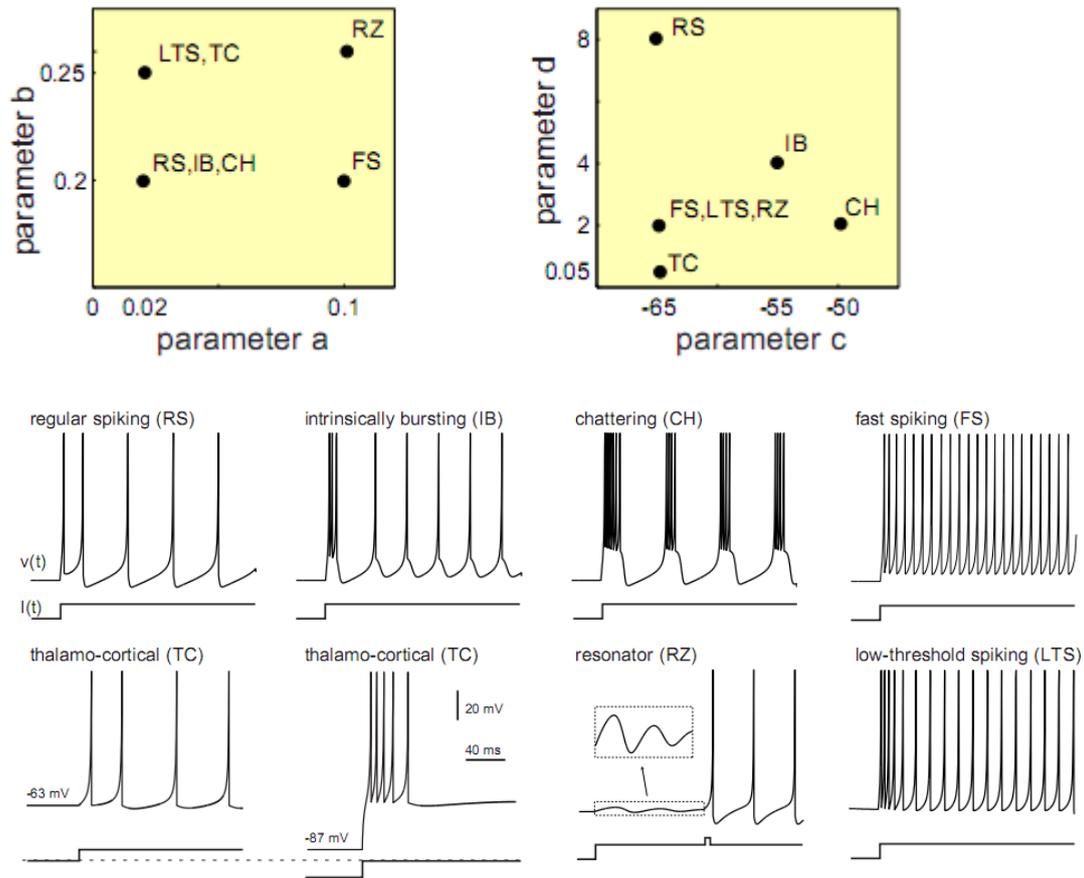


Figure 7 Relationships between parameters and variables of the Izhikevich model equations and different spiking patterns taken from reference [8]

Due to the size of the thesis, this section will plot only a few of these spiking patterns (Regular Spiking, Fast Spiking, Tonic Spiking and Resonator) using Matlab through four experiments. Noise will also be injected instead of direct DC input at experiment 2 in order to demonstrate some more realistic behaviors of the spiking pattern within the neurons.

There is a matlab implementation of a simple model of spiking neurons provided by Izhikevich in his journal "Simple model of Spiking Neurons" which illustrates the simulations of a network of randomly connected 1000 neurons in real time [8]. The project has modified the original version of the Matlab code in order to produce and demonstrate the various firing patterns in single neurons. A detailed Matlab code of the resonator spiking pattern will be attached in the Appendix and all the other related M files will be saved in the folder 'Izhikevich Model' in CD.

2.4.3 Regular Spiking and Fast Spiking

Experiment 1:

Parameter 'a' has been altered to see its effect on the spiking patterns of the neuron. Parameters b, c, d are fixed at 0.2, 65, 8 respectively. A dc step input is applied to the neuron in this model. Neuron A corresponds to the inhibitory fast spiking (FS) neurons in the cortex. Neuron B corresponds to the excitatory regular spiking (RS) neurons in the cortex.

Experiment1	Parameters				
Neurons	A	b	C	d	Input/magnitude
A	0.1	0.2	65	8	dc step input , 5
B	0.02	0.2	65	8	dc step input , 5

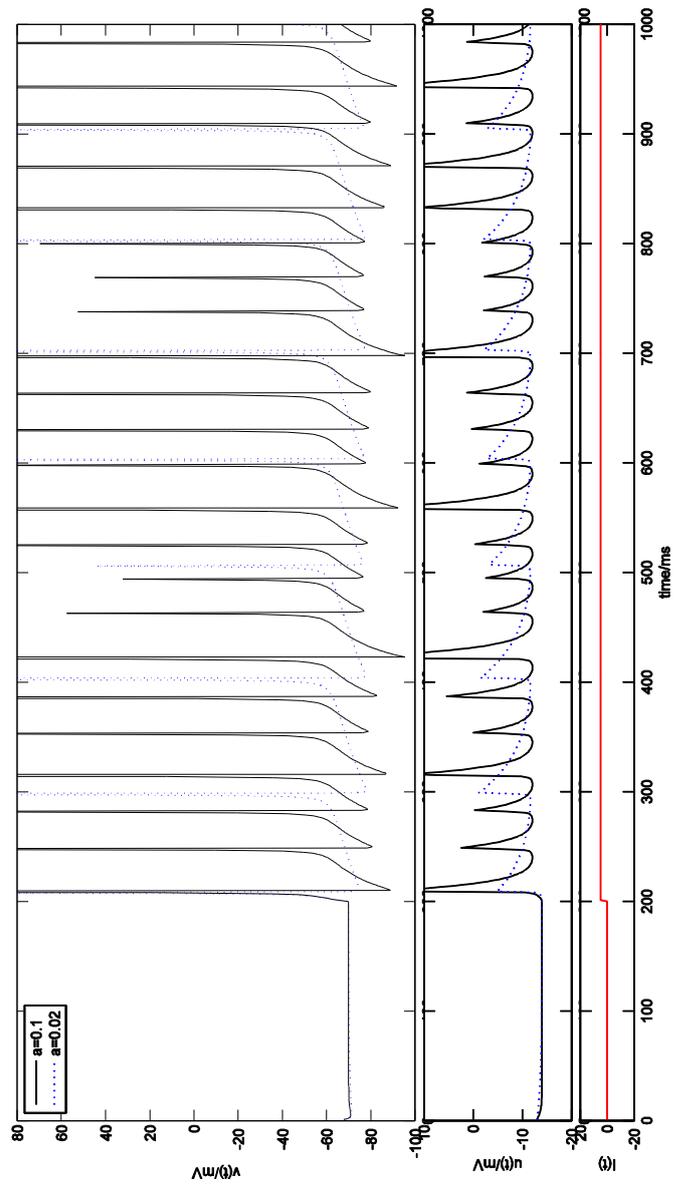


Figure 8 Spiking Pattern with different value of 'a'

The parameter 'a' describes the time scale of the recovery variable u [8]. A larger value of 'a' will speed up the recovery rate of the recovery variable 'u', thus speed up the recovery rate of the membrane potential v . In Figure 8 above, the dotted line shows the trace of a regular spiking neuron (RS) while the solid line indicates the trace of fast spiking (FS) neuron with a larger value of 'a'.

Larger values of 'a' (which is 0.1 in this case) make 'u' build up very quickly above zero at the beginning according to the first term $-au$ in Eq2. The second term 'abv' in Eq2 makes 'u' build up further to a very positive value before it resets. This positive value of 'u' makes the membrane potential 'v' fall to a very low level after the reset. At this moment, both $-au$ and abv terms in Eq2 will speed up the recovery rate of 'u' since 'u' is very positive and 'v' is very negative now. Faster recovery rate of 'u' means faster recovery rate of 'v', therefore if a larger value of 'a' is chosen for the model, more spikes will be generated.

Experiment 2:

Parameter 'b' has been altered to see its effect on the spiking patterns of the neuron.

Parameters a, c, d are fixed at 0.02,65,8 respectively. A random thalamic input with a mean value of 0 and a fluctuation of 5 is applied to the neuron in this model. Noise is also injected as a random thalamic input to show a more realistic view of the spiking pattern in neurons.

Experiment2	Parameters				
Neurons	a	b	c	d	Input/mean/fluctuations
A	0.1	0.2	65	8	random thalamic input,0,5
B	0.1	0.25	65	8	random thalamic input,0,5

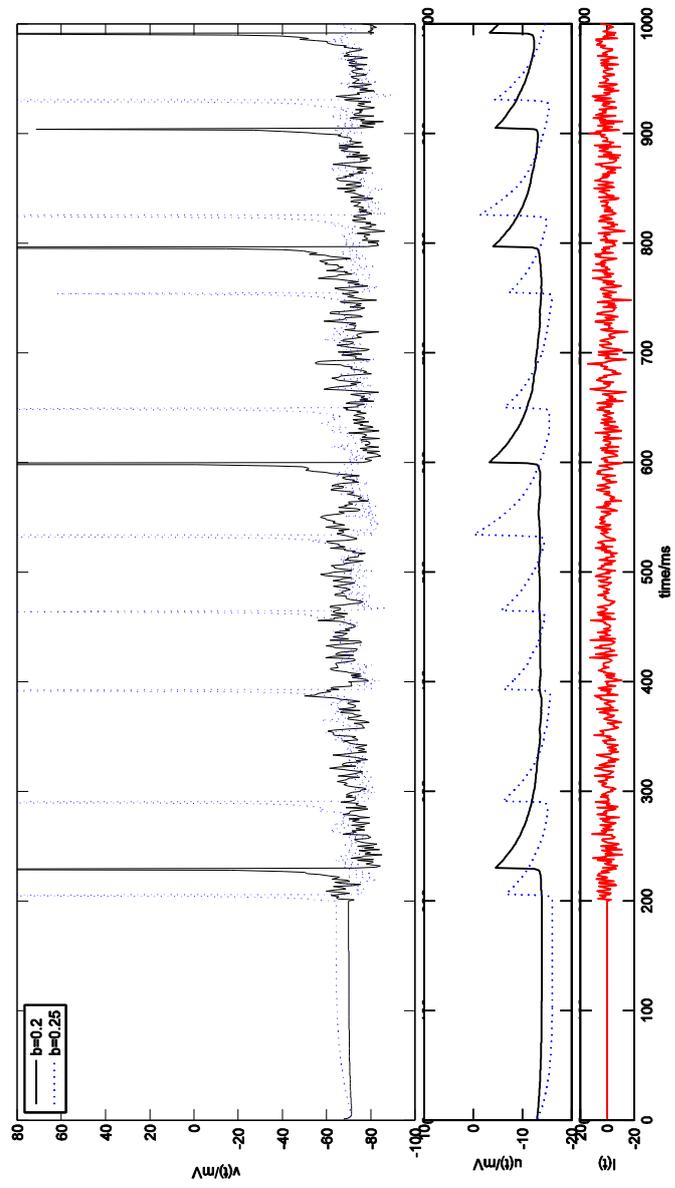


Figure 9 Noise injected Spiking Pattern with a different value of 'b'.

The parameter 'b' describes the sensitivity of the recovery variable u to the sub-threshold fluctuations of the membrane potential v [8]. In Figure 9 above, the solid line shows the trace of a regular spiking neuron (RS) while the dotted line indicates the trace of a fast spiking (FS) neuron with a larger value of 'b'.

Larger values of 'b' result low-threshold spiking dynamics. In Figure 9 above, you can observe a small peak value between the first and second spike of the dotted trace (FS). That peak value doesn't cause a spike, which means that the relative membrane potential sensitivity of the fast spiking neuron (FS) towards a success spike is relatively low compared to the regular spiking neuron (RS). For the fast spiking neuron (the dotted trace), the value of ' v ' is negative below the threshold, thus a greater value of 'b' slows the rising rate of u down at the sub-threshold level according to the 2nd term ' abv ' in Eq3. Slower rising rate of ' u ' means faster rising rate of ' v ', so more spikes will be generated if a larger value of b is chosen for the model.

2.4.4 Chattering

Experiment 3

Parameters 'c' and 'd' describe the after-spike reset value of the membrane potential ' v ' and recovery variable ' u ' respectively. They have been altered to produce the tonic spiking which is also called Chattering in this experiment. Chattering (CH) neurons can fire stereotypical bursts of closely spaced spikes [8]. It is believed that such neurons contribute to the gamma-frequency oscillations in the brain. It has a relatively large value of 'c' (c=-45) and a small value of 'd' (d=2).

Experiment3	Parameters				
Neurons	a	b	c	d	Input/magnitude
A	0.02	0.2	45	2	dc step input , 5

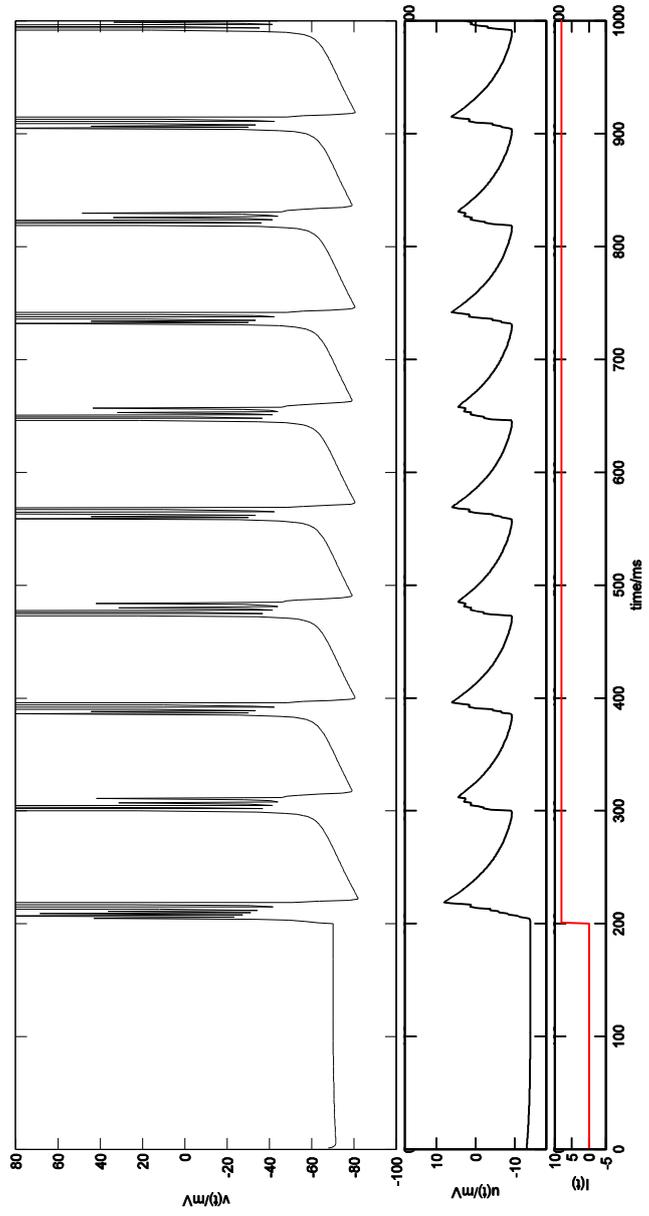


Figure 10 Spiking Pattern showing the Chattering (CH)

In Figure 10, a higher level of membrane potential 'v' also gives a faster rising rate of 'v'. Therefore if the after-spike reset value of v is high (i.e. v=-45), the value of u should also be greater in order to bring 'v' down to the sub-threshold value according to Eq1 of the Izhikevich model. However, a smaller value of d makes u build up slowly. That allows 'v' generating more spikes with greater frequencies during a limited amount of time (bursting) until the value of u is large enough to bring 'v' back to the sub-threshold value. As a consequence, each time a spike is triggered, the neuron will fire bursts of spikes and stop within a limited amount of time when the recovery variable 'u' becomes larger. That gives the Tonic Bursting behavior of the chattering neurons in cortex.

2.4.5 Resonator

Experiment 4

Resonator (RZ) neurons have damped or sustained sub-threshold oscillations. They resonate to rhythmic inputs having appropriate frequency [8]. RZ neurons have both relatively large parameters of 'a' and 'b'. Implementation of the model for resonator neuron using Matlab is shown in Appendix1.

Experiment4	Parameters				
Neurons	a	B	c	d	Input/magnitude
RZ	0.1	0.26	65	8	dc step input , ?~?
A stimuli applied at time 500ms with magnitude 0.4mV and duration 10ms					

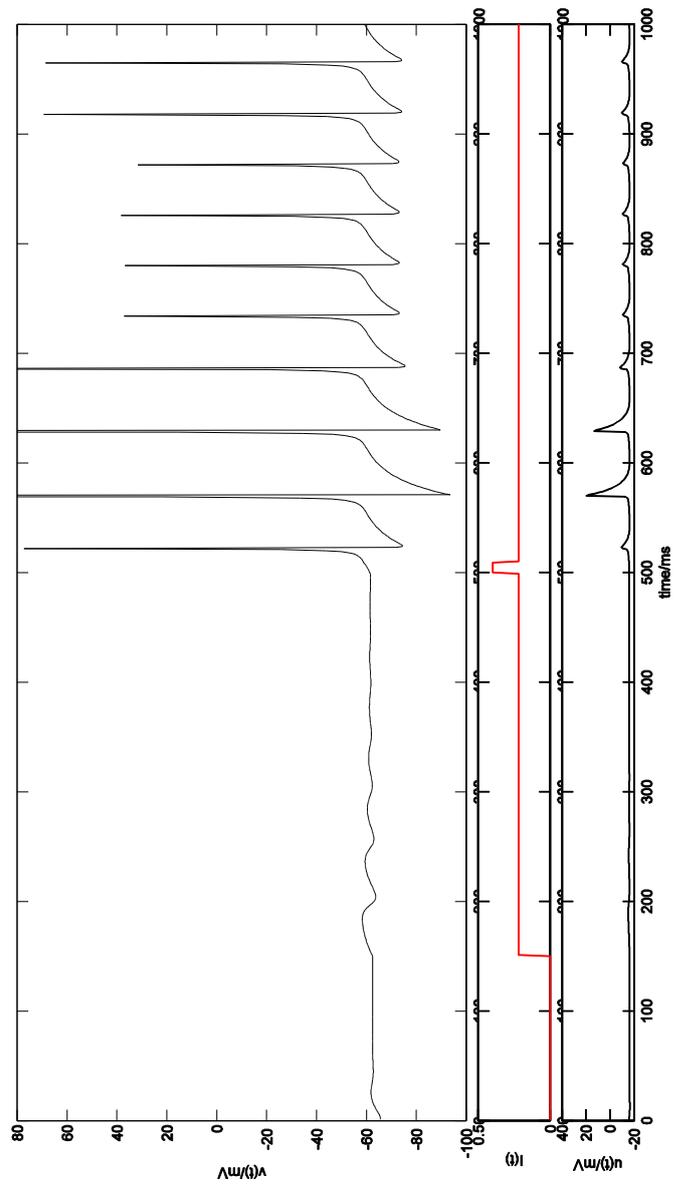


Figure 11 Spiking Pattern showing the Resonator (RZ)

After a few experiments, the RZ neuron is found resonating only when a dc step input with magnitude about 0.16~0.22mV is applied to the model. That corresponds to the behavior of RZ neuron which only resonates to rhythmic inputs having appropriate frequency. RZ neuron also shows a bi-stability of resting and repetitive spiking states. The neuron can be switched between the states by an appropriately timed brief stimulus [9] as shown in the experiment table above.

Relatively large value of 'a' would speed up the rising rate of u under the threshold (or when $u < 0$) according to the first term $-au$ in Eq3. However, if 'a' and 'b' are both relatively large, the second term abv in Eq2 which can slow down the rising rate under the threshold (or when $v < 0$) will balance the rate of 'u'. As a result, the value of u could resonate at particular input frequency and converge to a steady state if an appropriate input has been applied.

2.5 STDP (Spike Timing Dependent Synaptic Plasticity)

The artificial life form in this project is designed to possess some 'intelligence' so that it can 'learn' to avoid obstacles and catch the target—the food. This also means that the artificial life form can 'evolve' by itself. The 'intelligence' of the artificial life form is contained in the connections and weightings between neurons within the spiking neural network controller. The network controller can make the artificial life form 'intelligent' by controlling how these weights change, and the ways of controlling the weight are called learning rules. STDP is the learning rule used to control the weight of the spiking neural network controller in this project.

The robot under the control of STDP produces a rich variety of behaviors and desirable properties such as sensorimotor and synaptic robustness [14]. It also exhibits great computational abilities. What is interesting about using STDP in spiking neural networks compared to traditional rate-based plasticity is that it shows robust to jitter and spike train randomization, suggesting counter intuitively that, under certain conditions, spike-timing dependent synaptic (STDP) rules can work very well even when spike timing is disrupted [14]. The important question is in what situations, if any, analogous robustness may be found in natural systems. To answer this, behavioral studies are needed [14]. In Di Paolo's paper "Spike-Timing Dependent Plasticity for Evolved Robots", he demonstrates the usefulness of robotic studies for investigating the properties of STDP [14].

2.5.1 What is STDP?

The idea that information in the nervous system is coded by spike-rate has been widely accepted and used in the application of spike-based neural controllers traditionally. However, recent experimental studies indicate that the changes in synaptic efficacy (the weighting) could be highly dependent on the relative timing of the pre- and postsynaptic spikes, which means that the timing of the spikes could be responsible for carrying information. As Donald Hebb was the first to suggest a precise rule that might govern the synaptic changes, such rules have been named Spike-Timing Dependent Synaptic Plasticity (STDP). In Hebb's rules, "when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency as one of the cells firing B is increased"[11]. One feature of this rule is that it correlates the activity of pre-and postsynaptic neurons causally. Another feature of this rule is that it only describes the condition under which synaptic efficacy increases, but does not describe the condition under which it decreases. Recent experimental studies in vivo have found that depression also occurs in some of the synapses when presynaptic spikes occurs after the postsynaptic synapse. This result suggests that Hebb's original rule governing potentiation must be supplemented with an opposite timing dependent process of depression[12].

Therefore a temporally asymmetric STDP rule has been chosen to be implemented into the spiking neural network controller. In this case, if a presynaptic spike precedes the postsynaptic spike, the synapse is potentiated, whereas the opposite relation leads to the depression of the synapse. Figure 12 below describes how the weights of the synapse change according to the timing of the spikes in pre and postsynaptic neurons.

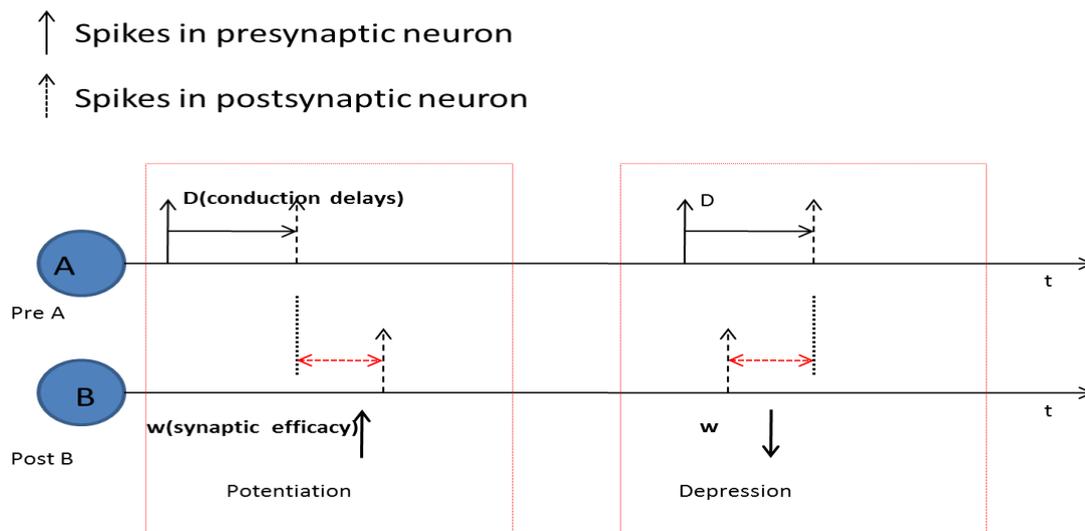


Figure 12 Relationships of the timing of the spikes between pre and postsynaptic neurons

In Figure 12, presynaptic neuron A fires and its spike arrives in postsynaptic neuron B after D (20ms in this project) milliseconds. If the spike from A arrives in B before neuron B fires, which means the pre spike precedes the post spike, the weight between A and B will increase. That is called the potentiation. If the spike from A arrives in B after neuron B fires, which means the post spike precedes the pre spike, the weight between A and B will decrease. That is called the depression. How much the weight is increased or decreased during potentiation and depression depends on how the value of STDP varies with the time interval between pre and postsynaptic spikes. The detailed time window which includes the modification function of the STDP showing this relationship is illustrated in Figure 13 below.

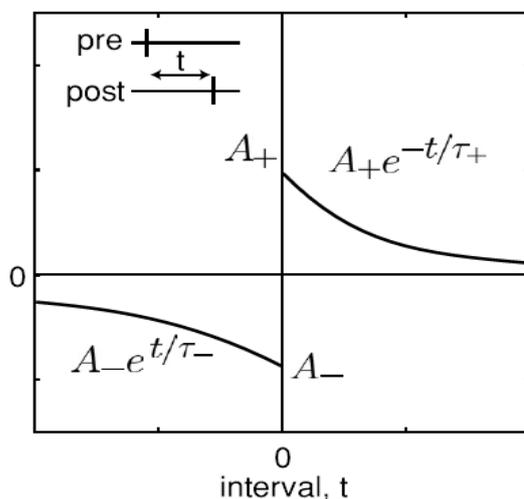


Figure 13 Time window of STDP showing its relationships with the time interval between pre and post synaptic spikes

Taken from reference [10]

In Figure 13, the weight of synaptic connection from pre- to postsynaptic neuron is increased if the post neuron fires after the presynaptic spike, that is, the interspike interval ' t ' > 0 . The magnitude of change decreases as A_+e^{-t/τ_+} . Reverse the order results in a decrease of the

synaptic weight with magnitude A_-e^{-t/τ_-} [10].

Parameters used [10]:

$$\tau_+ = \tau_- = 20ms, A_+ = 0.1 \text{ and } A_- = 0.12$$

According to Figure 13 above, each neuron's STDP will decay with a particular rate with time constant $\tau = 20ms$ and being reset to 0.1 once the neuron fires.

One of the key concerns of using this rule for synaptic plasticity is that the potentiation might go out of control, which will cause stability problems in the spiking network. Direction damping is one of the ways that can solve this problem; it has been added as an additional element to regulate the activity of STDP in order to make sure of the stability of the network. A detailed explanation and implementation including the equations about directional damping will be given at the end of Chapter3.

2.5.2 Investigation of the behavior of the artificial life form

The artificial life form will then show various interesting behaviors under the control of an STDP implemented spiking network. The project will alter various parameters in the network and try to increase the efficiency of the artificial life form (the speed of catching the food) and discover the potential of using STDP to develop the intelligence of the artificial life form. Investigation of the STDP will also be conducted through various experiments and tests under different conditions. From the results of these tests and experiments, the thesis will try to discover the relationships between the behavior of the artificial life form and its synaptic connections and the effect of STDP on them.

A spiking neural network that consists of 1000 randomly connected neurons simulated using Matlab is provided by Eugene M. Izhikevich in his paper "Computation with spikes" [10]. This project simplifies and modifies the original version of this Matlab code to fit its own use.

Chapter 2 has given the basic background information from various aspects. The relevant biological background, the underlying bio-inspired learning rules and algorithms, the uses of Izhikevich model and the basic theory of STDP are all introduced in this Chapter. Now a detailed explanation of this spiking neural network model and its implementation process will be unfolded in Chapter 3.

3. Method

In this chapter, Section 3.1 will first give a detailed description of the simulation environment used in this project. Then Section 3.2 will illustrate the initial design of the artificial life form. Section 3.3 and 3.4 will demonstrate implementation of the spiking network controller and the implementation of the external environment separately. Finally Section 3.5 will conclude the chapter by the showing the implementation of STDP.

3.1 Choice of simulation environment

3.1.1 Source of the code

The Matlab program code used in this project is based on a web source named 'Conway's game of life' provided by Moshe Lindner with a license file. The web source is downloaded from Matlab-Central <http://www.mathworks.com/matlabcentral/fileexchange/26805-conways-game-of-life>[15]. All the related M file in the original document is stored in the folder 'Conway's game of life' in CD. The original program implements a simple cellular automation which consists of no neural network. This project modifies and uses part of the program code of 'Conway's game of life' as its own environmental settings in order to save time for the research on neural networks. The modified version of Izhikevich model is then implemented into the environment as the artificial life form's central network controller. The Neural Network part is completely a novel part compared to the original program. The final version of the modified Matlab code named 'Artificial life form' is listed in Appendix2 in Chapter 7, and the related M file is stored in folder 'Final version of Artificial life form' in CD.

3.1.2 Matlab GUI (graphical user interface)

The original 'Conway's game of life' is represented as a Matlab GUI. A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks [16]. Matlab also provides a very powerful tool named Guide (graphical user interface development environment) which simplifies the process of programming and laying out GUIs. When the original GUI program is modified in this project, using the Guide Layout editor provides great advantages and saves a lot of programming time. The GUIDE Layout Editor allows the users to configure various GUI components such as buttons, text fields, sliders, and axes in the layout window. As soon as the configuration is completed and saved, an updated M file will be generated automatically and the user could use the M file editor to further develop the GUI by modifying the contents of the callbacks within the M file. A callback provides the routines that execute in response to user-generated events such as a mouse click. Each component in the layout window has its own corresponding callbacks in the M file editor. All the functions in the program can be configured in detail by editing the contents of callback.

Most parameters and variables of the spiking neural network and environment are stored in the handles structure at the beginning of the M file. The handles structure is another powerful function provided by Matlab which allows the users to save and load huge amounts of data at any time they want. When the program runs, the data in the handles structure is passed as an argument to all callbacks. This powerful data storing structure greatly saves time for testing since the experiments in this project are required to load plenty of data from time to time during simulations.

All the commands, handles, functions and callbacks mentioned in the later sections of the thesis that can be found in the Matlab program code in Appendix2 will be marked in blue color for the reader's convenience of understanding the program.

A final designed version of the layout window is shown in Figure 14 below.

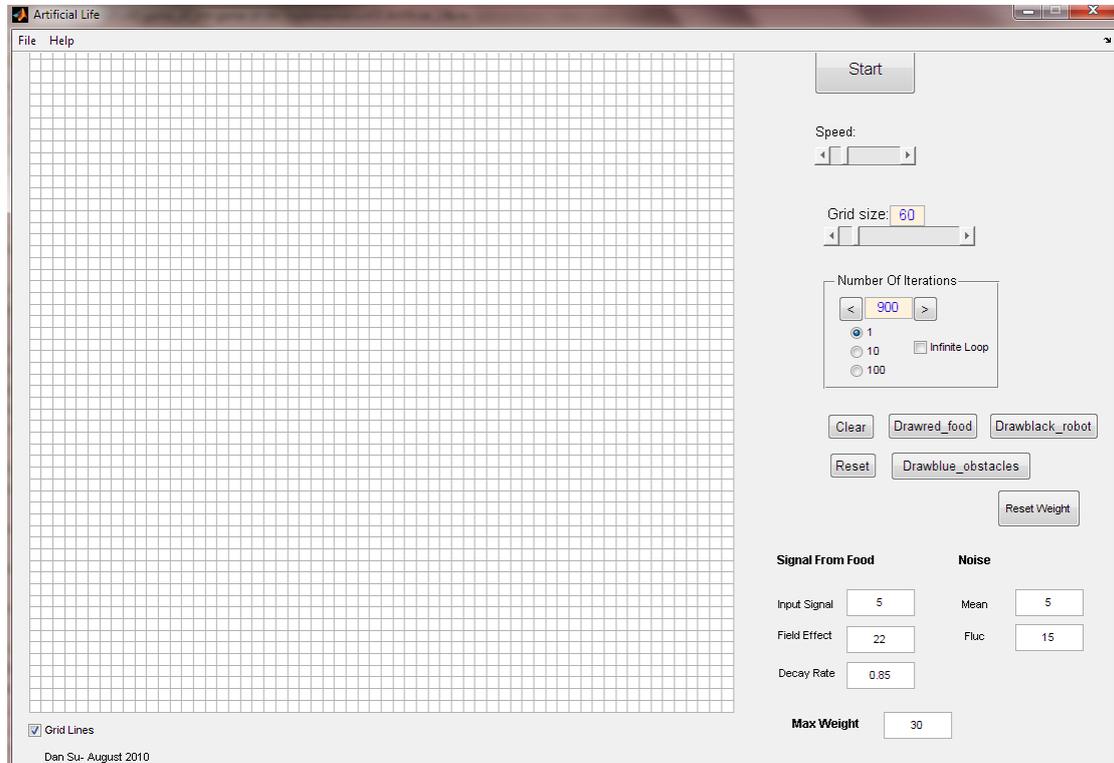


Figure 14 Final designed version of the layout window

Figure 14 shows the final version of the layout window. The menu tool bar placed on top of the layout window has been configured through the menu editor in this project. This menu provides a lot of functions such as saving images and videos which can show the whole running process of the artificial life form. The users could also find a user manual that briefly describes the functions of all the components in the layout window. A detailed explanation of functions of these components such as buttons, sliders, and boxes will be given in the later sections of this chapter.

3.1.3 Program structure

Figure 15 below shows the basic flow diagram of the whole program structure:

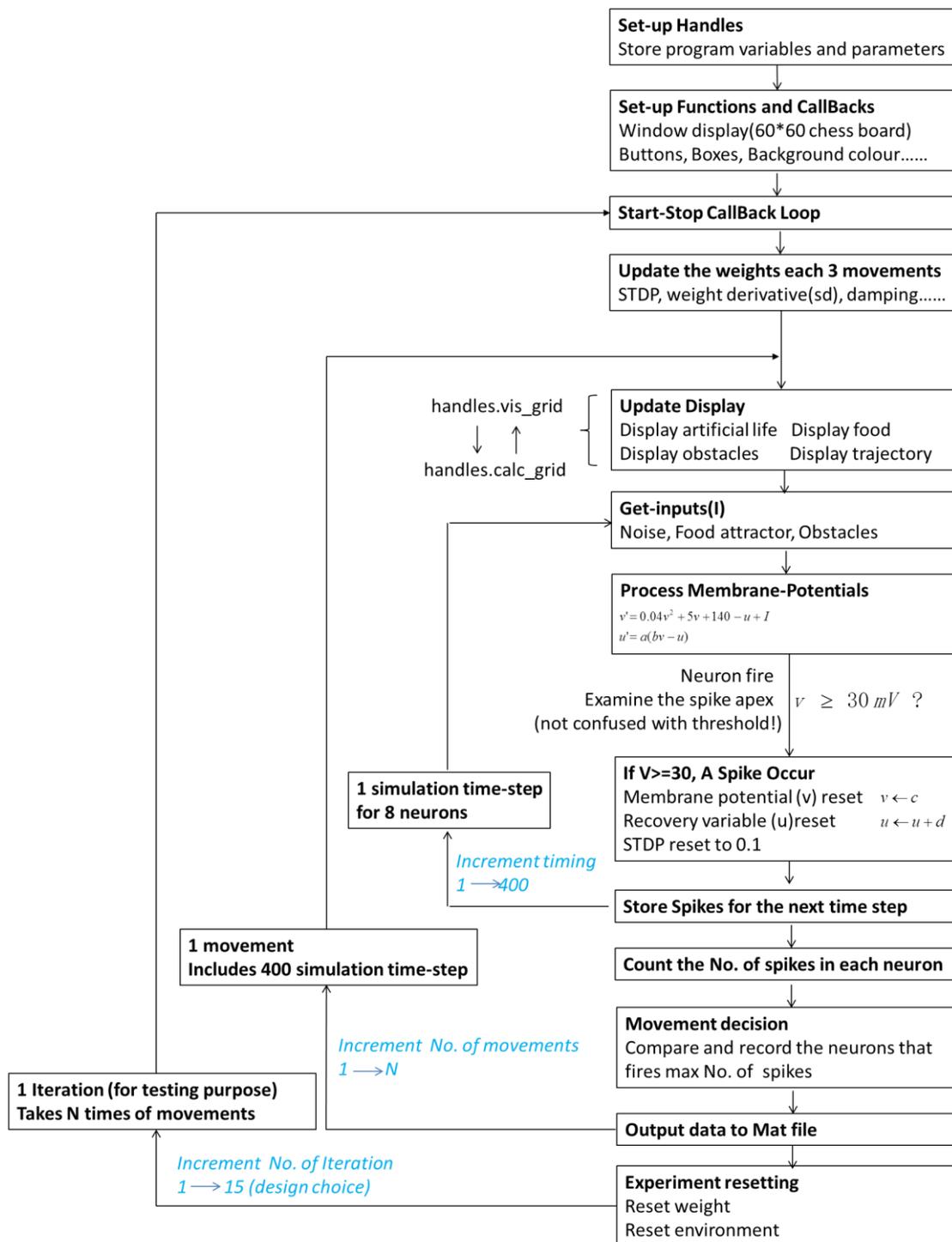


Figure 15 Whole program structure

Figure 15 above shows the whole program structure. The previous sections of this chapter have already introduced the handles structures, functions and callbacks including window display which will be initialized and set up at the start of the program. There are three main loops in the program. The inner most loop represents the computational process that happens inside the central spiking network controller. Its Matlab implementation will be explained in Section 3.3. The second loop is the simulation time window required for one movement of the artificial life form.

Its computational process is closely related to the outside environment, which will be described in detail in Section 3.4. In Section 3.5, the thesis will discuss the implementation of STDP which plays the most important role throughout the whole program including the third loop. The third loop represents the repeat runs of the artificial life form during each experiment.

3.2 Initial Design

3.2.1 Displaying the artificial life form on the screen

In Figure 14, the grids (cross lines) of the chessboard-like axes displayed in the layout window are created in the 'function `grid_size_N_Callback(hObject, eventdata, handles)`'. This callback allows users to change the size of the grid using the grid size-slider located at the right part of the layout window. The default size and color of the chessboard are defined as '`handles.vis_grid=zeros(60);`' in the handles structure at the start of program. The value of the grid is zero because the default background color-white has been assigned to zero. The variable "`handles.vis_grid`" is also used to define the position coordinates of each grid in the axes in this program.

Before designing the dynamics of the artificial life form, the project should first 'draw' an artificial life form on the screen, and display it in front of the users. All the 'drawing' command has been defined in the function '`function [a]`' located at the end of the program. The value of 'a' for the drawing function of artificial life form has been assigned to the numerical value '1'. Value '1' is the *image value* assigned to the color of the artificial life form; in this program it represents the display of color black in the layout window. Inverse and non-inverse, white and black, 0 and 1, only 2 image value of colors are required since there is only one type of object in the screen at current stage. As the development goes further, more colors will be needed and a function defining the color-map settings of the whole program will be described in Section 3.4. The position of the artificial life form is defined as '`handles.vis_grid(30,30)=artificial_life;`' in the handles structure. (0, 0) represents the coordinates of the origin which is located at the top left of the axes. (30, 30) is the initial coordinates of the artificial life form, which in this case is located in the center of the chessboard since the chessboard has the maximum horizontal and vertical size of 60 respectively. When the setting is completed, the layout window should look like this:

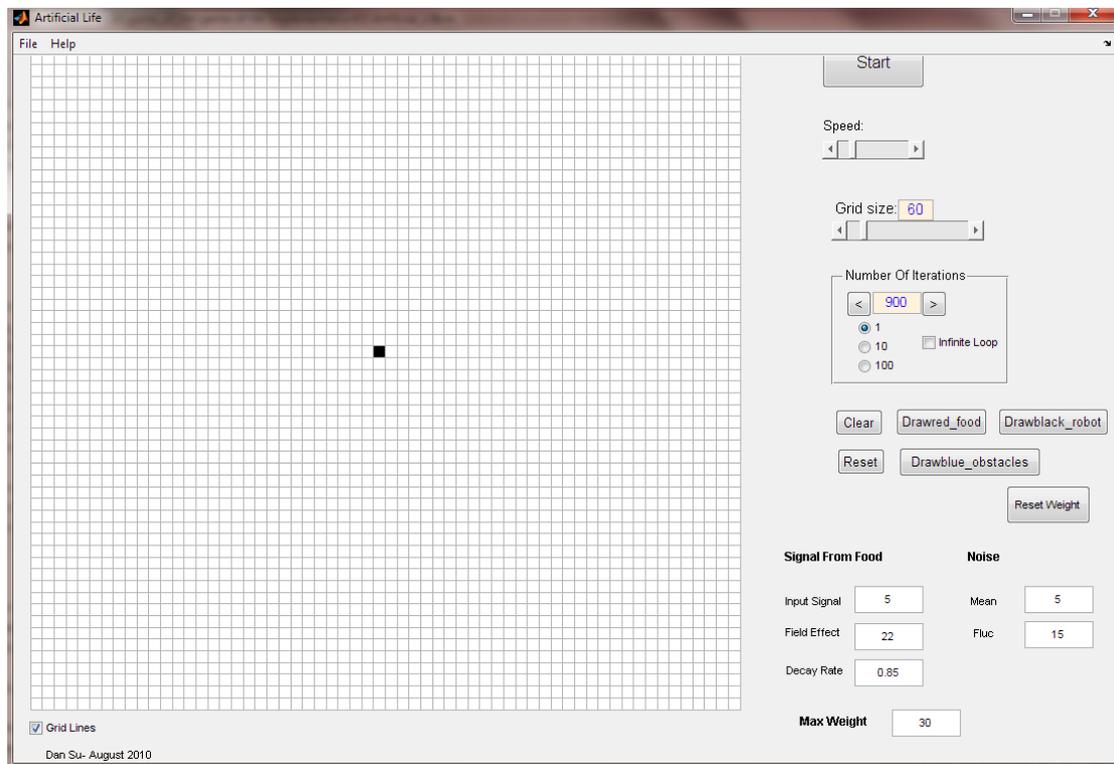


Figure 16 Display of the artificial life form

Figure 16 shows the artificial life form displayed in the center of the chessboard. Its value is '1', so that its color appears to be the inverse color of the background which is black.

3.2.2 Making the artificial life form move

In order to make the artificial life form move, the program should update its position at each movement inside the start-stop loop callbacks. As mentioned in the previous section, the position information of the artificial life form is stored inside the handles 'handles.vis_grid'. Assume that the horizontal and vertical coordinates of the artificial life form are RI and CI respectively, the relationships between the 8 moving directions and the coordinates of the artificial life form are shown in Figure 17 below.

RI-2,CI-2	RI-2,CI-1	RI-2,CI	RI-2,CI+1	RI-2,CI+2
	↖ N8	↑ N1	N2 ↗	
RI-1,CI-2	RI-1,CI-1	RI-1,CI	RI-1,CI+1	RI-1,CI+2
RI,CI-2	← N7	RI,CI	N3 →	RI,CI+2
RI+1,CI-2	RI+1,CI-1	RI+1,CI	RI+1,CI+1	RI+1,CI+2
	↙ N6	N5 ↓	N4 ↘	
RI+2,CI-2	RI+2,CI-1	RI+2,CI	RI+2,CI+1	RI+2,CI+2

Figure 17 Relationships between the moving direction and the coordinates of the artificial life form

In Figure 17, the 8 moving directions have been associated with 8 coordinates. For instance, if the artificial life form wants to move up (N1 fires the max No. of spikes), the computation for this movement will be given as the equation:

$$handles.vis_grid(RI - 1, CI) \leftarrow handles.vis_grid(RI, CI) \quad eq4$$

$$handles.vis_grid(RI, CI) \leftarrow 0 \quad eq5$$

Handles.vis_grid (RI, CI) represents the *image value* of the color of the grid at position RI, CI inside the axes. The current *image value* for the grid located at [RI, CI] is '1' which represents black. This *image value* '1' will be assigned to the *image value* of the upper grid located at [RI-1, CI] during this movement. The second equation clears the grid located at the original position of the artificial life form back to the background color-white.

Now, the artificial life form still needs a 'command' that can tell it to choose which direction to go. Recall from Chapter 2 the 8 neurons N1~N8 represent the 8 directions of the movement. To associate the 8 neurons (N1~N8) with these 8 directions (RI+?,CI+?), the program uses a 'switch' statement. The neuron that fires maximum number of spikes during a fixed time window will be the direction that the artificial life form is going to move towards, with its neuron number matching its direction 'case' under 'switch' statement. There is only one 'switch' statement inside the program code, therefore it is easy for the reader to find it in Appendix2. It is implemented inside 'function [handles]=calc_game(handles)' which is the function that is responsible for the computational process carried out by the spiking neural network controller.

How does the neuron fire? How does the spiking network controller deals with spikes? A detailed explanation of the implementation of the spiking neural network will be provided in the next section.

3.3 Spiking Neural Network Implementation

The spiking network model used in this project is the Izhikevich model. The Izhikevich model controls all the dynamics of the 8 direction neurons such as their resting and firing status. The computational information of the movement of the artificial life form is stored in the handles structure 'handles.calc_grid' and the function 'function [handles]=calc_game(handles)'. Therefore all the code for the Izhikevich model will be implemented inside the function 'function [handles]=calc_game(handles)' since the spiking network is the only part that is responsible for the computational process of the movement. When the computational process is completed, the start-stop loop will update the position information 'handles.vis_grid' with the computational result inside 'handles.cal_game' and refresh the display of the layout window with a new version of the 'handles.vis_grid', and the artificial life form displayed on the layout window will move one grid square.

3.3.1 Connections between the neurons

All the neurons inside the spiking network are fully interconnected, apart from self-connections. A figure that describes how these neurons are connected with each other is shown below.

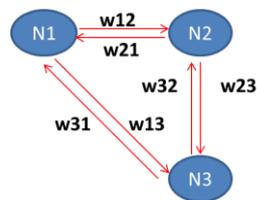


Figure 18 Interconnections of the neurons

Due to the limit of space and complexity of the diagram (there are 56 connections in total), Figure 18 only shows the interconnections between 3 neurons N1~N3. Although the connections related to the other neurons have been eliminated from the figure, the way they are connected with each other is the same as these 3 neurons.

The synaptic connections between these 8 neurons are represented as weight value 'w_{ij}' where 'i' indicates the pre-synaptic neuron number, 'j' indicates the post-synaptic neuron number, and 'w_{ij}' is the value of weight for the connection from neuron i to neuron j. A table that lists all the synaptic connections between each of the 8 neurons is shown in Figure 19 below.

Neuron No.	1	2	3	4	5	6	7	8
1	×	W ₂₁	W ₃₁	W ₄₁	W ₅₁	W ₆₁	W ₇₁	W ₈₁
2	W ₁₂	×	W ₃₂	W ₄₂	W ₅₂	W ₆₂	W ₇₂	W ₈₂
3	W ₁₃	W ₂₃	×	W ₄₃	W ₅₃	W ₆₃	W ₇₃	W ₈₃
4	W ₁₄	W ₂₄	W ₃₄	×	W ₅₄	W ₆₄	W ₇₄	W ₈₄
5	W ₁₅	W ₂₅	W ₃₅	W ₄₅	×	W ₆₅	W ₇₅	W ₈₅
6	W ₁₆	W ₂₆	W ₃₆	W ₄₆	W ₅₆	×	W ₇₆	W ₈₆
7	W ₁₇	W ₂₇	W ₃₇	W ₄₇	W ₅₇	W ₆₇	×	W ₈₇
8	W ₁₈	W ₂₈	W ₃₈	W ₄₈	W ₅₈	W ₆₈	W ₇₈	×

Figure 19.1 Table showing the interconnections between 8 neurons in the spiking network

In Figure 19.1, the first number of the weight W 's subscript represents the pre-synaptic neuron number, which is also the column number in the table. The second number of the weight W 's subscript represents the post-synaptic neuron number, which is also the row number in the table. There are no self-connections, therefore all the self-connections are marked with '×'.

The Izhikevich model belongs to the class of pulse-coupled neural networks (PCNN)[8]: The synaptic connection weights ' w_{ij} ' is stored in the weight matrix ' s ' and defined in the handles structure '`handles.calc_weight`'. The firing of neuron i will act as an additional input with value ' w_{ij} ' which instantaneously change the membrane potential ' v ' of neuron j . By doing that, the presynaptic neuron i could influence the activity of neuron j through the weight value between their connections. The updating equation for input ' I ' in neuron j at each simulation time step is given by:

$$I(i) \leftarrow I(i) + W_{i,j} \quad \text{eq6}$$

For example, when neuron 1 fires, the value of ' w_{12} ' will act as an additional input(I) which will increase or decrease the membrane potential ' v ' of neuron 2 after a certain amount of time delay. The time delay between neurons in this project has been set to 20 time steps which is inherited from the original Izhikevich model settings and is also proved to be working well during various tests. The same settings will apply to the other neurons that connect to neuron 1.

The initial weight value ' w_{ij} ' for all the connections between neurons have been fixed at '+1' currently. The maximum and minimum weight values have been set to '30' and '-30' later in this project respectively. Therefore value '+1' represents a medium synaptic weight in this case. The reason of fixing all the weightings to a small positive value is to temporally establish small positive correlations between pre and post synaptic spikes. So that the connections between neurons could be established before STDP is implemented to the network. The synaptic weight values used in this project have been chosen carefully. During the testing, they were demonstrated to be appropriate when cooperating with other parameter settings such as the signal strength of food and noise. Other sets of parameters that might exhibit better effects can be considered in future.

Neuron No.	1	2	3	4	5	6	7	8
1	×	1	1	1	1	1	1	1
2	1	×	1	1	1	1	1	1
3	1	1	×	1	1	1	1	1
4	1	1	1	×	1	1	1	1
5	1	1	1	1	×	1	1	1
6	1	1	1	1	1	×	1	1
7	1	1	1	1	1	1	×	1
8	1	1	1	1	1	1	1	×

Figure 19.2 Initial weight settings of the network

In Figure 19.2, all the weights have been set and fixed at '1'. Before STDP is implemented to the program to regulate the changing of weight, the influence of neurons on each other through their connections has been kept at a reasonable value.

Now, the spiking network requires the first input (I) stimulus to make the artificial life form 'alive'.

Most of the inputs the artificial life form receives at each movement are coming from its external environment which will be described in Section 3.4 of the thesis. However, some noise can be injected as random input stimuli to the spiking network to make the artificial life form move before configuring any external environment.

3.3.2 Injecting Noise

The inputs applied to the 8 neurons of the spiking network have been stored in an 8*1 matrix 'I'. All values inside the input matrix 'I' are assigned with value zero initially at the start of the simulation time window. The noise is defined just after the matrix I' is created, given by the equation:

$$I(\text{ceil}(N * \text{rand})) \leftarrow \text{mean} + \text{rand} * \text{fluc} \quad \text{eq7}$$

'rand' is the Matlab command that will generate uniformly distributed pseudorandom numbers between 0 and 1 [17]. For simplicity, the project uses uniform distribution instead of normal distribution which is more similar to the distribution of noise in real neurons according to experimental studies. Normal distribution will be the design choice for future works. The variable 'fluc' at right part of eq7 represents the magnitude of fluctuations of the stimulus while the variable 'mean' stands for the offset value of the stimulus. This offset value has an impact on the actual mean of the noise but they are not the same, which should not be confused. The left part of eq7 indicates that the stimulus will be randomly injected to anyone of the 8 neurons at each simulation time step. That means only 1 neuron will receive the noise input at each simulation time step in this project. However, in real biological systems, there is noise in all the neurons, and again for the design simplicity, such design choice will be left for future works.

All the neurons used in the spiking network of this project have been set to excitatory regular spiking (RS) neurons with 0.02, 0.2, 65, and 8 as the values of neurons' parameters a, b, c, d respectively. Please refer to Section 2.33 in Chapter 2 of the thesis for the detailed firing pattern and dynamics of a regular spiking (RS) neuron. The membrane potential 'v' and recovery variable 'u' have been set to the value '-65' and '13' respectively. (Due to time limit, the other types of neuron will not be used in this project and will be concerned as future works at the end of the thesis.)

As soon as the noise is implemented to the spiking network when the start button in the layout window is pressed by user's mouse click, the spiking networks start to receive some random inputs; the membrane potential 'v' of the neuron starts to increase; and the neuron that receives the random input starts to fire with regular spiking patterns. When the whole simulation time step is completed, the membrane potential 'v' of the firing neuron will reset to the resting potential (-65) if it reaches the apex value. That is the natural behavior of a neuron simulated by the Izhikevich model. During each simulation time step, all the firing neuron number along with its corresponding firing time will be stored in the variable 'firings'. After the simulation time steps run 400 times, which means the time window of one movement is completed, the inner simulation loop will stop and come into the second loop-the movement loop (see Figure 12). At current stage, the spiking network will compare the number of spikes that has occurred inside each neuron at the end of this simulation time window according to the information stored in the

variable 'firings', and the neuron number that fires the maximum number of time will be recorded using Matlab command 'max'. This neuron number will then match its corresponding direction 'case' inside the 'switch' statement and the artificial life form will start to move. As soon as the display has been updated, the robot will change its position and the user could see a moving artificial life form in the layout window. However, after a few tests, the artificial life form can be observed moving more than one grid sometimes! Recall from Section 2.3 in Chapter 2, 'more than one neuron could fire with the same max number of spikes even with the effect of noise', that is the reason why this happens. A solution is also applied which leads to the implementation of the population vector.

3.3.3 Implementation of the population vector

As mentioned in Section 2.24 in Chapter 2, the population vector combines the summed activity of all the neurons and calculates the overall moving direction of the artificial life form; each neuron will contribute to its own vote in this case. Recall from Figure 17, equ4 and equ5 in the previous section of this chapter that each of the 8 neurons has its own corresponding moving computational value, what the program needs to do is to sum the computational values of all the neurons that fire the maximum number of spikes together, and match the results back to its corresponding moving direction, which will be the final resultant movement.

The neurons' number that fires the maximum number of spikes is compared in the variable 'firings' and stored in the variable 'sum_fired' at the end of the simulation time window.

In order to implement the population vector, the program creates a 1*2 matrix variable 'moving_calc' and assigns an initial value [3,3] to it.

moving_calc ← [3,3]

[3,3] indicates the value for artificial life form's original population vector. These numbers are only used as a middle point reference value for the movement decision's case matching. The first number indicates the vertical vector; the second number indicates the horizontal vector. When the information in the variable 'sum_fired' are ready for use, the computational values of all the neurons that fire the maximum number of spikes will be summed and added to the variable 'moving_calc'. The value of the computational result and its corresponding direction matching will be shown in Figure 20 below.

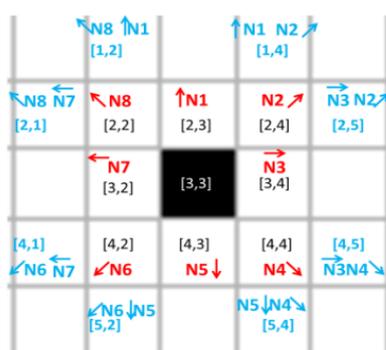


Figure 20 Value of population vector and its corresponding direction matching

In Figure 20, the parts marked with bold blue color are the case that the neuron moves more than

one grid during a movement step, which is the particular combination result of two specific directions.

For instance, if N2 fires the maximum number of spikes, the computational process will be given as the 2 equations listed below:

$$moving_calc(1) \leftarrow moving_calc(1) - 1 \quad eq8$$

$$moving_calc(2) \leftarrow moving_calc(2) + 1 \quad eq9$$

The resultant value of variable 'moving_calc' will be [2,4] as shown in Figure 20.

If N6 fires the maximum No. of spikes, the computational process will be given as the equations:

$$moving_calc(1) \leftarrow moving_calc(1) + 1 \quad eq10$$

$$moving_calc(2) \leftarrow moving_calc(2) - 1 \quad eq11$$

The resultant value of variable 'moving_calc' will be [4,2] as shown in Figure 20.

Now, if both N2 and N6 fire the maximum number of spikes simultaneously, the program will combine the equation eq8, eq9, eq10 and eq11 together:

$$moving_calc(1) \leftarrow moving_calc(1) - 1 + 1 \quad eq12$$

$$moving_calc(2) \leftarrow moving_calc(2) + 1 - 1 \quad eq13$$

The resultant value of variable 'moving_calc' will be [3,3] which is a combinational value of N2 and N6. According to the information shown in Figure 20, the artificial life form will remain still, which means the effect of N2 and N6 has cancelled out with each other.

Now, how the parts that marked with bold blue color in Figure 20 can be dealt with? The artificial life form of this program is designed to move only one grid at each movement step. Recall from the Section 2.3 in Chapter 2, if the particular combination such as N1 and N2 fires with the same maximum number of spikes during the same simulation time window, the spiking network controller will randomly choose from one of them, which also increases the degree of randomness of the artificial life form. Therefore, the artificial life form will only move one grid at each movement step no matter what happens.

For example, if N1 and N2 fire the same maximum number of spikes, their summed value of the population vector will be [1,4], and if this value appears at the end of the simulation time window, additional calculations will be given as the 2 equations listed below:

$$moving_calc(1) \leftarrow moving_calc(1) + 1 \quad eq14$$

$$moving_calc(2) \leftarrow moving_calc(2) - round(rand) \quad eq15$$

The command 'round(rand)' at the right of eq15 will produce a value '1' or '0' with equal

opportunities. The overall resultant value for the population vector will be either [2,3] or [2,4]. Refer to the information shown in Figure 20, the artificial life form in this case will move towards either N1 or N2 with the same chances.

When the relative settings for all neurons are finished, the user will observe an artificial life form moving randomly with one grid at each movement in the layout window. Now, the program is ready to assign the artificial life form some tasks which require the design of the external environment. The detailed implementation of the external environment will be explained in the next section of this chapter.

3.4 Environment Implementation

3.4.1 Assign different colors to various objects

As mentioned at the start of Section 3.2, the grids of the axes are created in the function `'function grid_size_N_Callback(hObject, eventdata, handles)'` and defined in the grid variable `'handles.vis_grid'`. However, the whole axes window itself is created as an image in the function `'function []=plot_grid(handles)'`. To add various objects with different colors to the environment, the program should define the color properties of the axes window first.

Matlab provides 'color image' for drawing of complex figures. The 'color image' consists of 3 primitive colors: red(R), green (G) and blue (B). Each primitive color has been assigned with values from 0 to 1, representing their strength in the resultant combined color. A table that lists all the RGB values used in this program and their corresponding colors, representation and image value is given below in Figure 21.

R	G	B	Color	Image value	Representation
1	1	1	White	0	Background Color
0	0	0	Black	1	Artificial Life
1	0	0	Red	2	Food
1	1	0	Yellow	3	Predators
0	0	1	Blue	4	Obstacles
0	1	0	Green	5	Trajectories
0	0.8	0		6	
0	0.6	0		7	
0	0.4	0		8	
0	0.2	0		9	

Figure 21 Table showing RGB values in this program

In Figure 21, the color 'Green' becomes darker as its RGB value changes from 7th to 11th row of the table. These green colors with different brightness will be used to represent the trajectories of the artificial life form, which will be introduced later in this section. The 5th column of the table represents the image value of the corresponding colors used in this program.

The whole RGB value is stored in matrix 'MM' in the program and this matrix is then added to the color property of the axes window by the Matlab command `'colormap'`. After that, the axes' color property is divided into 9 values and displayed as an image that can show 9 different color properties to the screen by Matlab command `'imagesc'`. (The relevant Matlab code: `'imagesc(handles.vis_grid,[0 9]);'`). Once the color image of the program has been set, these color properties are assigned to different objects as shown in Figure 21 above through the drawing function `'function[a]'`.

Recall from Section 3.2, the position coordinates of each grid in the axes are defined in the variable 'handles.vis_grid'. Therefore, in order to display various objects in the layout window, the program just needs to assign the object's corresponding image value (listed in the 5th column of Figure 21) to the required position in the variable 'handles.vis_grid' and updates it in the handles structure. For example, if the program wants to display an artificial life form, a food and a big square obstacle in the layout window, the required equation in the handles structure will be given as:

$$\text{handles.vis_grid}(30,30) \leftarrow 1 \text{ (Artificial life form)} \quad \text{eq16}$$

$$\text{handles.vis_grid}(15 : 25,35 : 45) \leftarrow 4 \text{ (Square obstacles)} \quad \text{eq17}$$

$$\text{handles.vis_grid}(8,52) \leftarrow 2 \text{ (Food)} \quad \text{eq18}$$

The result is shown below

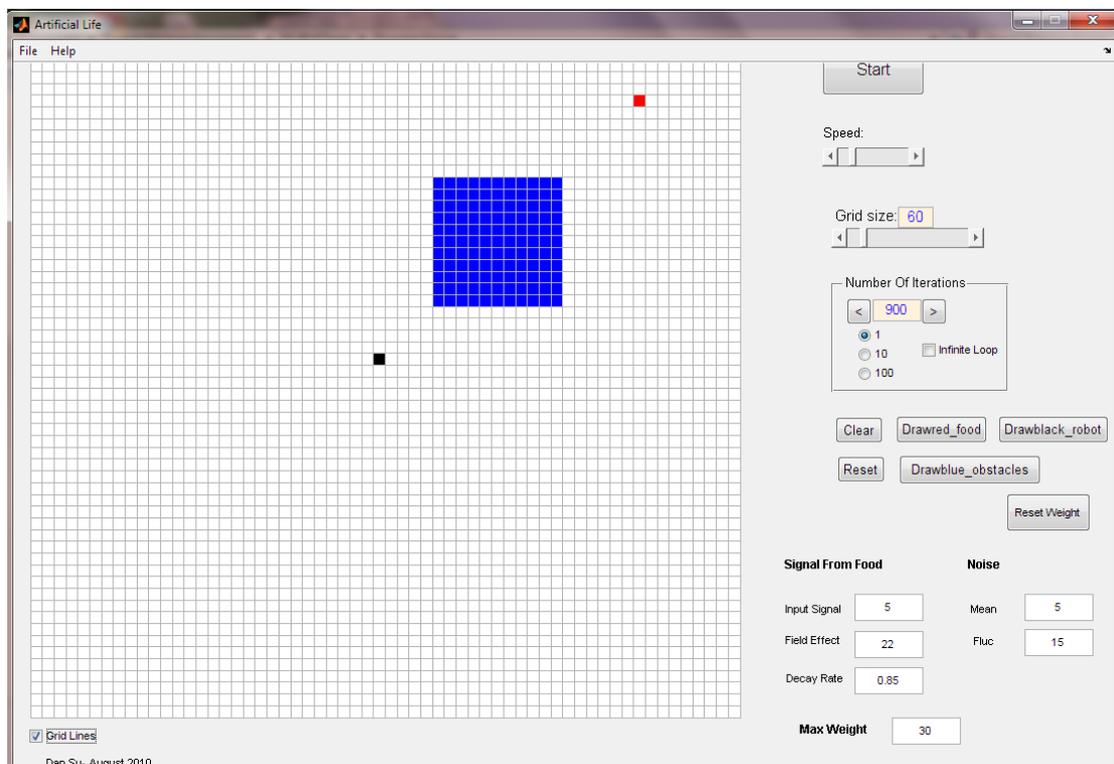


Figure 22 Display of an artificial life form, a food (red) and a square obstacle (blue)

3.4.2 Representation of trajectories of the artificial life form

In Figure 21, the trajectories of the artificial life form have been assigned to color green with different brightness. The brightness of the color depends on the number of passing time of the artificial life form going through the same grid. The reason to do this is to provide the user a more understandable presentation, so that they could see a more detailed running process of the artificial life form displayed in the layout window through pictures and images.

The number of passing time of the artificial life form going through each grid will be stored in the variable 'handles.numberpass_grid'. Each time the artificial life form goes through a specific grid,

the number stored in that grid will be updated according to its corresponding position coordinates in the variable 'handles.numberpass_grid'. The equation is given as follows:

$$handles.numberpass_grid(RIL, CIL) \leftarrow handles.numberpass_grid(RIL, CIL) + 1$$

Eq19

RIL and CIL represent the coordinates of the specific grid that the artificial life form is going through.

These numbers of passing time are then divided into 5 levels and assigned to 5 different image values. A table that lists the number of passing time with their corresponding image value is given below:

Number of passing time	Image value
$1 \leq NPA \leq 4$	5
$4 < NPA \leq 8$	6
$8 < NPA \leq 30$	7
$30 < NPA \leq 60$	8
$60 < NPA$	9



Figure 23 5 Level of passing time with their corresponding image values

As shown in Figure 23 above, the more times the artificial life form goes through the same grid, the darker the color will appear on that grid in the layout window.

Recall from eq4 and eq5 in Section 3.2 of this chapter, when the artificial life form moves into a grid, the program will update the image value of that grid through eq4 to '1' which represents black. As soon as the execution of eq4 is finished, the program will clear the image value of the grid the artificial life form is leaving behind through eq5 back to '0', which means the grid's color has been set back to the axes original background color--white. Now, to implement the trajectories of the artificial life form, the program just needs to replace eq5 with a set of new equations. So that when the artificial life form leaves one grid, the image value of that grid will be set back to a numerical value from '5' to '9' instead of '0'. As a result, the artificial life form will leave a trace with different brightness of green behind. For instance, if the number of passing time through a particular grid is 22 times and the artificial life form is about to move onto that grid, the process of the computation will be given as the two equations shown below:

$$handles.vis_grid(RI - 1, CI) \leftarrow handles.vis_grid(RI, CI) \quad eq4$$

$$handles.vis_grid(RI, CI) \leftarrow 7 \quad eq20$$

Now, the user can observe how the artificial life form moves randomly with trajectories displayed on the layout window through mere images.

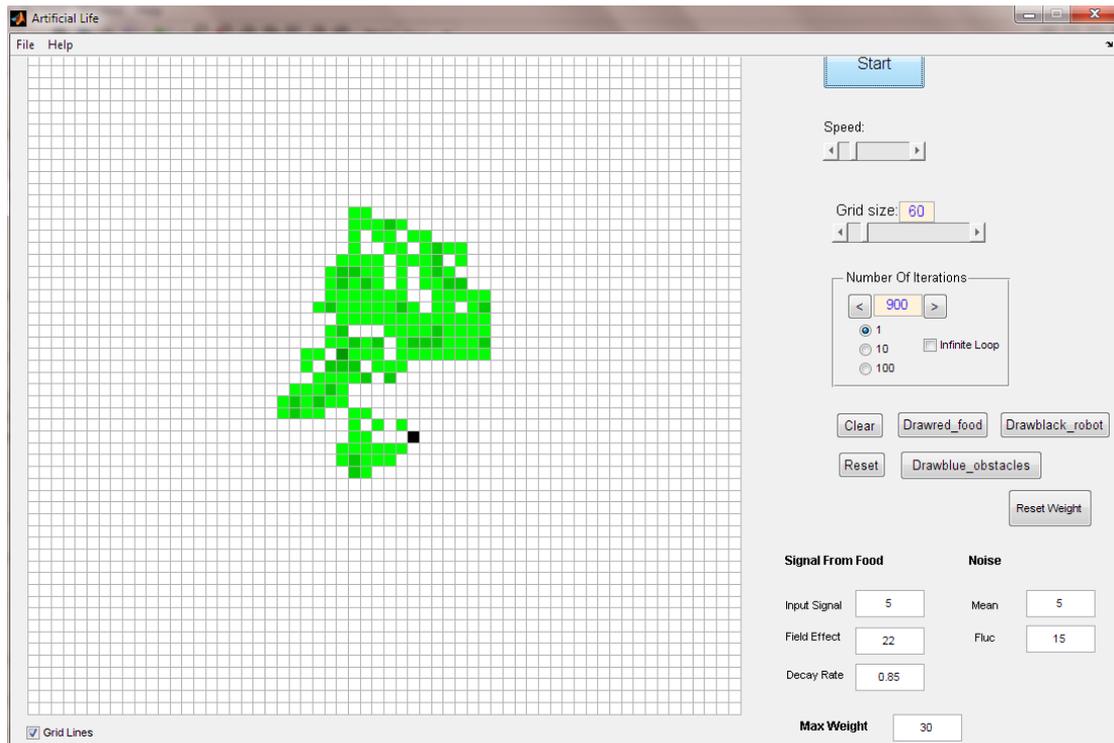


Figure 24 Image of an artificial life form moving randomly

In Figure 24, the green color represents the trajectories of the artificial life form. The darker the green is, the more times the artificial life form has passed through that grid.

3.4.3 Adding signals to food--the implementation of the direction coding scheme

In Figure 22, the red grid in the upper-right corner of the axes represents a food. The artificial life form does not move towards that food because there are no signals from it. Now, the program is going to add positive signals to the food so that the food will trigger additional input in the neuron which points to its direction.

The artificial life form can 'detect' the direction of the food by comparing its position coordinates with the food's position coordinates during each movement. For instance, if the food is located to the upper right direction of the artificial life form, both horizontal and vertical coordinates of the food will be greater than the coordinates of the artificial life form within the axes in that case. Once the comparison result is coming out, an additional input will be triggered in N1 which is the neuron that points to the upper right direction of the artificial life form (please see Figure 1&2 in Chapter 2). The value of the additional input triggered is stored in the variable '`handles.input_source`' and the equation will be given as follows:

$$I(1) \leftarrow I(1) + input_source \quad eq21$$

The strength of the additional input is set to '27' in order to overcome the effect of noise at the current stage. Now the artificial life form is able to move towards the food.

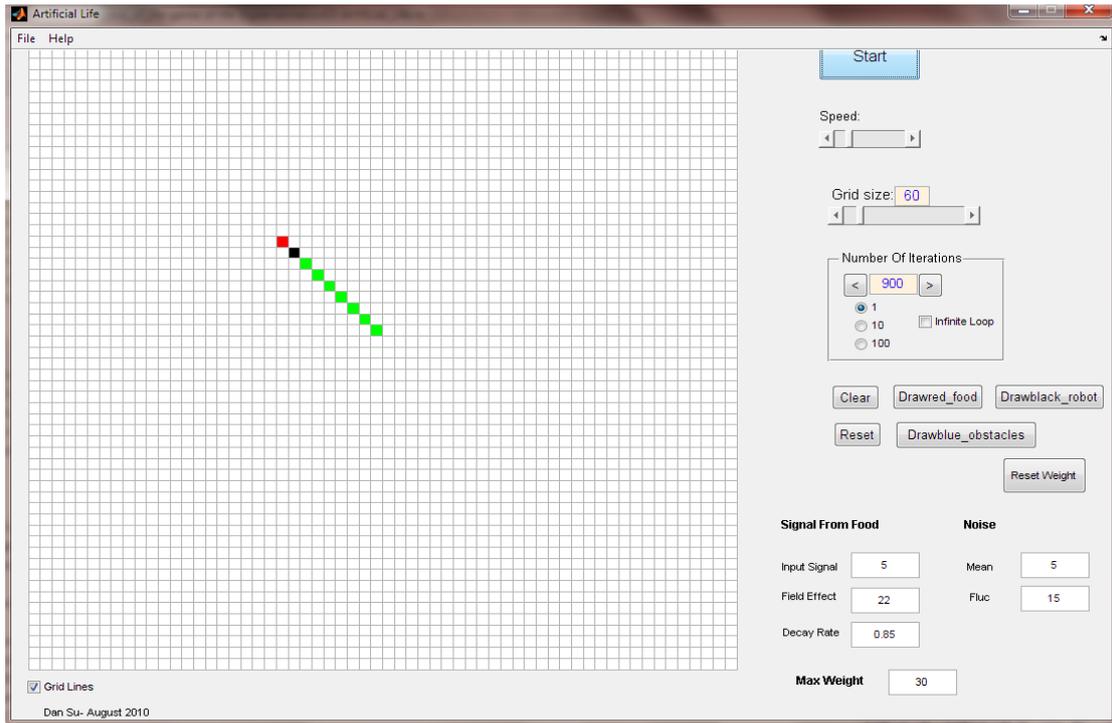


Figure 25 Artificial life form moving towards the food

In Figure 25, the food is located to the upper left of the artificial life form, additional input is triggered in Neuron 8, and the artificial life form is moving towards that direction

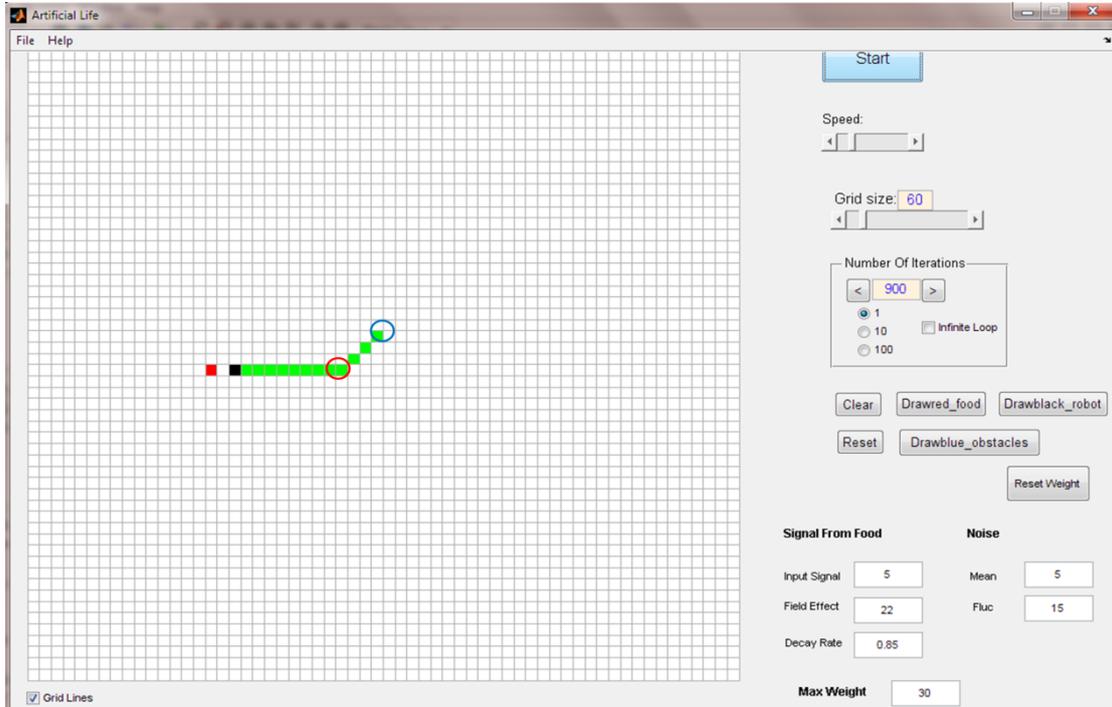


Figure 26 Artificial life form moving towards the food

In Figure 26, the food is located to the down-left of the artificial life form, and the blue circle marker indicates the artificial life form's starting position. At the beginning of the running process,

additional input is triggered in neuron 6 and the artificial life form moves diagonally down to the left. However, when the artificial life form moves to the point where the red marker indicates, and the direction of the food switches from 'down-left' to 'left', neuron 7 will receive additional input instead of neuron 6, and the artificial life form will move left and catch the food finally.

3.4.4 Assign field to the signals of the food-implementation of the distance coding scheme

If there is more than one food in the environment, implementation of the distance coding scheme allows the artificial life form to detect how far away the foods are. With this coding scheme implemented as a rule of external environment, the artificial life form will always move towards the nearest food, and the efficiency of the tasks can be greatly increased.

Similar to the method used in the previous Section 3.4.3, the program can calculate the artificial life form's distance from the food by performing 'subtraction' and 'absolute' operations between the coordinates of the food and the artificial life form. As soon as the food's distance from the artificial life form is known, the 'field' will be assigned to the signals of the food, which means the signal strength of the food will be inversely proportional to its distance from the artificial life form—the longer the distance is, the weaker the strength of the signal will be. The equation of the field is given as:

$$field(d + 1) \leftarrow A * field(d) \quad eq22$$

In eq22, 'd' represents the distance from the artificial life form to the food. 'A' describes how the values of the field strength change as the distance increases. In this project, 'A' is treated as an exponential decay and implemented into the program using AR (Autoregressive) process approximation. The equation of A is given:

$$A \leftarrow e^{-\Delta d / \tau} \quad eq23$$

τ in eq23 represents the decay constant of the exponential and its value has been chosen carefully and given as '6.153' in this program. The value of 'A' can then be calculated and the result is 0.85. To substitute the value of 'A' into eq22 will get:

$$field(d + 1) \leftarrow 0.85 * field(d) \quad eq24$$

The maximum field strength has been set to value '22' and the equation is:

$$field(1) \leftarrow 22 \quad eq25$$

A for loop 'dd' has been used to assign all the other field values to their corresponding distances and implement them into the program. Another for loop named 'm' is implemented outside the for loop 'dd', which allows the artificial life form to 'scan' and detect the distance of all the food around it. Now, recall from previous Section 3.4.3, the variable 'input_source' in eq21 will be divided into 2 parts: one is the basic signal strength of the food stored in the variable 'input_source' with its value being set to '5' currently which will be fixed in spite of any change in distance. The other is the field of the food, and its strength will decrease as the distance from the artificial life form to the food increases. The final input updating equation from the food at each simulation time step will be given as:

$$I(N) \leftarrow I(N) + input_source + field(d) \quad eq26$$

The distribution of the strength of the signals within 3 grids surrounding a food is shown in Figure 27 below.

15.9+5	15.9+5	15.9+5	15.9+5	15.9+5	15.9+5	15.9+5
15.9+5	18.7+5	18.7+5	18.7+5	18.7+5	18.7+5	15.9+5
15.9+5	18.7+5	22+5	22+5	22+5	18.7+5	15.9+5
15.9+5	18.7+5	22+5	22+5	22+5	18.7+5	15.9+5
15.9+5	18.7+5	22+5	22+5	22+5	18.7+5	15.9+5
15.9+5	18.7+5	18.7+5	18.7+5	18.7+5	18.7+5	15.9+5
15.9+5	15.9+5	15.9+5	15.9+5	15.9+5	15.9+5	15.9+5

Figure 27 Distribution of the signals surrounding a food

In Figure 27, '22' is the maximum field value when the distance from the food is 1. It can be seen that its strength decreases as the distance increases: $22 * 0.85 = 18.7$, $18.7 * 0.85 = 15.9$. '5' is the value of input source constant which will not change with distance. One thing to notice is that as the field strength decreases, the noise might start to take effect. Recall from Section 3.32, the strength of noise has been set to '22' which is larger than the signal value located 3 grids away from the food. That triggers the competitions between noise and target at certain stages. Relative tests on these competitions will be conducted in the first section of Chapter 4.

Now, once the start button is pressed, the artificial life form will move towards the nearest food.

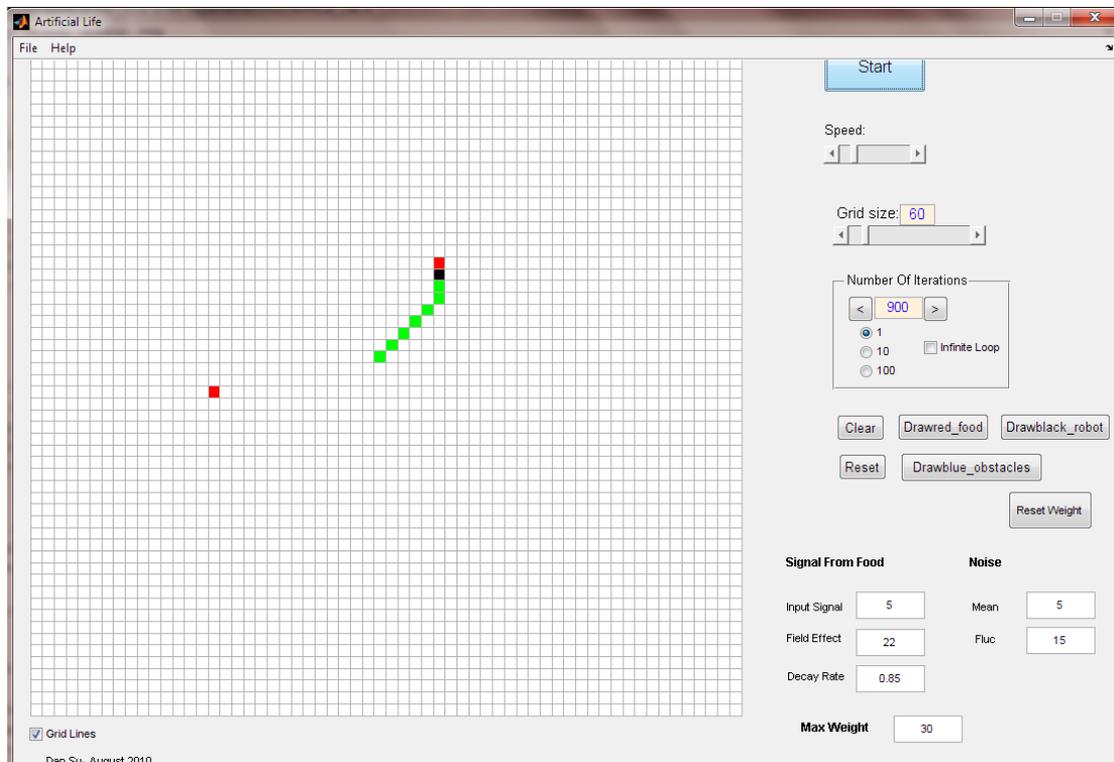


Figure 28 Artificial life form moving toward the nearest food

In Figure 28, at the very start of the running, the program will first scan the surrounding area and

detect the direction and distance information of the 2 foods (using the for loop 'm'). Both N2 and N6 will receive input signals from the foods. However, the input N2 is receiving will be stronger than the input N6 is receiving since the food located at the upper-right is closer to the artificial life form. N2 will fire more number of spikes than N6 within one simulation time window in that case and thus the artificial life form will move upper-right.

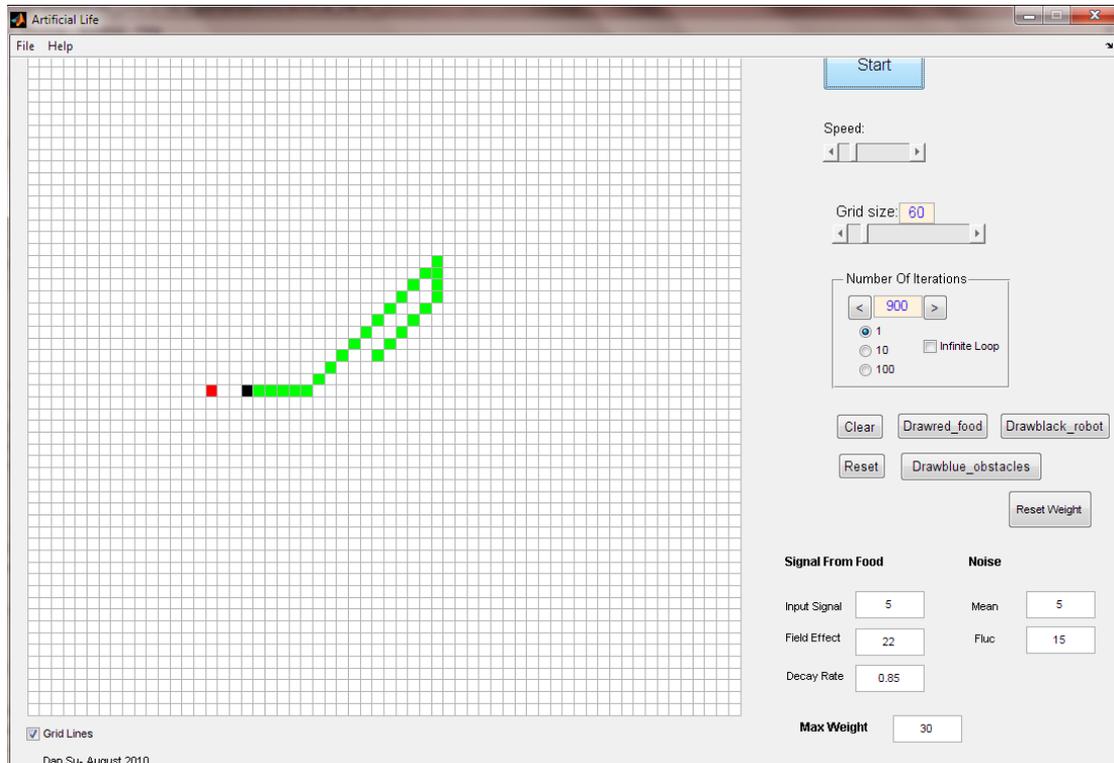


Figure 29 Artificial life form moving toward the second food

In Figure 29, once the artificial life form eats the first food, it will go and catch the second food.

3.4.5 Adding 'zero' signal to the obstacles

The artificial life form is designed to stop when it touches the edge of obstacles which are represented as blue grids in the axes of the layout window. To implement obstacles into the program, a 3*3 matrix 'chess_grid' is created to store the image values surrounding the artificial life form within one grid distance. The value of the matrix will be updated before each movement of the artificial life form, so that the artificial life form could detect whether there is any obstacle around it before moving.

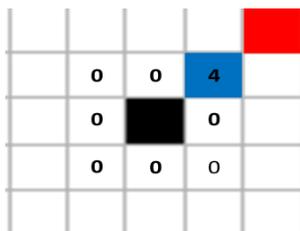


Figure 30 Artificial life form scanning the surrounding area's image value to detect obstacles

The program can stop the artificial life form by 'shutting down' the neuron that points to the direction of obstacles before the artificial life form is about to move into it.

In Figure 30, the artificial life form is attracted by the food located at the upper-right direction. Therefore N2 will receive inputs from the food first. However, when the artificial life form detects the image value '4' around it through matrix 'chess_grid', it will shut down N2 immediately. A neuron which has been 'shut down' means its membrane potential will be forced to be maintained at its resting state—65mV. The equation in the situation in Figure 30 will be given as:

$$v(2) \leftarrow -65 \quad \text{eq27}$$

Now, the artificial life form certainly cannot move towards N2 directions. However, as the noise is still working, the artificial life form gets an equal chance of moving towards the other directions and thus is able to go around the obstacles and catch the food finally.

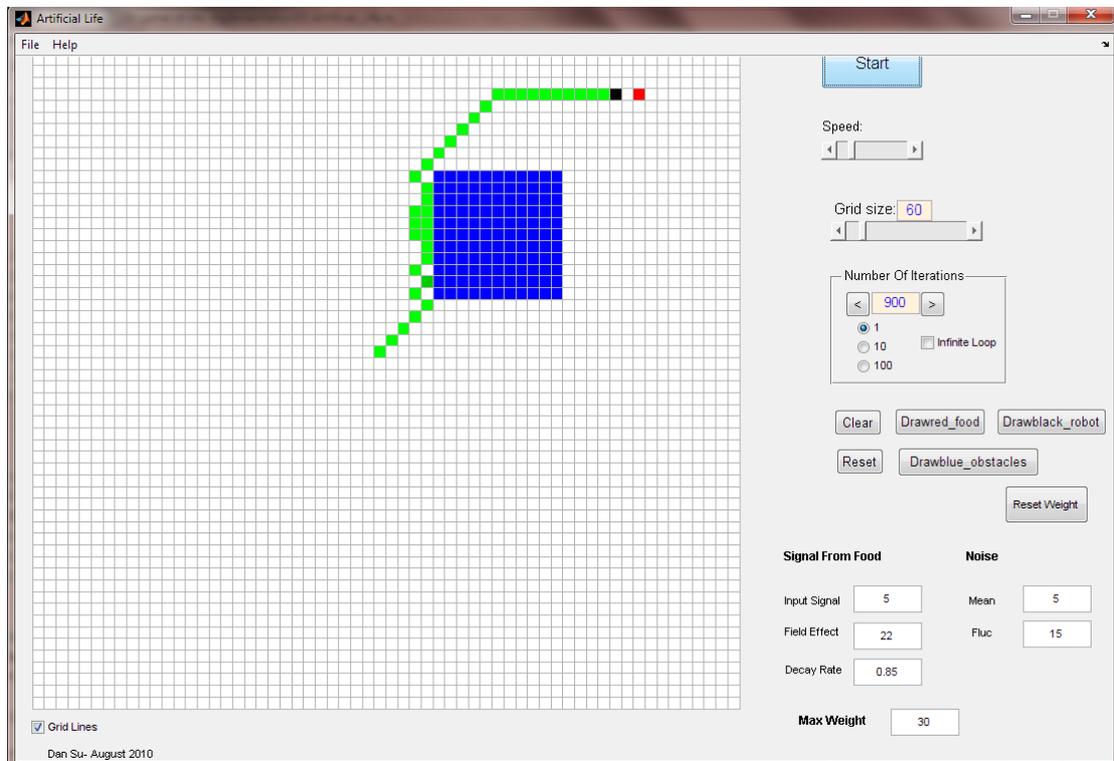


Figure 31 Artificial life form going around square obstacles and catch the food

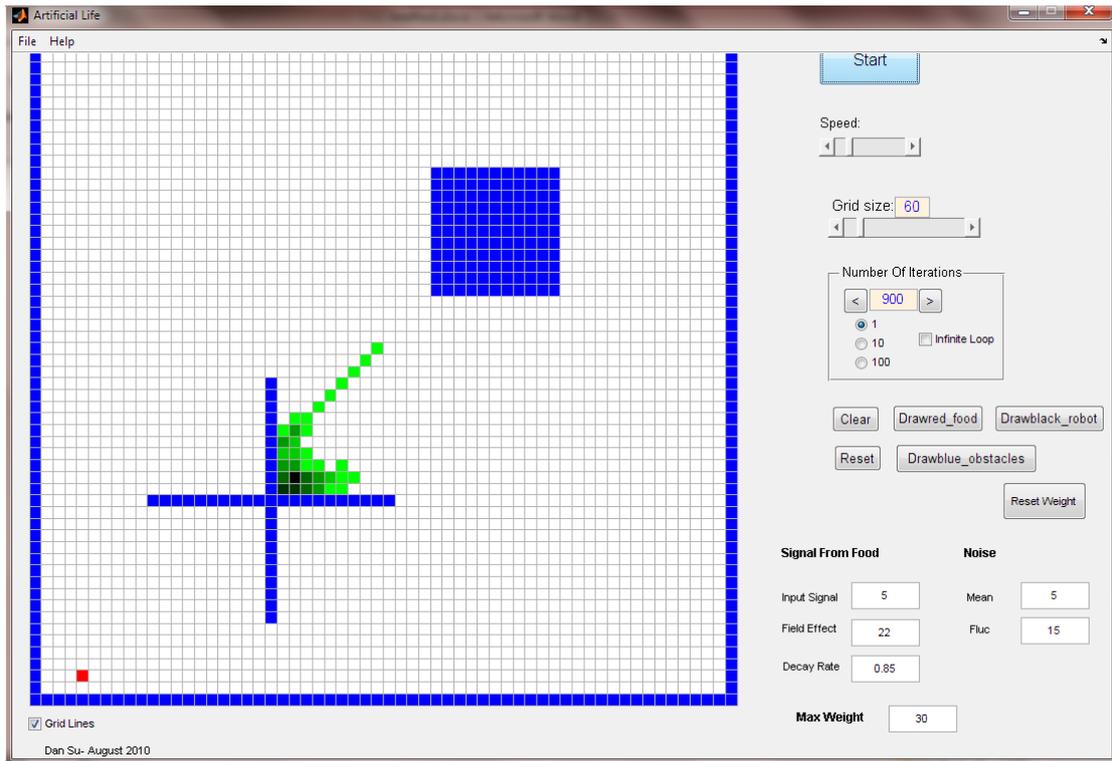


Figure 33 Artificial life form trying to go around the cross obstacle and catch the food

In the case shown in Figure 33, the artificial life form has never moved around the big cross obstacles and caught the food no matter how many movements have been taken. To increase the strength of noise (randomness) would decrease the efficiency of the tasks totally (Please see the test in Section 4.1). Therefore STDP has been implemented into the program in order to further develop the 'intelligence' of the artificial life form. All the weights between the connections of the neurons have been set and fixed at value '1' currently. By regulating the changing of these weight values, STDP could control the influence of neurons on each other and thus increase the causal interactions in the network. The artificial life form should be able to self-evolve and show some abilities of adaption to the external environment in that case. More relevant discussions can be found in Chapter 5. Tests on the behaviors of artificial life form under the control of STDP will be carried out under different conditions in Chapter 4. The project will compare and discuss the results and try to discover the most efficient behavior of the artificial life form under certain circumstances. The implementation of STDP is shown in the next section.

3.5 Implementation of STDP

3.5.1 Value and timing of STDP

Each of the 8 neurons has its own corresponding STDP value, and these values are stored as variables in the 8*400 matrix 'STDP' and assigned with value '0' at the start of the program. The values of STDP will evolve as the simulation time step increases. The 400 columns in matrix 'STDP' store the values of STDP at each simulation time step within the time window. Recall from Chapter 2, Figure 13 shows that the variation of the value of STDP depends on the relative timings of pre and postsynaptic spikes. To associate the timing of spikes with the timing of STDP, the value of STDP will reset to 0.1 whenever it detects that the corresponding neuron fires. For example, if neuron N1 fires at time t, its value of STDP will be updated using the equation below (assume the case is potentiation):

$$STDP(1, t + D) \leftarrow 0.1 \quad \text{eq28}$$

In eq28, value '1' represents the fired neuron number which is stored in the variable 'fired' at each time step. t+D means this change will take effect after D seconds which has been set to 20ms in this program according to the original Izhikevich model settings. D is the conduction delay between neurons. When pre synaptic neuron fires, its spike will reach postsynaptic neuron after 20ms, therefore the effect of this firing activity will only take effect after 20ms, and so does the STDP reset corresponding to this firing activity. 0.1 is the maximum value of STDP used to potentiate the value of the weight when the time intervals between pre and postsynaptic spikes are at their minimum values. The maximum value of STDP for depression is given by 0.12. The depression has been set stronger than potentiation in this project in order to strengthen the causal interactions between the neurons. The relevant issue will be discussed in Chapter4 and 5 in detail.

In Figure 13 [10], the variation of value of STDP against the positive and negative time intervals can be represented as 2 exponential decay curves. The equations and parameters have been given as follows (taken from reference [10]):

$$A_+ e^{-t / \tau_+} \quad \text{eq29}$$

$$A_- e^{-t / \tau_-} \quad \text{eq30}$$

Where $A_+ = 0.1$ $A_- = 0.12$ time constant $\tau_+ = \tau_- = 20ms$

Values of A_+ and A_- have already been set by eq28 (However, A_- requires a stronger value than A_+ which will be described later). To implement the exponential into the network, the program uses the same method introduced in the previous section. AR (the autoregressive) process has been used to approximate the exponential curve of STDP at each simulation time step. The equation will be given as:

$$STDP(t + 1) \leftarrow B * STDP(t) \quad \text{eq31}$$

Where $B = e^{-t / \tau_+}$ or e^{-t / τ_-}

According to eq29 and eq30, value of the time constant τ is also known as 20ms. The value of B could be calculated and the result is $e^{-1/20} = 0.95$. Now, adding the connection delay D to the equations, the final updating equation for the value of STDP at each simulation time step will be given as:

$$STDP(:, t + D + 1) \leftarrow 0.95 * STDP(:, t + D) \quad \text{eq32}$$

The variation of STDP against simulation time step within the time window is plotted below.

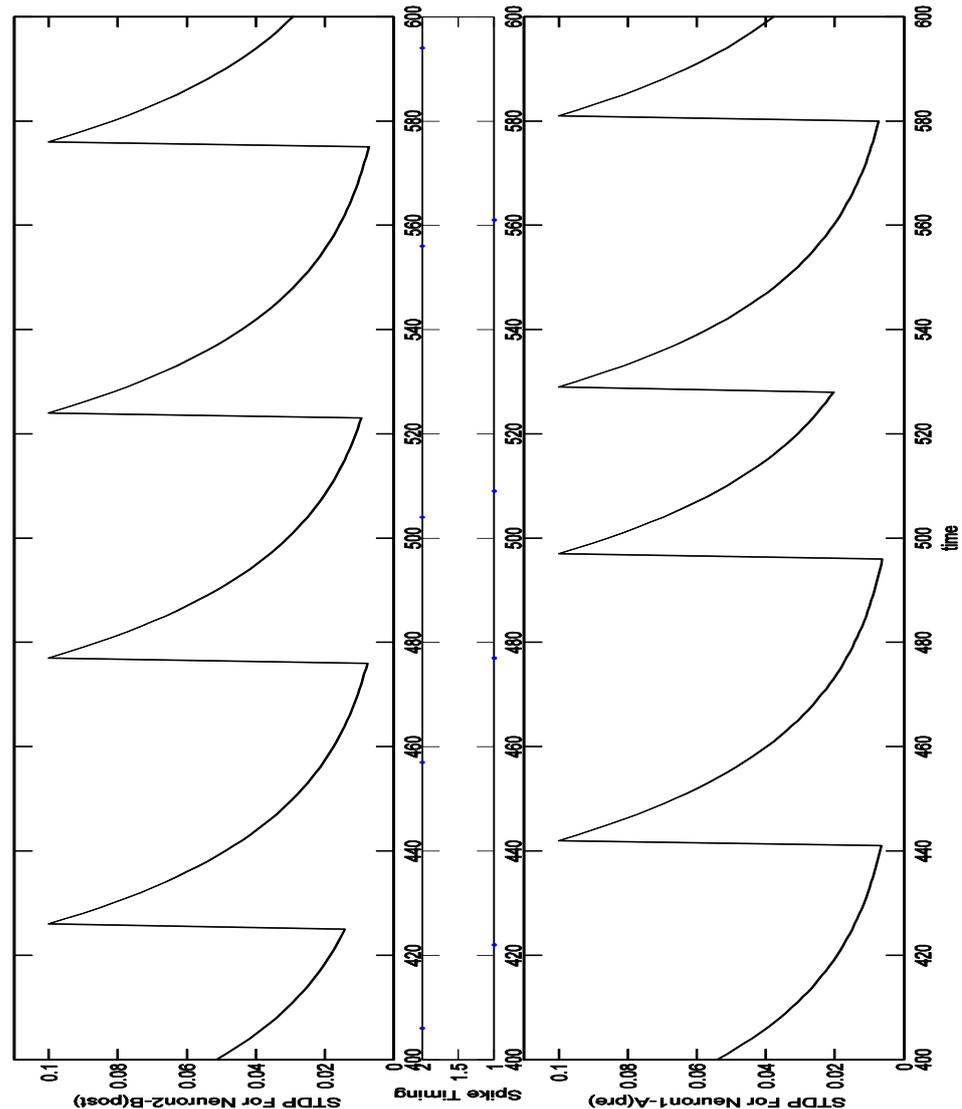


Figure 34 Plot of variation of STDP against time

For the demonstration purpose, the time window has been enlarged from 400 to 1000 simulation time steps in order to give sufficient presentation. The diagram intersects the part from the time step 400 to 600. The blue spot in the middle of the diagram represents the firing time of corresponding neuron. As mentioned above, it can be seen that the value of STDP has been reset after 20ms after a neuron fires. By using the resetting value of STDP (0.1or0.12) and its corresponding timing within the time window as a reference point, the model can associate the time interval between the pre and postsynaptic spikes with the value of STDP. A detailed

explanation of this association will be presented in the next two sections.

3.5.2 Potentiation

Instead of changing the synaptic weight ' W_{ij} ' directly, the program will change their derivatives ΔW_{ij} and update the weights each 3 real time movements of the artificial life form (please see the outer most loop--the start and stop loop in Figure 15). More details about this updating algorithm and their implementation will be described in Section 3.5.4. The value of ΔW_{ij} for all ' W_{ij} ' is stored as variable in the 8*8 matrix ' sd ' in this program.

In this section, the thesis will describe the potentiation of the synaptic weight. Recall from Figure 12 in Chapter 2, if a presynaptic spike arrives before the postsynaptic neuron fires, the synaptic weights between them will be potentiated while the strength of potentiation depends on their timing intervals, and such information is to be stored in the variable ' $STDP$ ' in the program as mentioned in previous section.

In this project, the implementation of potentiation is depicted as follows: For each fired neuron, the program will search all its presynaptic neurons and determine the last excitatory spike that arrives from these neurons [10]. Since these spikes make the neuron fire, the synaptic weights are potentiated according to the value of STDP at the presynaptic neuron adjusted for the conduction delay [10]. This corresponds to the positive part of the STDP exponential curve in Figure 13. The updating equation for potentiation at each simulation time step will be given as:

$$\Delta W_{ij} \leftarrow \Delta W_{ij} + STDP(fired, t) \quad \text{eq33}$$

In eq33, j indicates the postsynaptic neuron that fires at current time step t , where i is one of the presynaptic neurons that connects to neuron 2, and its neuron number is stored in the variable ' $fired$ ' as mentioned above.

Take Figure 34 as an example. At time step 458, if Neuron 2 fires, its synaptic connection with presynaptic Neuron 1 will be updated as follows:

$$\Delta W_{12} \leftarrow \Delta W_{12} + STDP(1,458) \quad \text{eq34}$$

A diagram intercepted from part of Figure 13 demonstrating this updating algorithm is shown below.

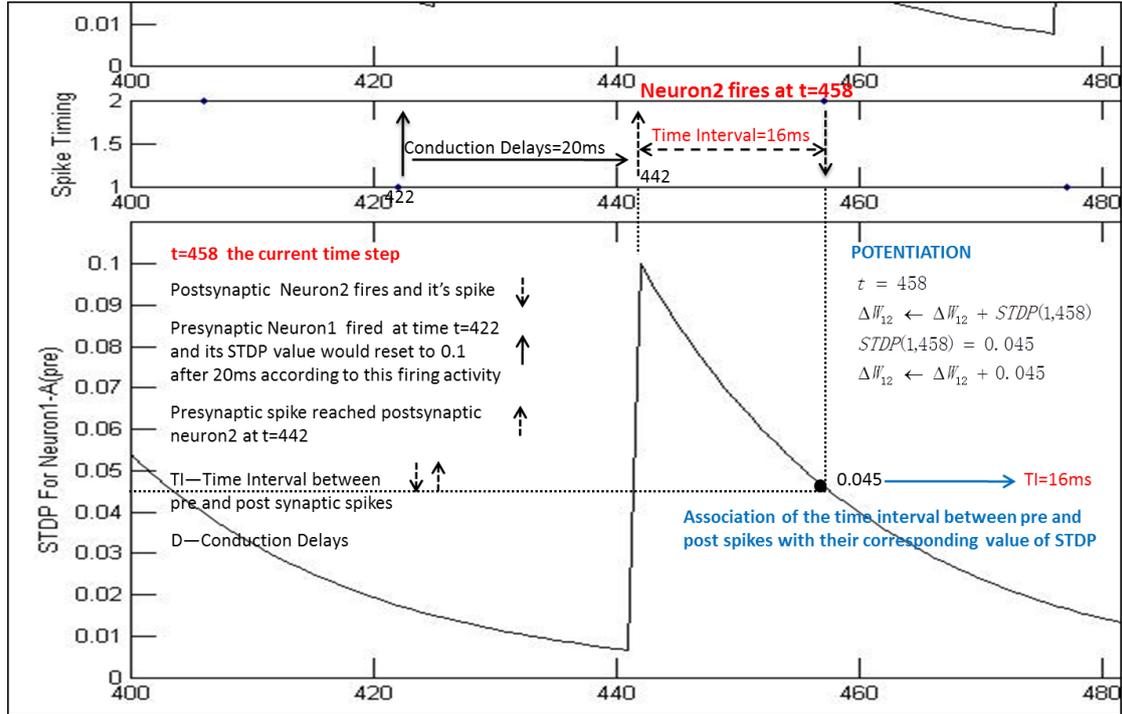


Figure 35 Association of the time intervals between pre and post synaptic spikes with their corresponding values of STDP during potentiation

In Figure 35, at simulation time step $t=458$, postsynaptic neuron 2 fires and the program will search all the excitatory presynaptic neurons that connect to it and potentiate their synaptic connections. Neuron 1 is one of these presynaptic neurons that fires before neuron2 and its spike arrives at Neuron2 at time $t=442$. Their spikes time interval is therefore $458-442=16\text{ms}$. The program then potentiates their synaptic connections according to neuron1's value of STDP corresponding to this time interval as shown in the right part of the diagram. The same process will be duplicated with the other neurons at each simulation time step if the required circumstances are met.

3.5.3 Depression

In contrast with potentiation, if the presynaptic spike arrives right after the postsynaptic neuron fires, the synapse is depressed as shown in Figure 12.

In this project, the implementation of depression is depicted as follows: when a presynaptic spike arrives at a postsynaptic neuron [10], the program depresses the synapse according to the value of STDP at the postsynaptic neuron [10]. This corresponds to the negative part of STDP exponential curve in Figure 13.

Now, at current simulation time step t , the equation is given below

$$\Delta W_{ij} \leftarrow \Delta W_{ij} - 1.2 * STDP(\text{post}\{\text{firings}(:, t - D + 1)\}, t + D) \quad \text{eq34}$$

In eq34, i represents a series of neuron numbers which fire at time $t-D+1$. This series of numbers could also be represented as $\text{firings}(:, t - D + 1)$. Neuron i 's spike will certainly arrive at

the postsynaptic neuron j at time $t+1$ in future. j in this equation represents a series of neuron numbers which include all the postsynaptic neurons that neuron i connects to. This series of neuron numbers can also be represented as $post\{firings(:, t - D + 1)\}$.

Now, another example is plotted and explained below. At current simulation time $t=429$, the program will search the neurons that fire at time $t-D+1=410$, and in this case, Neuron1 fires at $t=410$. The example will show the depression of the synaptic connections between Neuron1 and one of the postsynaptic neurons that Neuron1 connects to which is Neuron2 in this example. The equation for the depression of the synaptic connection between presynaptic Neuron1 and postsynaptic Neuron2 at time $t=429$ is given below:

$$\Delta W_{12} \leftarrow \Delta W_{12} - 1.2 * STDP(2,449) \quad \text{eq35}$$

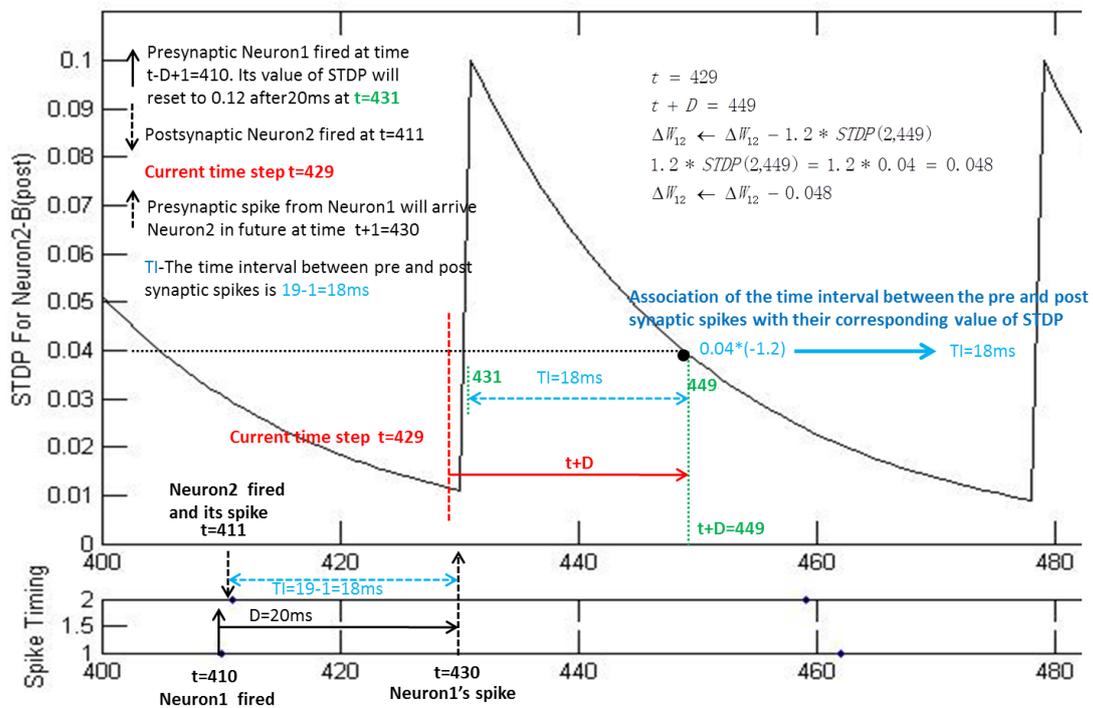


Figure 36 Association of the time intervals between pre and post synaptic spikes with their corresponding values of STDP during depression

If the spikes from pre and postsynaptic neuron occur at the same simulation time step, their time intervals is zero and the synaptic connection between them will be potentiated. The minimum time interval for the potentiation has been set to zero. Therefore in this project the minimum time interval for the depression has to be at least 1ms since the 0ms has been already assigned to the potentiation. In Figure 36, the current time step is $t=429$, Neuron2 fires at $t=411$, and Neuron1's spike will arrive at Neuron2 at $t=430$. Their spikes interval is originally 19ms represented in bold blue in the diagram. However, since the minimum of the time interval has been set to 1ms instead of 0ms, the whole reference point of value of STDP will also shift leftwards 1ms. As a result, the time interval in Figure 36 is treated as 18ms instead of 19ms. The program then depresses their

synaptic connections according to neuron2's value of STDP corresponding to this time interval as shown in the right part of the diagram. The same process will be duplicated with the other neurons at each simulation time step if the required circumstances are met.

3.5.4 Updating the weight W_{ij}

In the previous two sections, the thesis describes the implementation of potentiation and depression, and their relationships with the weight derivative values ΔW_{ij} . However, the changing of ΔW_{ij} alone will not influence the synaptic weight between neurons since the weight information is stored in W_{ij} instead of ΔW_{ij} . Now, the program needs to update the values of W_{ij} according to their derivatives ΔW_{ij} . The equation will be given below:

$$W_{ij} \leftarrow W_{ij} + \Delta W_{ij} \quad \text{eq36}$$

Eq36 will not be updated at each simulation time step or each movement like the other equations introduced in the previous sections. That is because the changing of weight (synapse) is not instantaneous, but is slow, taking many times to develop in real neurons [10]. In this project, the weight will be updated each 3 movements of the artificial life form (see Figure 15).

Now, all the weights in the network have been set to '1' initially and STDP is implemented to develop these weights according to the external environment of the artificial life form. Test on the behavior of the artificial life form under the control of STDP is shown in Section 4.3 in Chapter 4.

From the results in Sections 4.3 and analysis in Section 4.4, STDP has been proved to be working under different noise conditions, but still with relatively low efficiency. Now, the project is going to implement Direction Damping to the learning rule in order to make STDP work better in the next section.

3.5.5 Implementation of Directional Damping

From Figure 49 in Section 4.4, it can be seen that STDP working alone tend to produce bimodal distribution of weightings; most of the weights lie in the maximum and minimum boundaries which are 30 and -30 in this project. There are many ways of preventing this from happening. Direction Damping is one of the biological plausible methods that is used in this project in order to solve this problem. It is taken from Di Paolo E.A and his published paper "Spike-Timing Dependent Plasticity for Evolved Robots [14]." Direction Damping will slow down the changes that push a weight towards the boundary if the weight is near that boundary, and leave the changes that push the weight away from it unaffected. The closer the weight from the boundary, the slower rate of change is allowed to change towards that boundary. The algorithm for damping in this project is given by the equation:

$$\Delta W_{ij} \leftarrow \Delta W_{ij} * A * \zeta \quad \text{eq37}$$

Eq37 will be updated each three movements of the artificial life form since the weight W_{ij} itself will be updated with that rate as mentioned in the previous sections. ζ is the damping factor used to control the changing of weight under different conditions (maximum and minimum boundaries). It is stored in the variable 'zeta' in the program. A is the value used to control the scale of ζ in order to cooperate with the scales of the maximum and minimum boundaries. In this project, A has been set to 27 which has been proved to be appropriate when working with maximum and minimum synaptic weights of 30 and -30. The value of ζ and its corresponding equations under different conditions are shown below.

Before eq36 executes, the program will check the states of W_{ij} and ΔW_{ij}

Case 1: $W_{ij} > 0$ and $\Delta W_{ij} > 0$ (Weight is approaching the maximum value)

$$\zeta \leftarrow (5 - 4.9 * \frac{W_{ij}}{W_{\max}}) * 10^{-8} \quad \text{eq38}$$

Case2: $W_{ij} < 0$ and $\Delta W_{ij} < 0$ (Weight is approaching the minimum value)

$$\zeta \leftarrow (5 - 4.9 * \frac{W_{ij}}{W_{\min}}) * 10^{-8} \quad \text{eq39}$$

Case3: (Otherwise, Weight is not approaching either boundary)

$$\zeta \leftarrow 1 \quad \text{eq40}$$

Again, $5 * 10^{-8}$ and $4.9 * 10^{-8}$ are the scales selected to cooperate with the parameters within in the program. $5 * 10^{-8}$ represents the maximum value of changes which the damping allows when the weight has the largest distance from the boundary ($W_{ij}=0$). $4.9 * 10^{-8}$ represents the minimum value of changes which the damping allows when the weight has the closest distance from the boundary ($W_{ij}=30$ or -30). These values have been set relatively low in order to avoid the quick saturation rate of the spiking network in this program. Combining eq36--40 together, the equation is given:

$$W_{ij} = W_{ij} + \Delta W_{ij} * A * \zeta \quad \text{eq41}$$

However, this is not the final equation for updating the weight. The weight distribution now will tend to be unimodal and centered around where potentiation and depression equilibrate [14].

Recall from the previous sections, the strength of depression has been set stronger relative to potentiation in order to strengthen the causal interactions within the network. However, after damping is implemented into the network, this effect will be amplified. As a result the equilibrium of the weight distribution lies near the minimum boundaries (-30), which will consequently cause most neurons in the network to be 'silent' [10]. Therefore an activity independent value stored in variable 'ES' has been added to eq42 in order to potentiate the synapses coming to silent neurons, thus the equilibrium of the weight distribution will shift to the positive directions depending on the magnitude of ES. In Chapter 4, a numbers of tests are conducted to discover value of ES in the most efficient case, and the weight distribution is also plotted. Please see Section 4.5 for the detail.

From the results shown in Section 4.5.1, value '4' has been proved to be the most efficient value of ES for the artificial life form under the control of STDP and directional damping. Test of the artificial life form under the control of STDP and directional damping is shown in Section 4.5.2.

The final version of the equation which is used to update the weights at each three movements of the artificial life form is shown below:

$$W_{ij} \leftarrow W_{ij} + \Delta W_{ij} * A * \zeta + ES \quad \text{eq42}$$

Summary of Chapter3

In Section 3.1 of Chapter 3, the thesis introduces the simulation environment of the program which uses Matlab Guide as the tool of the development. The initial design of the artificial life form is then described in Section 3.2. A detailed flow diagram of the whole program structure is also shown in Figure 15 in this section. The implementations of the central spiking network controller and its external environment are illustrated in Section 3.3 and 3.4 respectively. Finally in Section 3.5 the thesis discusses the implementation of STDP and explains the process in detail with the aid of diagrams. During this last section, all the relative equations of directional damping including the final version of the weight updating equations are illustrated. Therefore, Chapter 3 includes all the significant implementation processes of all kinds of algorithms, equations, functions and variables inside this program. The next chapter will show all the results of testing.

4. Testing

This chapter will demonstrate all the experiments conducted in this project including the experiments mentioned in Chapter 3.

4.1 Testing the ability (efficiency) of the artificial life form overcoming the square obstacle under different noise conditions (with no STDP)

Experiment 1

Please refer to Figure 28 and 29 for the image display of the layout window in this experiment. The settings of experiment 1 are shown in Figure 37 below:

Experiment1	Food(unit:mv)		Noise(unit:mv)		No. of iterations per test	Results No. of movements
	Input Source	Field	Mean	Fluctuations		
test1	5	22	5	5	15	251
test2	5	22	5	15	15	55
test3	5	22	5	45	15	61
test4	5	22	5	75	15	92

Figure 37 Setting of experiment 1

In Figure 37, 4 tests have been taken in experiment 1. The fluctuations of the noise inside the spiking neural network have been set to 5,15,45,75 in these 4 tests respectively. The other values including the mean of the noise, the input strength sent from the food are all kept the same.

The program is designed as such when the artificial life form catches the food, its position will reset to where it starts and the whole process repeats again automatically (see the start-stop loop in Figure 12 in Chapter 2). Each test will take 15 repeating times (iterations), which means the artificial life form will have to repeat the process under the same condition 15 times. The number of movements taken for the artificial life form catching the food will be recorded as the sample value at the end of each of these 15 processes (The sample values taken in these 4 tests are stored in matrix 'Iteration_Matrix' , outputted to the Mat file and saved in the folder 'Experiment data/test4.1' in CD). After the test is finished, an average value is calculated from these 15 sample values, and the result will be an approximation to the efficiency of the artificial life form under the corresponding noise conditions. The larger the number of movements taken, the **less** efficient the artificial life form is.

The efficiency of the artificial life form in **experiment 1** is calculated as:

$$Efficiency \leftarrow \frac{1}{Movements} \quad eq43$$

All the experiment data collected in Chapter 4 which are also outputted to the Mat file are stored in the folder 'Experiment data' in CD

The results from these 4 tests are plotted below:

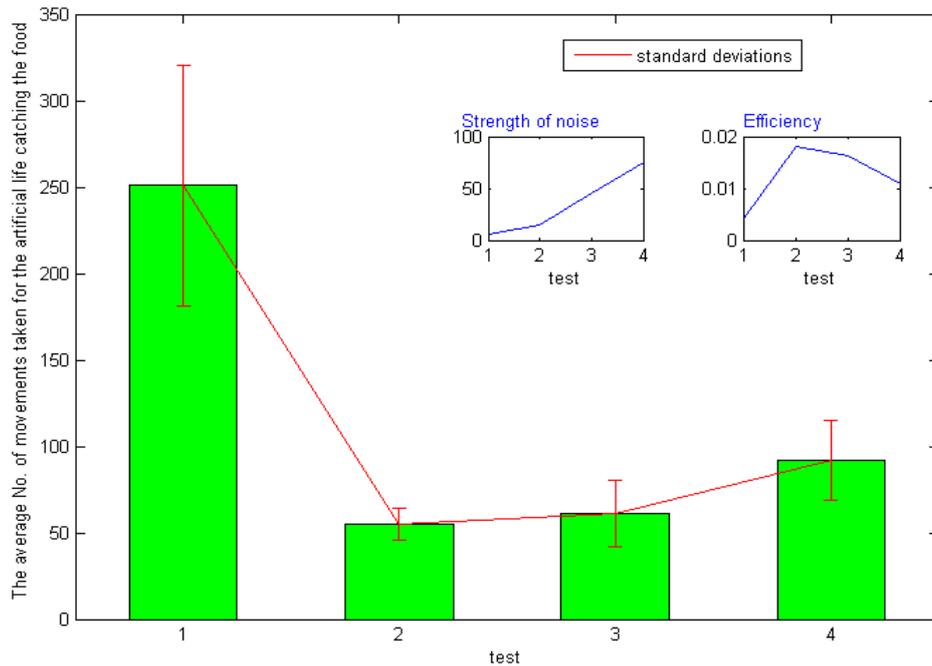


Figure 38

In Figure 38, tests 2, 3, 4 have shown a decreasing tendency of the efficiency of the artificial life form as the fluctuation of the noise increases. That is because as the effect of noise grows inside the artificial life form, the artificial life form moves more randomly and the signal from the noise becomes dominant relative to the signal sent from the food. The artificial life form tends to move towards the other directions rather than moving towards the direction of the food, thus taking longer time to catch the food. In contrast with tests 2,3,4, the level of noise is too low under the conditions of test 1. In that case the signal sent from the food becomes dominant, and the artificial life form loses its degree of randomness and becomes less efficient to move around an obstacle. A compromise between the signal strength of the food and noise is required for the artificial life form reaching its maximum efficiency in this case. In experiment 1, the solution lies around the settings in test 2.

4.2 Testing on the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with no STDP)

Please refer to Figure 30 for the image display of the layout window in this experiment. Another figure showing the image display of one of the samples in test 3 is also shown below.

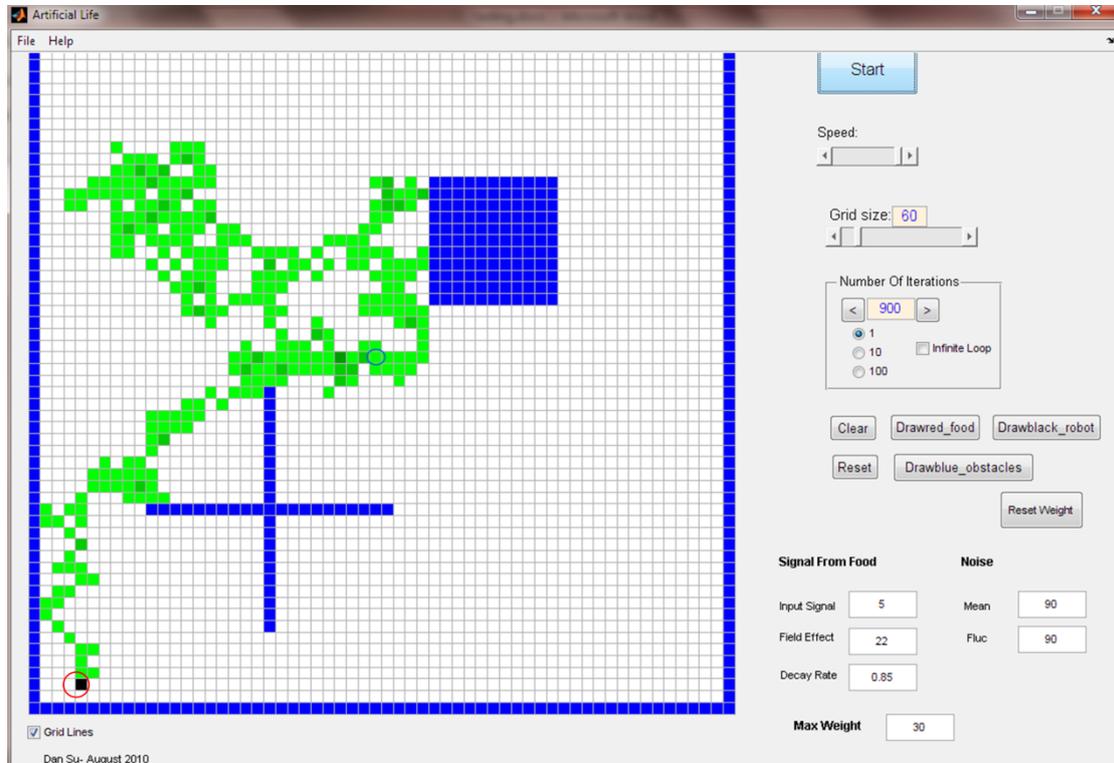


Figure 39 Artificial life form going around the cross obstacle under a large noise level

In Figure 39, the blue marker represents the starting point of the artificial life form while the red marker represents the end point of the artificial life form (where it catches food).

The settings of experiment 2 are shown in Figure 40 below:

Experiment2	Food(unit:mv)		Noise(unit:mv)		No. of iterations per test	Results:movements <600/>600
	Input	Source	Field	Mean		
test1	5	22	50	50	15	3/12
test2	5	22	70	70	15	8/7
test3	5	22	90	90	15	6/9
test4	5	22	110	110	15	0/15

Figure 40

The cross obstacle is much more difficult to overcome than the square obstacle. The artificial life form in this case requires a larger degree of randomness to catch the food compared to experiment 1 in the previous section. Therefore the degree of noise in experiment 2 has been set to very large scales in order to collect sufficient results for the presentation. The program will also record the number of movements the artificial life form has taken to catch the food at the end of each iteration. Value '600' at the top of the last column in this experiment only indicates a reference number of the movements. For later tests in this chapter, if the artificial life form takes less than 600 movements to catch the food, the experiment will treat it as a failure (inefficient) example and plot the efficient examples (Number of iterations in which the artificial life form has taken less than 600 movements to catch the food) into the graph. However, 600 is only a suggested value in this project design due to the time limit, so does the iteration number 15

shown in Figure 40. The results will certainly be more accurate and reliable if more parameters with different numbers are used, which can be considered as future works.

The less the movements the artificial life form takes, the more efficient it is. The results are plotted below in Figure 41. Note that the smaller the value of green bar is, the less efficiency the artificial life forms are.

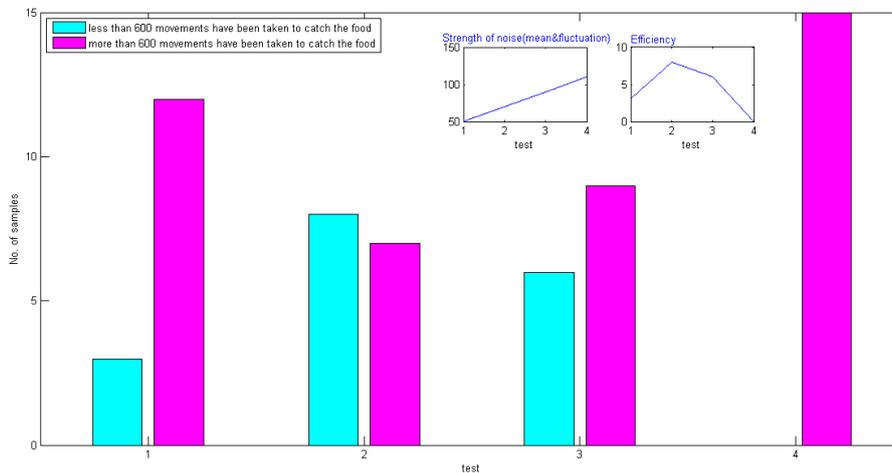


Figure 41

As shown in Figure 41, the efficiency of the artificial life form could be represented by the green bar inside the axes. The plot shows the same phenomenon as in experiment 1 of the previous section. The artificial life form achieves its best performance under certain level of noise (test 2). However, the overall efficiency of the artificial life form in experiment 2 is also in relatively very small scale compared to experiment 1 due to the implementation of larger scale of noise, which means the artificial life form's best performance is greatly limited under such conditions. Therefore STDP has been implemented in order to improve the maximum performance of the artificial life form in this case. The test on STDP will be shown in the next section.

4.3.1 *Testing the ability (efficiency) of the artificial life form overcoming the cross obstacle under different noise conditions (with STDP)

The experiment table is shown below

Experiment3	Food (unit:mv)		Noise (unit:mv)		No. of iterations per test	Efficiency (NO. of iterations that takes <600 movements) STDP/with out STDP
	Input Source	Field	Mean	Fluctuations		
test1	5	22	10	10	15	4/0
test2	5	22	30	30	15	0/0
test3	5	22	50	50	15	4/3
test4	5	22	70	70	15	8/8
test5	5	22	90	90	15	8/6
test6	5	22	110	110	15	3/0

Figure 42

In order to see whether STDP is taking effect, a comparison must be made between the efficiency of the artificial life form under the control of STDP and the efficiency of the artificial life form without STDP. All the weights in the network have been set to '1' initially and allow the STDP to develop the weights according to the external environment of the artificial life form. The result is shown in the last column of Figure 42 and plotted below.

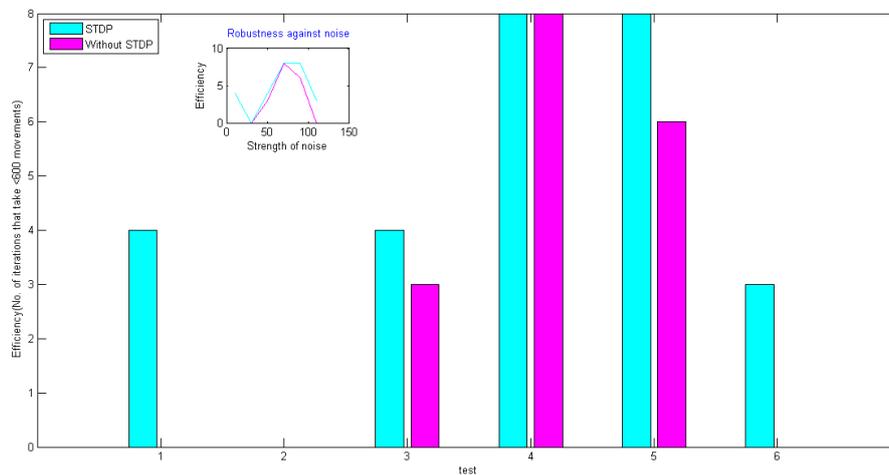


Figure 43

From Figure 43, although the efficiency is still not high, it can be seen that the STDP is actually working, especially under very low level of noise (test 1). When the mean of noise has been increased to 30(test 2), the effect of STDP also starts to decrease since the spike timing has been disrupted. In the meantime, the efficiency of the artificial life form has been reduced to the lowest value because the value of noise is also not at its optimum value to overcome the cross obstacles (the optimum value actually lies around 70 according to both experiments 2 and 3). However, from the small diagram in the figure, it can also be observed that STDP dose show some robustness against high level of noise although the difference between the curves is not clear.

4.3.1 *Statistical test

In Section 4.3.1, the thesis compares the ability of the artificial life form under control of STDP with the artificial life form under no STDP control (fixed weight value). However, the result is not very reliable since there is noise present in later tests and the sample size is not sufficiently large to see the difference between the two conditions (configurations) due to the time limit of the project. Therefore a statistical test has been performed on a particular noise condition (test 5 in Experiment3) in order to test if STDP results in an improvement in the performance of the network. 5 samples have been taken for each condition. The statistical test for the other tests will be added as future works to further justify the reliability of the data.

The experiment table is shown below in Figure 42.1

Experiment3.1 test5	Food(unit:mv)		Noise(unit:mv)		No. of iterations per test	Efficiency(NO. of iterations that takes <600 movements) STDP/with out STDP
	Input Source	Field	Mean	Fluctuations		
Sample1	5	22	90	90	15	6/1
Sample2	5	22	90	90	15	6/4
Sample3	5	22	90	90	15	9/4
Sample4	5	22	90	90	15	7/3
Sample5	5	22	90	90	15	6/5

Figure 42.1

Matlab provides statistics toolbox which contains functions of various statistical tests. The thesis uses the t-test as the statistical test in this experiment. The function is “**ttest2**”. A null hypothesis is established before performing this statistical test on these two sets of data (see the last column of Figure42.1). The null hypothesis is “The using of STDP does not have an effect on the controlling of artificial life form under conditions in test5 of Experiment3.” The function will return a value (h) which will equal to one if the means of this two sets of data are not equal, otherwise its value will equal to zero. In the case h equals one, the means of these two sets of data are not equal and the null hypothesis can be rejected at 5% significance level at current stage. Such result will illustrate that under test 5 conditions, there is a significant difference between the efficiency of the artificial life form under control of STDP and the efficiency of the artificial life form under no STDP control.

After performing the t-test on these two sets of data, the result of value h derived in this experiment is 1. This result suggests that the null hypothesis can be rejected at current stage because the mean of these two sets of data are not equal. That supports the argument “the using of STDP does have an effect on the controlling of artificial life form under conditions in test 5 of Experiment3”. Using different ttest2 functions, the thesis also produced a small p value- 0.0052, which is much below a typical threshold p value to prove the null hypothesis is invalid. The typical p value equals 0.05. The confidence limit is also well removed from the value zero (NULL hypothesis of equal means). The confidence interval in this case is 1.3374 and 5.4626.

By observing the distribution of weights within the spiking neural network in success examples shown in Figure 43, analysis on the relationships between a few single synaptic connections and the behaviors of artificial life form under control of STDP is presented in the next section.

4.4 Analysis of the weight distribution

The display windows of two typical success examples in test 1 of experiment 3 are shown below:

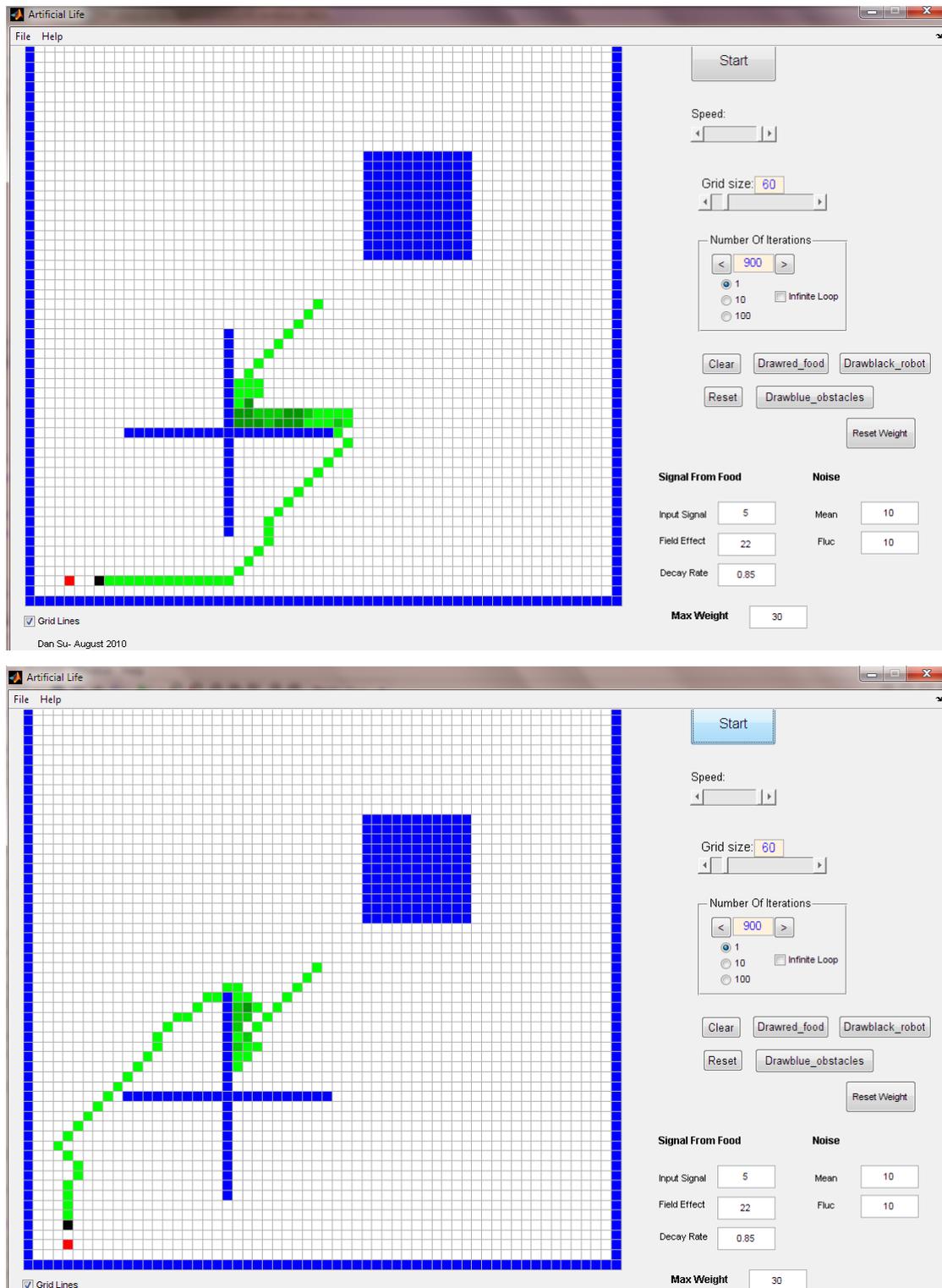


Figure 44 & Figure 45 Success examples of artificial life form overcoming the cross obstacles under the control of STDP (all the weights have been set to '1' initially)

The weight distribution in success and failure (inefficient) examples of experiment 3 is similar in this case. A plot of one of the examples in experiment 3 is shown below.

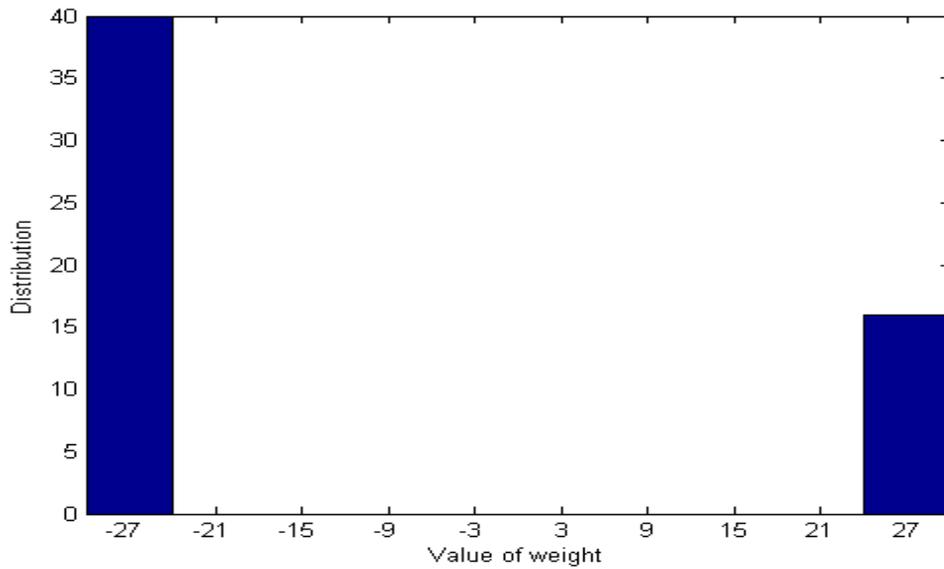


Figure 46 Weight distribution

In Figure 46, it can be seen that it is a bimodal distribution and most of the weight lies within the maximum and minimum boundaries which is -30 and 30 in this case. This will reduce the effect of STDP since the network quickly becomes saturated before STDP is getting more time to adjust the weights. The weights which have increased to 30 might not be the desired synaptic connections that need to be potentiated in the weight configuration of success examples. Undesired potentiation could make the artificial life form move towards undesired directions. Some typical failure examples (Number of movements > 600) are shown below.

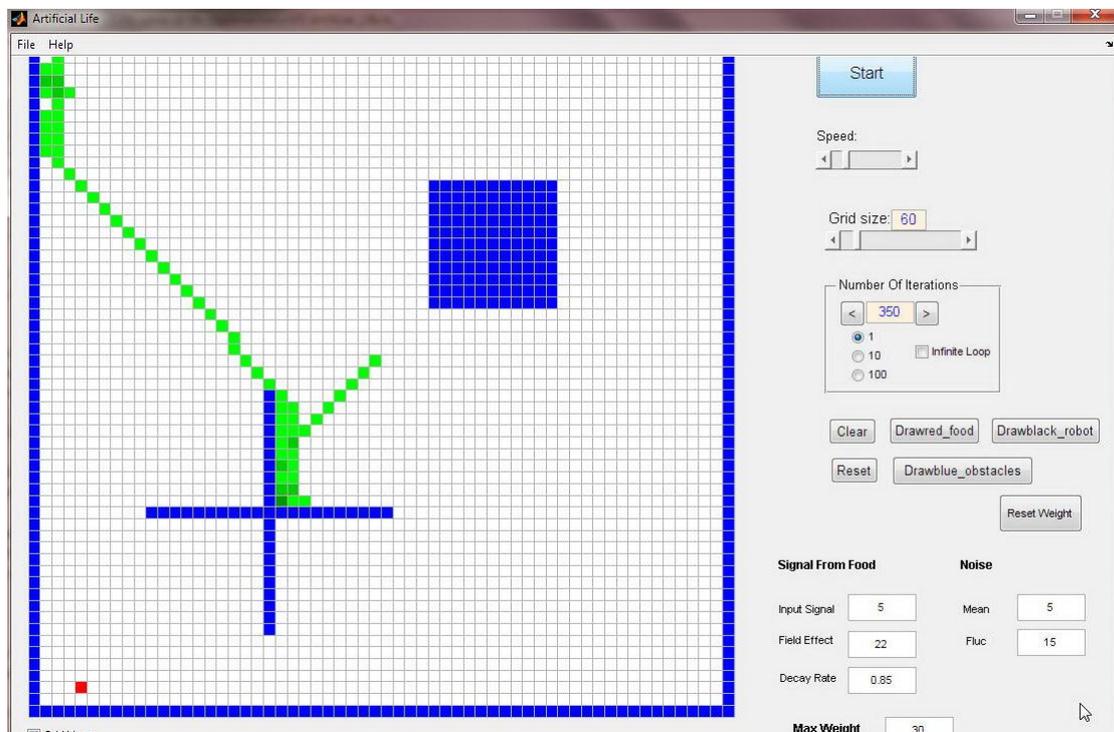


Figure 47

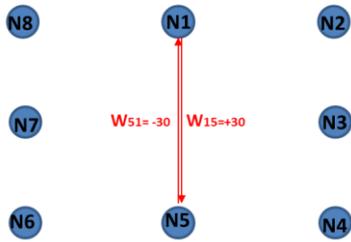


Figure 50

Testing on these two configurations on the behavior of artificial life form has been conducted in experiment 4. The results and the initial configurations of weights are illustrated below.

Test1

Neuron No.	1	2	3	4	5	6	7	8
1	×	1	1	1	1	1	1	1
2	1	×	1	1	1	1	1	1
3	1	1	×	1	1	1	30	1
4	1	1	1	×	1	1	1	1
5	1	1	1	1	×	1	1	1
6	1	1	1	1	1	×	1	1
7	1	1	-30	1	1	1	×	1
8	1	1	1	1	1	1	1	×

Test2

Neuron No.	1	2	3	4	5	6	7	8
1	×	1	1	1	30	1	1	1
2	1	×	1	1	1	1	1	1
3	1	1	×	1	1	1	1	1
4	1	1	1	×	1	1	1	1
5	-30	1	1	1	×	1	1	1
6	1	1	1	1	1	×	1	1
7	1	1	1	1	1	1	×	1
8	1	1	1	1	1	1	1	×

Experiment4	Food(unit:mv)		Noise(unit:mv)		No. of iterations	Efficiency(No. of iterations that takes <600 movements)
	Input Source	Field	Mean	Fluctuations		
test1(set W_{37} & W_{73} initially)	5	22	10	10	15	15
test2(set W_{15} & W_{51} initially)	5	22	10	10	15	15
test3(No initial plasticity all weights are fixed at 1)	5	22	10	10	15	0

Figure 51

In Figure 51, the test results indicate that the efficiency of artificial life form has been greatly increased compared to the case in which no plasticity has been given to the initial configuration of the weights within the spiking neural network.

The display windows of examples in test 1 and test 2 are shown below.

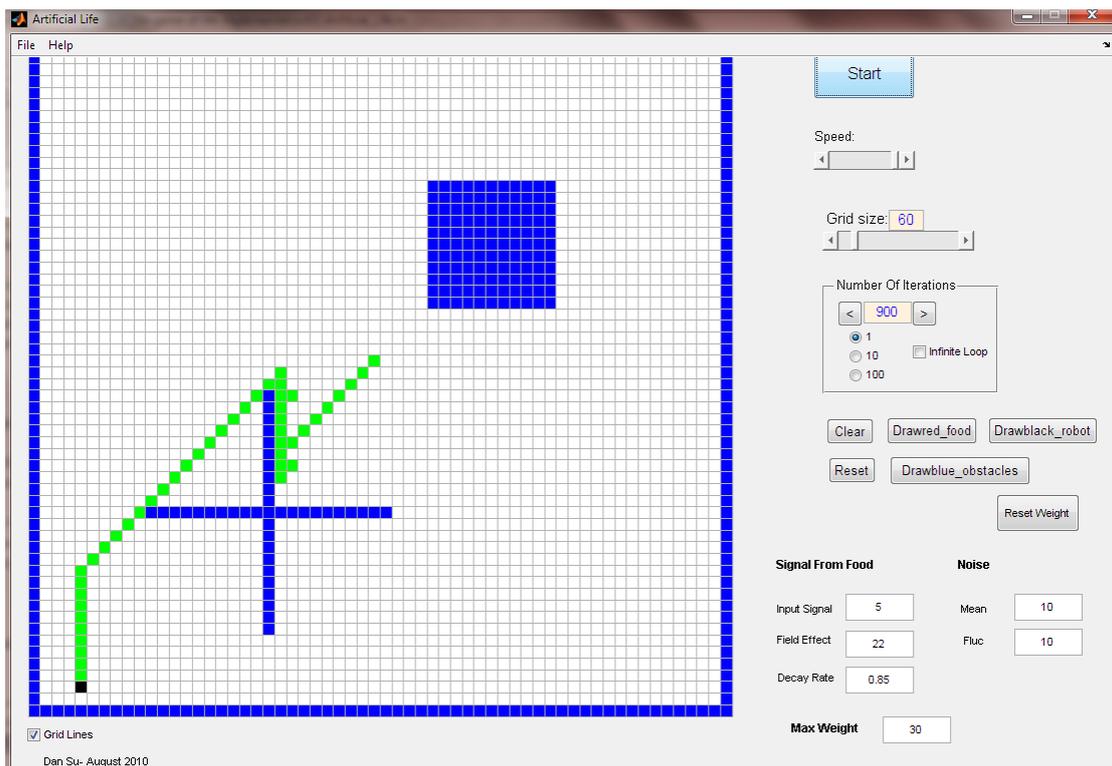
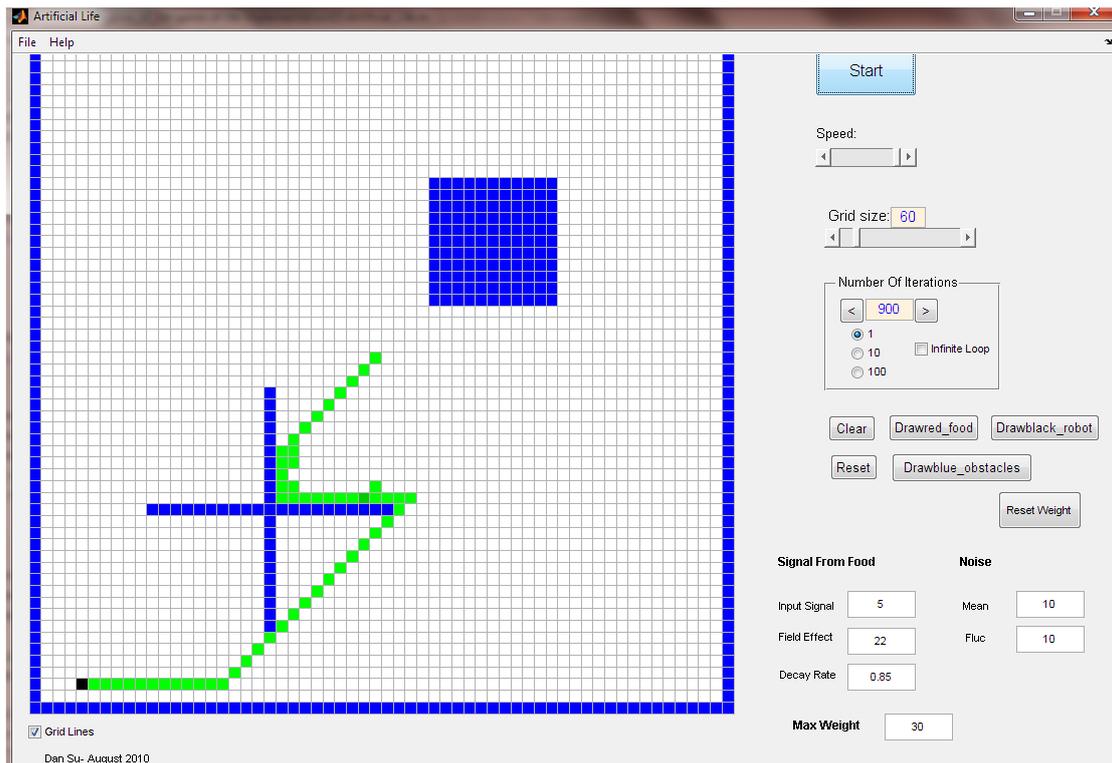


Figure 52 & 53 Display window of the examples in test 1 and test of experiment4

Notice that the green trajectories of the artificial life form in Figure 52 and 53 shows less iterations when moving around obstacles compared to the trajectories of the artificial life form in Figure 44 and 45. That demonstrates that the STDP is actually working in the cases shown in Figure 44 and 45. The extra iterations the artificial life form have spent in those cases represent the process of development of corresponding synaptic weights (W_{37} , W_{73} , W_{15} , W_{51}) within the

spiking network.

It is certain that there are other synaptic connections which are responsible for a particular behavior of the artificial life form. More discussions on the relationships between synaptic connections and the behavior of artificial life form will be made in Chapter 5. The project's focus is on the ability of STDP in developing the weights from initially fixed values (no plasticity) to the desired configurations that are required to complete certain tasks as shown in the experiments above. Therefore, the project is going to examine the way that could further increase the ability of STDP in controlling the behavior of artificial life form by adding extra components to it--such as the directional damping. Testing on the behavior of artificial life form under control of STDP with directional damping is illustrated in the next section.

4.5 *Testing the ability (efficiency) of the artificial life form in overcoming the cross obstacle under different noise conditions (with STDP and Directional Damping)

4.5.1 Analysis of the equilibrium of the weight distribution with Directional Damping

Refer back to the end of Chapter 3, the direction damping has been implemented to the network using eq42. Now an experiment has been conducted with different values of 'ES (The rate of shifting towards the positive equilibrium in weight distribution)' to discover the case in which the best efficiency of the artificial life form under the control of direction damping is produced.

Experiment5	Food (unit:mv)		Noise (unit:mv)		No. of iterations per test	ES	Efficiency (No. of movements<600)
	Input Source	Field	Mean	Fluctuations			
test1	5	22	10	15	15	6	10
test2	5	22	10	15	15	4.5	12
test3	5	22	10	15	15	3	9
test4	5	22	10	15	15	1.5	2

Figure 54

In Figure 54, all the other parameters including the level of noise have been kept the same throughout the program in order to see the effect on the value of ES.

The result is plotted below.

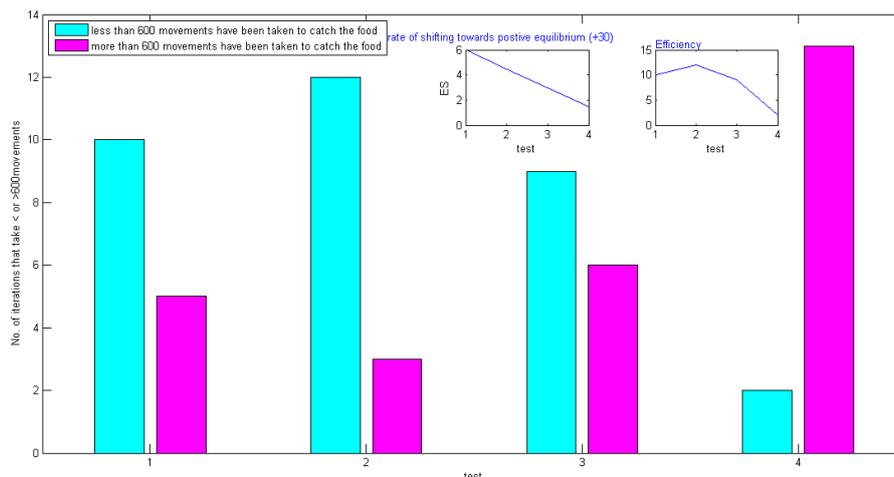


Figure 55 Tests on the effect of ES on the efficiency of the artificial life form

Figure 55 shows that the artificial life form behaves best when the value of ES is 4.5. Figure 56 below shows the plot of the weight distribution of the network in the success examples of test 2.

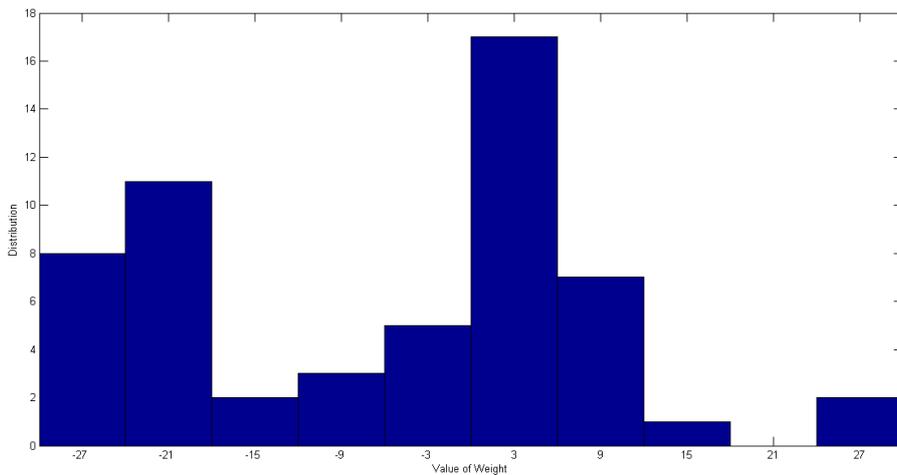


Figure 56

From Figure 56, it can be seen that the distribution tends to be unimodal as the network develops under the control of STDP and directional damping. Now with this ES value implemented, the direction damping will operate at a relatively good state to help STDP control the spiking networks. A display window of a success example under the control of STDP and directional damping is shown below:

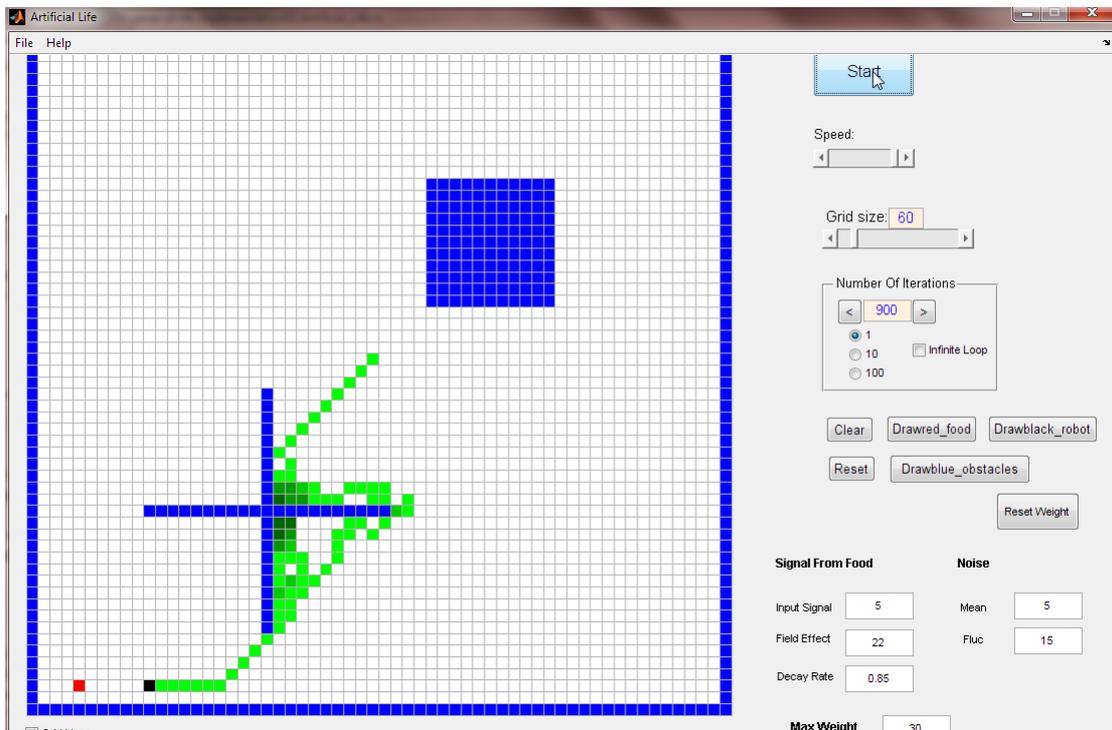


Figure 57 Artificial life form under control of both STDP and Directional Damping

Testing on the behavior of artificial under the control of STDP and directional damping under different noise levels will be discussed in the next section.

4.5.2 Testing the ability (efficiency) of the artificial life form in overcoming the cross obstacle under different noise conditions (with STDP and Directional Damping)

The experiment table is shown below:

Experiment6	Food (unit:mv)		Noise (unit:mv)		No. of iterations per test	Efficiency (NO. of iterations that takes <600 movements) with STDP-DD/STDP/with out STDP
	Input Source	Field	Mean	Fluctuations		
test1	5	22	10	10	15	15/4/0
test2	5	22	30	30	15	0/0/0
test3	5	22	50	50	15	6/4/3
test4	5	22	70	70	15	6/8/8
test5	5	22	90	90	15	7/8/6
test6	5	22	110	110	15	1/3/0

Figure 58

All the experimental settings in Figure 58 are the same as the settings in Figure 42 of experiment 3. All the weights have been initially set to '1' to allow the network to develop them gradually according to the external environment. This time the behavior of artificial life form under the control of both STDP and directional damping has been added into the comparison, and the result is plotted below:

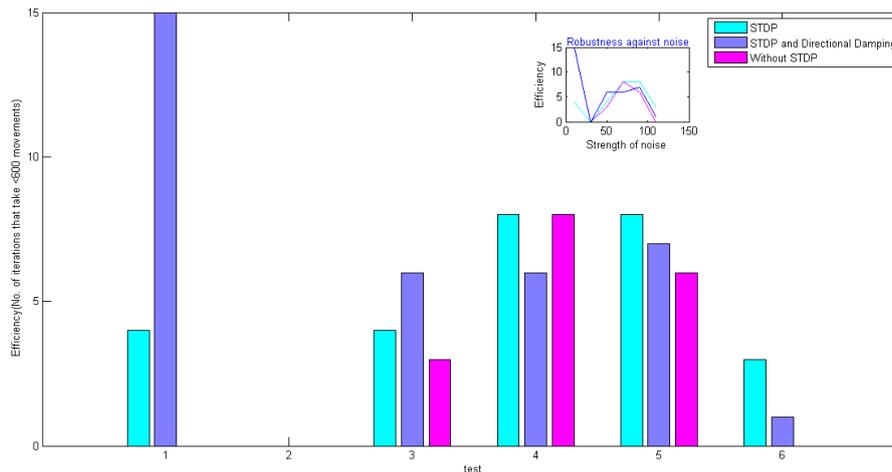


Figure 59

From Figure 59 it can be seen that the artificial life form under control of both STDP and directional damping has shown very efficient behavior under low level of noise. The network now gets more time to develop the weights according to its external environment. However, there is no evidence showing that STDP and directional damping together get good ability of robustness against high level of noise.

To further develop the performance of the artificial life form, another method ADS (Activity Dependent Scaling [14]) could also be implemented as additional components to the network. However, due to the time limit of the project, the research will end at this point and such implementations will be added as future works discussed in Chapter 6.

5. Discussions

Experiment 1 and 2 in Chapter 4 both investigate the ability of artificial life form in overcoming various kinds of obstacles under the different levels of noise. The results indicate that in both cases, there is an optimum level of noise for the artificial life form to go around obstacles and catch food. The reason for this kind of behavior is that, if the level of noise is too low, the randomness of the artificial life form will be greatly limited, resulting in huge amounts of movement to be taken for the artificial life form to go around obstacles. In contrast, if the level of noise is too high, the target attracting effect which depends on the strength of signals sent from food will be greatly reduced due to the vast random movements of the artificial life form, and the artificial life form tends to move towards the undesired directions which drive it away from the food in that situation. The key difference between these two experiments is that, compared to the square obstacles in experiment 1, the cross obstacle in experiment 2 requires a larger level of noise to allow the artificial life form to reach its optimum efficiency. The level of noise is correlated with the ability of the artificial life form in conquering larger and complex obstacles in this case. The success examples are also due to the chance effect of the noise.

However, this is not just a matter of trial and error. There is some kind of systematic way to find relative optimal noise. More statistical tests such as the one shown in section 4.3.1 could be performed to seek the most reliable result. Furthermore, additional functions such as experience learning could be added to the artificial life form in future works. Relative methods and implementation can be found in the discussion of the research paper "A Model of Spatial Map Formation in the Hippocampus of the Rat" written by K. I. Blum and L. F. Abbott: "We will first show how the location represented by place cell activity shifts in a direction that reflects the past experience of the animal in the environment and then discuss how this shifted location can be compared with the present location of the animal to provide a navigational cue." [18]

The project then tries to add some 'intelligence' to the artificial life form so that it can catch the food more efficiently without the help of a high level of noise. The 'intelligence' of the artificial life form is contained in the synaptic connections and weights within the network. In order to overcome cross obstacles efficiently, the network should find a way to encode the information of the external environment into appropriate relationships between each movement direction of the artificial life form. Those relationships are represented using the values of synaptic weights between each synaptic connection within the network. Once the relationships are established successfully using the synaptic weights, which means the network reaches its optimum configuration, the artificial life form should be able to catch the food quickly even without a high level of noise under that situation.

STDP is a learning rule that uses the precise timing between pre and post spikes to encode the information and adjusts the synaptic weights within the network accordingly. The depression has been set stronger than potentiation so that the effect of obstacles could be strengthened. The obstacle itself contains 'zero' signals and will inhibit the activity of a neuron that points to its direction before the artificial life form is about to move onto it. A neuron which is always inhibited will lose its correlations with the other neurons gradually. Therefore relationships showing the different firing possibilities of each neuron (direction) can be established and stored

in the values of weights.

Experiment 3 in Chapter 4 investigates the behavior of the artificial life form under the control of the STDP. The result indicates that the efficiency for artificial life form to catch food under the control of STDP has increased under very low noise conditions, which proves that STDP is actually working in that situation. By examining the effect of single synaptic connections on the behavior of the artificial life form in experiment 4, STDP is also seen to produce successful weight configurations which have been used to guide the artificial life form according to its external environment. However, the configuration is also not accurate enough to produce better behaviors. The bimodal weight distribution illustrates that the weight of network becomes saturated very quickly. As a result, STDP does not get sufficient time to establish enough accurate relationships between each movement directions according to the external environment. The network only 'knows' a rough difference between the firing possibilities of each movement directions of the artificial life form under that situation. The efficiency of artificial life form is also reduced in that case since undesired synaptic connections could also develop before STDP gets sufficient time to make correct adjustments. Figure 47 and 48 in Section 4.4 exhibit the display window of some failure examples when undesired synaptic weights have been potentiated.

Now the network needs some ways to regulate the changing of weights so that it will not become saturated so quickly. Direction Damping is the method used and implemented into the network in this case. It will slow down the changes that push a weight towards the boundary if the weight is near that boundary, and leave the changes that push the weight away from it unaffected. The closer is the weight from the boundary, the slower is the rate of change allowed for the weight to change towards that boundary.

Experiment 6 compares the behaviors of the artificial life form under the control of STDP, under the control of both STDP and directional damping, and without the control of STDP under different levels of noise. The result shows that the directional damping is working successfully. There is a great increase of efficiency compared to the cases when STDP is working alone. STDP now gets more time to develop the relationships between each movement direction and is able to distinguish them with minor differences of firing possibilities

However, the effect of noise should not be eliminated. The earlier examples also demonstrate the correlations between the level of noise and the ability for artificial life form to overcome complex obstacles. In experiment 6, what would happen if the length of the cross obstacles had been increased? Although the artificial life form is still able to catch the food within a relatively efficient manner (which is not shown in the thesis, the user could try to modify the settings in the program to observe the effect), a higher level of noise would be required for the artificial life form to reach its optimum behavior in that situation. In real biological systems, there are also huge amounts of noise existing in neurons. The question is: could the STDP work under a high level of noise? Even if the timing of the spikes has been disrupted? Comparing the results in experiment 3 and experiment 6, STDP working alone tends to exhibit better robustness against noise than STDP and directional damping working together. Although STDP working alone shows bimodal weight distributions, its fast reflections to the external environment can establish a

rough accurate relationship between different movement directions within a relatively short time when it encounters any unexpected stimuli from outside--such as an obstacle. In contrast, in the case of STDP and directional damping working together, although the synaptic weights can be adjusted to a very accurate degree, the speed is too slow. The artificial life form will be driven far away from the stimuli by a larger level of noise before a successful configuration is reached.

Therefore the future investigations for this project could focus on the attempts to discover or use better techniques of adjusting the synaptic weights so that an appropriate balance could be reached between the rate and accuracy of the ability for STDP to adjust the weights. 'ADS' (Activity Dependent Scaling) is one of the methods that could be implemented to the network in future. Some research also indicates that the 'rate code' in which the plasticity depends on the different firing rate between neurons could be used. This coding scheme will also be added to the network as future works to investigate its effect on improving the performance of the artificial life form.

The thesis only describes one scenario in Chapter 4. However, a single scenario/environment is not sufficient for the artificial life form in coping with other scenarios which have more complex obstacles and conditions (such as multiple targets). To solve this problem, additional neurons, algorithms and tests will be required for further investigations.

To conclude, the thesis first gives a brief introduction of the project aims and program environment in Chapter 1. In Chapter 2, there is a detailed description of all the background theories behind the project. In Section 2.4 of Chapter 2, a number of experiments are conducted to demonstrate various spiking behaviors of neurons under the simulation of the Izhikevich model. The whole implementation process including the implementation of central spiking network controller and external environment are described in detail in Chapter 3. The algorithms and equations for STDP and directional damping are also illustrated at the end of Chapter 4. In Chapter 4, all the data collected are gathered and used for testing. 6 experiments are conducted to investigate the behaviors of artificial life form under different situations. There are also plots on the weight distributions depicted in Section 4.4 and 4.51 of Chapter 4. An analysis of results is also made in Chapter 4. A more detailed discussion, however, is presented in Chapter 5. Finally, all the future works including those mentioned in previous chapters will be estimated in the next chapter.

6. Future Works

As mentioned at the beginning of Section 2.2 of Chapter 2, Neural Robotics has two main research applications in academic world. The first is to create autonomous artifacts or intelligent systems which inherit some or part of the brain's functions in order to help people solve real-world problems. The second is to construct complex neural models and architectures to investigate the basic principles of the brain and its relationships with behaviors. The investigation of the artificial life form in this project is intended to be in the direction of the first research application with experiments trying to discover a way to increase the efficiency of the artificial life form and its ability to evolve under the control of STDP in various situations. However, Section 2.4 of Chapter 2 proves that the properties of Izhikevich model also allow the neuron to produce rich biological plausible spiking patterns, which leaves more research space especially in the direction of the second research application.

During the implementation and testing, many parameters have been simulated a number of times and selected to fit the program use. There are also the cases that the parameters are chosen for simplicity of implementation. One example is the implementation of the uniform random distribution for noise which could be replaced by more biological plausible normal random distributions. The implementation of external environment also has room for improvement. The program could add fields to the obstacles so that a repulsion force could be sent from them just like the attraction force is sent from the food. The artificial life form could exhibit better behaviors under that situation.

The implementation of STDP is always followed by the implementation of direction damping and ADS. However, due to the time limit of the project, ADS has not been implemented to the program. It is believed that the implementation of ADS could have a very positive effect on the behavior of artificial life form under the control of STDP and directional damping. Experiments in the thesis are attempted to test the ability of the artificial life form to evolve itself under the control of the STDP. In future, more properties inspired from living systems could also be added to the artificial life form.

More statistical tests similar to the one described in Section 4.3.1 can be performed on the experiment data in order to increase the reliability.

As mentioned in Chapter 2 and the discussion section of Chapter 5, the project only uses one food in one scenario for experiment testing on STDP. That is far from drawing any fixed conclusions for the effect of STDP on the behaviors of the artificial life form. Additional problems will arise, however, if more food (target) is added to the maze, since the relatively strong attractive force of the target from the surrounding directions could 'confuse' the artificial life form. The project should therefore make more modulations between the input strength of targets, obstacles and noises. Better external environmental rules may also be adopted in future.

The artificial life form in the single scenario is attracted towards a target behind the cross obstacles. STDP has been added to the spiking neural network controller to strengthen the correlations between the direction neurons, so that the "external spatial information" could be

passed to the artificial life. However, more complete and reliable spatial information will be required if the artificial life is tasked in multiple scenarios. That requires more studies and tests on the behavior of the artificial life form including further modulation of different parameters such as the time delay.

One interesting example is the ability for the artificial life to 'remember'. After the artificial life form has gone around any obstacle once, could it remember the successful weight configurations and would response more quickly when it encounters the same situation again in future? Suitable temporal information can also be established before the artificial life makes any decisions.

The general approach in this project is relevant to bio-inspired engineering design, which is a focus of the group in the University of York. It also studies various behavior of artificial life under the control of a spiking neural network controller with and without STDP. The data collected in the testing section will be useful for further investigations on STDP or any other related experiments.

Finally, although there are many research directions to which this project can move in future, the program designed in this project can also be used as a research tool for many problems ranging from biological to engineering perspectives. The display window is well designed and number of parameters could be altered by users themselves. The program can be used as a software demonstration of spiking neural network applications. The program written in Matlab code is well structured and explained, which will be attached in Appendix.

7. Acknowledgements

The author would like to thank his supervisor Dr. David Halliday. He has provided very useful suggestions and comments to the thesis on each stage of the project with great patience. His enthusiasm on the subject has also given the author great momentum to complete the thesis, especially when difficulties were encountered. He has also corrected the spelling and grammatical mistakes of the thesis. The thesis would be far less understandable without the valuable guidance of Dr. David Halliday, many thanks.

The author would also like to thank his parents for their financial support on the degree and their encouragements to the author throughout this project, many thanks.

8. Appendix

Appendix1: Resonator

```
Ne=800; Ni=200;
re=rand(Ne,1); ri=rand(Ni,1);
a=[0.02*ones(Ne,1); 0.02+0.08*ri];
b=[0.2*ones(Ne,1); 0.25-0.05*ri];
c=[-65+15*re.^2; -65*ones(Ni,1)];
d=[8-6*re.^2; 2*ones(Ni,1)];
S=[0.5*rand(Ne+Ni,Ne), -rand(Ne+Ni,Ni)];
v=-65*ones(Ne+Ni,1); % Initial values of v
u=b.*v; % Initial values of u
firings=[]; % spike timings

X=zeros(Ne+Ni,1000);
Y=[];
X1=zeros(Ne+Ni,1000);
X2=zeros(Ne+Ni,1000);

NET=200;      NET2=300;

a(NET,1)=0.02;   a(NET2,1)=0.02;
b(NET,1)=0.2;   b(NET2,1)=0.25;
c(NET,1)=-45;   c(NET2,1)=-65;
d(NET,1)=2;     d(NET2,1)=8;

NOOS=0; %Set the Initial value of No of spikes
NOOS2=0;

for t=1:1000 % simulation of 1000 ms
I=0;

if t>200
%I=[5*randn(Ne,1);5*randn(Ni,1)];% thalamic input
I=[8*ones(Ne,1);5*ones(Ni,1)]; %Normal dc input
end

fired=find(v>=30); % indices of spikes
firings=[firings; t+0*fired,fired];
v(fired)=c(fired);
u(fired)=u(fired)+d(fired);
%I=I+sum(S(:,fired),2);
v=v+0.5*(0.04*v.^2+5*v+140-u+I); % step 0.5 ms
```

```

v=v+0.5*(0.04*v.^2+5*v+140-u+I); % for numerical
u=u+a.*(b.*v-u); % stability

if v(NET,1)>=30 % count the No. of spikes
    NOOS=NOOS+1;
end

if v(NET2,1)>=30 % count the No. of spikes
    NOOS2=NOOS2+1;
end

X(:,t)=v;
Y=[Y;t];
X1(:,t)=u;
X2(:,t)=I;

end;

NOOS %Number of spikes
NOOS2

%plot(firings(:,1),firings(:,2),'.');
subplot(20,1,1:13);
plot(Y,X(NET,:), 'k');
%hold on
%plot(Y,X(NET2,:), 'b:');
axis([0 1000 -100 80]);
%legend('a=0.1 b=0.2', 'a=0.02 b=0.25', 'Location', 'Northwest');
ylabel('v(t)/mV');
%hold off

subplot(20,1,14:18);
plot(Y,X1(NET,:), 'k');
%hold on
%plot(Y,X1(NET2,:), 'b:');
axis([0 1000 -18 18]);
ylabel('u(t)/mV');
%hold off

```

```
subplot(20,1,19:20);  
plot(Y,X2(NET,:), 'r');  
axis([0 1000 -5 10]);  
xlabel('time/ms');  
ylabel('I(t)')
```

Appendix2 the final designed version of artificial life form-STDP&DD have been turned on, all the other settings are as shown in the display window.

```
% Dan Su
function varargout = Artificial_Life(varargin)
% Aritificial_Life M-file for Aritificial_Life.fig

% Last Modified by GUIDE v2.5 27-Oct-2010 10:36:24

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @Artificial_Life_OpeningFcn, ...
    'gui_OutputFcn',  @Artificial_Life_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Aritificial_Life is made visible.
function Artificial_Life_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Aritificial_Life (see VARARGIN)

% Choose default command line output for Aritificial_Life
set(gcf,'name','Artificial life form');
handles.output = hObject;
handles.iter_num=900;
handles.iter_step=1;
```

```

handles.N=60;
handles.vid=0;
handles.vis_grid=zeros(60);
handles.numberpass_grid=zeros(65);

%experiment handles-record the number of Iterations
handles.Iteration_Matrix=[];
handles.Excounter=0;

%the parameters of input signal scattered from food
handles.input_source=5;
handles.field_effect=22;
handles.decay_rate=0.85;

%Noise
handles.mean_value=5;
handles.fluc_value=15;

%Drawing handles
handles.vis_grid(30,30)=artificial_life;
handles.vis_grid(42,22)=artificial_life;
handles.vis_grid(15:25,35:45)=obstacle1; %square obstacles
handles.vis_grid(43,11:31)=obstacle1; %cross obstacles
handles.vis_grid(33:53,21)=obstacle1;
handles.vis_grid(1:60,1)=obstacle1; %outter most wall
handles.vis_grid(1:60,60)=obstacle1;
handles.vis_grid(1,1:60)=obstacle1;
handles.vis_grid(60,1:60)=obstacle1;

handles.vis_grid(43,11:41)=obstacle1; %an extended version of the
cross
%obstacles
handles.vis_grid(23:53,21)=obstacle1;

%Initial food location
handles.vis_grid(58,5)=food1; %down left
handles.vis_grid(8,52)=food1; %upper right
handles.vis_grid(30,24)=food1;

```

```

%handles.vis_grid(33,16)=food1;
%handles.vis_grid(22,35)=food1;

%Spiking network model parameters including STDP,weight.....
handles.reset_grid=handles.vis_grid;
handles.calc_weight=[1*ones(4,8);1*ones(4,8)];
    for ww=1:8      %setting selfconnection to zero
        handles.calc_weight(ww,ww)=0;
    end
%handles.calc_weight(7,3)=-30;
%handles.calc_weight(3,7)=30;
%handles.calc_weight(5,1)=-30;
%handles.calc_weight(1,5)=30;

for qq=1:8
handles.calc_weight(qq,qq)=0;
end
handles.sd=zeros(8,8);
handles.time_window=400;
handles.delay=20;
handles.firings=[-handles.delay,0];
handles.stdp=zeros(8,handles.time_window+handles.delay+1);
handles.max_weight_value=30;

handles.singleweight_sc=zeros(8,2);

% Update handles structure
guidata(hObject, handles);
plot_grid(handles);
% UIWAIT makes Aritificial_Life wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Artificial_Life_OutputFcn(hObject, eventdata,
handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

```

```

% Get default command line output from handles structure
varargout{1} = handles.output;
%
-----
function file_Callback(hObject, eventdata, handles)
% hObject    handle to file (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%
-----

function load_mat_Callback(hObject, eventdata, handles)
% hObject    handle to load_mat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName] = uigetfile('*.mat','Load');
if strcmpi('mat',FileName(end-2:end))
    try
        a=importdata([PathName,FileName]);
        size_a=size(a);
        if (length(size_a)~=2) | (size_a(1)~=size_a(2))
            errordlg('The matrix is not of the proper
dimensions.', 'Error');
        elseif size_a(1)<5
            errordlg('The matrix is too small.', 'Error');
        elseif size_a(1)>500
            errordlg('The matrix is too big.', 'Error');
        elseif a~=logical(a)
            errordlg('Not all values in the matrix are '0' or '1'.
', 'Error');
        else
            handles.vis_grid=a;
            set(handles.N_size, 'value', size_a(1));
            set(handles.grid_size_N, 'string', num2str(size_a(1)));
            handles.N=size_a(1);
            guidata(hObject, handles);
            plot_grid(handles);
        end
    catch
        errordlg('The file cannot be loaded', 'Error');
    end % try-catch
else % if f_type
    errordlg('The program can only load MATLAB mat-files.', 'Error');
end %if f_type

```

```

%
-----
function save_mat_Callback(hObject, eventdata, handles)
% hObject    handle to save_mat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName] = uiputfile('*.mat','Save');
a=handles.vis_grid;
save([PathName,FileName], 'a') ;
%
-----

function save_pic_Callback(hObject, eventdata, handles)
% hObject    handle to save_pic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
types={'*.jpg'; '*.tif'; '*.bmp'; '*.png'};
[FileName,PathName,file_type] = uiputfile(types,'Save As Image');
frm=getframe(handles.axes1);
[im,map] = frame2im(frm);
imwrite(im,[PathName,FileName],types{file_type}(3:5));
%
-----

function save_video_Callback(hObject, eventdata, handles)
% hObject    handle to save_video (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName] = uiputfile('*.avi','Save As Video');
handles.vid_name=[PathName,FileName];
handles.vid=1;
guidata(hObject, handles);
%
-----

function close_fig_Callback(hObject, eventdata, handles)
% hObject    handle to close_fig (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close;
%
-----

function help_menu_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%

```

```

-----
function user_manual_Callback(hObject, eventdata, handles)
% hObject    handle to user_manual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%open('\Artificial_Life_manual.htm')
open('\Artificial_Life.txt')
%
-----

function help_about_Callback(hObject, eventdata, handles)
% hObject    handle to help_about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
about={'Artificial life form:';...

    'This program written using MATLAB version R2009a.';...
    '';...
    'Dan Su August 2010 (c)';...
    };

about_msg=msgbox('','About');
set(about_msg,'position',[345, 417, 330,150]);
ah = get( about_msg, 'CurrentAxes' );
ch = get( ah, 'Children' );
set(ch,'string',about)

% --- Executes on slider movement.
function N_size_Callback(hObject, eventdata, handles)
% hObject    handle to N_size (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=round(get(hObject,'Value') );
if n>handles.N
    handles.vis_grid(n,n)=0;
elseif n<handles.N
    handles.vis_grid(n+1:end,:)=[];
    handles.vis_grid(:,n+1:end)=[];
end
handles.N= n;
set(handles.grid_size_N,'string',num2str(handles.N));
guidata(hObject, handles);
plot_grid(handles);
% Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine range of

```

```

slider

% --- Executes during object creation, after setting all properties.
function N_size_CreateFcn(hObject, eventdata, handles)
% hObject    handle to N_size (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in grid_lines.
function grid_lines_Callback(hObject, eventdata, handles)
% hObject    handle to grid_lines (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of grid_lines
if get(hObject,'Value')==1
    axis on
else
    axis off
end
guidata(hObject, handles);

% --- Executes on button press in decrease_iterations.
function decrease_iterations_Callback(hObject, eventdata, handles)
% hObject    handle to decrease_iterations (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.iter_num=handles.iter_num - handles.iter_step;
if handles.iter_num<1
    handles.iter_num=1;
end
set(handles.iterations_str,'string',num2str(handles.iter_num));
guidata(hObject, handles);

% --- Executes on button press in increase_iterations.
function increase_iterations_Callback(hObject, eventdata, handles)
% hObject    handle to increase_iterations (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles    structure with handles and user data (see GUIDATA)
handles.iter_num=handles.iter_num + handles.iter_step;
set(handles.iterations_str,'string',num2str(handles.iter_num));
guidata(hObject, handles);

% --- Executes on button press in inf_loop.
function inf_loop_Callback(hObject, eventdata, handles)
% hObject    handle to inf_loop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of inf_loop
switch get(hObject,'Value')
    case 1
        handles.iter_num=inf;
        button_enable(handles,'off','INF');
    case 0
        button_enable(handles,'on','INF');

handles.iter_num=str2num(get(handles.iterations_str,'string'));
end;
guidata(hObject, handles);

% --- Executes when selected object is changed in iterations_num.
function iterations_num_SelectionChangeFcn(hObject, eventdata,
handles)
% hObject    handle to the selected object in iterations_num
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none
was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
switch get(eventdata.NewValue,'Tag')
    case 'units'
        handles.iter_step=1;
    case 'tens'
        handles.iter_step=10;
    case 'hundreds'
        handles.iter_step=100;
end
guidata(hObject, handles);

function grid_size_N_Callback(hObject, eventdata, handles)
% hObject    handle to grid_size_N (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of grid_size_N as text
%        str2double(get(hObject,'String')) returns contents of
grid_size_N as a double
if
(str2num(get(hObject,'string'))>=5) & (str2num(get(hObject,'string'))<=
500)
    n=round(str2num(get(hObject,'string')));
    if n>handles.N
        handles.vis_grid(n,n)=0;
    elseif n<handles.N
        handles.vis_grid(n+1:end,:)=[];
        handles.vis_grid(:,n+1:end)=[];
    end
    handles.N= n;

set(handles.N_size,'value',round(str2num(get(hObject,'string'))));
else
    errordlg('Input must be number between 5 and 500','Wrong Input');
end
set(handles.grid_size_N,'string',num2str(handles.N));
guidata(hObject, handles);
plot_grid(handles)

function iterations_str_Callback(hObject, eventdata, handles)
% hObject handle to iterations_str (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of iterations_str as text
%        str2double(get(hObject,'String')) returns contents of
iterations_str as a double
if (str2num(get(hObject,'string'))>=1)
    handles.iter_num=round(str2num(get(hObject,'string')));
else
    errordlg('Input must be number larger than 1','Wrong Input');
end
set(handles.iterations_str,'string',num2str(handles.iter_num));
guidata(hObject, handles);

% --- Executes on button press in start_stop.
function start_stop_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to start_stop (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of start_stop

vid='';
if get(hObject,'Value')==1
    if handles.vid==1
        handles.vid=0;
        vid=questdlg({'The animation will be saved to a video
file: ';handles.vid_name},'Video','OK','Cancel','OK');
        if strcmp(vid,'OK')
            aviobj =
avifile(handles.vid_name,'compression','Cinepak','fps',get(handles.sp
eed_slider,'value'));
            set(handles.vid_str,'visible','on');
        end
    end
    set(hObject,'string','Stop');
end
handles.calc_grid=zeros(handles.N+4);
handles.calc_grid(3:end-2,3:end-2)=handles.vis_grid;
handles.reset_grid=handles.vis_grid;
button_enable(handles,'off','START_STOP')
i=1;
%previous=handles.vis_grid;
while 1% the use in "while" instead of "for" is because 'handles.iter_num'
may equal infinity
    handles=calc_game(handles);
    handles.vis_grid=handles.calc_grid(3:end-2,3:end-2);
    plot_grid(handles);
    title(['Iteration #',num2str(i)]);
    pause(0.1/get(handles.speed_slider,'value'))    %control the real
time speed of the robot
    if (get(hObject,'Value')==0) || (i>=handles.iter_num)
        weightrecording_temp=handles.calc_weight; %record the weight
value when the stop button is pressed
        save Iteration_numbers04.mat weightrecording_temp %output the
weight value
        singleweight_sc=handles.singleweight_sc;

        save singleweightrecording01.mat singleweight_sc %output the
success sample's singleweight value

```

```

        break
    end
    guidata(hObject, handles);
    if strcmp(vid, 'OK')
        frame = getframe(handles.axes1);
        aviobj = addframe(aviobj, frame);
    end
    i=i+1;
    %H=i;

    %save iteration_numbers01 H

    %Record STDP, fired neuron in the last section of the previous time
    %window and copy them to the start section of the current time window

handles.stdp(:, 1:1+handles.delay)=handles.stdp(:, handles.time_window+
1:handles.time_window+1+handles.delay);
    ind=find(handles.firings(:,1) > handles.time_window-handles.delay);
    handles.firings=[-handles.delay
0;handles.firings(ind,1)-handles.time_window,handles.firings(ind,2)];

    %experiment

    [foodv, foodh]=find(handles.calc_grid==2);    %check if the food
still exist

    check_food=[foodv, foodh];

NITER=600; %maximum No.of Iterations

if isempty(check_food) || i>=NITER    %execuates when there is no food
or the Iteration No.is not too high

Excounter=handles.Excounter;    %number of experiments
Excounter=Excounter+1
handles.Excounter=Excounter;
if Excounter>15
break

end

```

```

[robv,robh]=find(handles.calc_grid==1);      %find the position of the
robot

%reset all the parameters
handles.calc_grid(robv,robh)=0;      %reset robot postion
handles.calc_grid(32,32)=1;

if i<NITER
weightrecording_success=handles.calc_weight;
save Iteration_numbers05.mat weightrecording_success % record the
success weight data before resetting
end

if i==NITER
weightrecording_fail=handles.calc_weight;
save Iteration_numbers06.mat weightrecording_fail
end

handles.calc_weight=[1*ones(4,8);1*ones(4,8)];%reset weight values,
sd, stdp.....
%handles.calc_weight(7,3)=-30;
%handles.calc_weight(3,7)=30;
%handles.calc_weight(5,1)=-30;
%handles.calc_weight(1,5)=30;

handles.sd=zeros(8,8);
handles.stdp=zeros(8,handles.time_window+handles.delay+1);
handles.numberpass_grid=zeros(65);

%handles.calc_grid(10,54)=2; %food resetting %upper-right
handles.calc_grid(60,7)=2; %down left

[trv,trh]=find(handles.calc_grid>=5 &
handles.calc_grid<=9);%trajectory reseeting
trex=length(trv);
for tr=1:trex
handles.calc_grid(trv(tr),trh(tr))=0;
end

```

```

%record the Iteration No.
Iteration_Matrix=handles.Iteration_Matrix;
Iteration_Matrix=[Iteration_Matrix;i];
handles.Iteration_Matrix=Iteration_Matrix;

save Iteration_numbers03.mat Iteration_Matrix
i=0; %reset the iteration number
guidata(hObject,handles);
end

    sm=handles.max_weight_value; ES=4;    %sm is the maximum (and minimum)
weight value, ES is the weight increment at each update
    if i/3-round(i/3)==0 && i~=0    %update the weight each 3 real time
turns

        %damping
        zeta=ones(8,8);
        max_damp=find(handles.calc_weight>0 & handles.sd>0 );

zeta(max_damp)=0.00000005-0.000000049*(handles.calc_weight(max_damp)/
sm);

        min_damp=find(handles.calc_weight<0 & handles.sd<0 );

zeta(min_damp)=0.00000005-0.000000049*(handles.calc_weight(min_damp)/
-sm);

    for uw=1:64
        handles.sd(uw)=handles.sd(uw)*zeta(uw);
    end

    %update the weight between sm and -sm

handles.calc_weight=max(-sm,min(sm,ES+handles.calc_weight+handles.sd)
);

    for ww=1:8    %setting selfconnection to zero
handles.calc_weight(ww,ww)=0;
    end

```

```

%data collecting
nturn=i/3;
singleweight_sc(1,nturn)=handles.calc_weight(3,7);
singleweight_sc(2,nturn)=handles.calc_weight(7,3);
handles.singleweight_sc=singleweight_sc;

weight=handles.calc_weight
i
handles.sd=27*handles.sd;      %control the scale of derivatives
end
guidata(hObject, handles);
end

%Iteration_Matrix=[Iteration_Matrix;H];
%Iteration_Matrix=[]; %clear out the Matrix for the next experiment
%save iteration_numbers02.mat Iteration_Matrix

plot_grid(handles);
title('');
if strcmp(vid,'OK')
    aviobj = close(aviobj);
    set(handles.vid_str,'visible','off');
end
button_enable(handles,'on','START_STOP')
if get(handles.inf_loop,'value')==1
    button_enable(handles,'off','INF');
end
set(hObject,'string','Start','value',0);
guidata(hObject, handles);

% --- Executes on slider movement.
function speed_slider_Callback(hObject, eventdata, handles)
% hObject    handle to speed_slider (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

% --- Executes during object creation, after setting all properties.
function speed_slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to speed_slider (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in draw_red_button.
function draw_red_button_Callback(hObject, eventdata, handles)
% hObject handle to draw_red_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.reset_grid=handles.vis_grid;
set(handles.draw_mode,'visible','on')
button_enable(handles,'off','DRAW');
while 1
    x=1;
    [y,x]=ginput(1);
    if ~isempty(x)
        x=round(x);
        y=round(y);
        if (x>0) && (x<handles.N+1) && (y>0) && (y<handles.N+1)
            %handles.vis_grid(x,y)=~(handles.vis_grid(x,y));
            handles.vis_grid(x,y)=2*~(handles.vis_grid(x,y));
            guidata(hObject, handles);
            plot_grid(handles)
        end
    else
        break
    end
end %while
set(handles.draw_mode,'visible','off')
button_enable(handles,'on','DRAW');
if get(handles.inf_loop,'value')==1
    button_enable(handles,'off','INF');
end
% Hint: get(hObject,'Value') returns toggle state of draw_red_button

function draw_yellow_button_Callback(hObject, eventdata, handles)
% hObject handle to draw_yellow_button (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.reset_grid=handles.vis_grid;
set(handles.draw_mode, 'visible', 'on')
button_enable(handles, 'off', 'DRAW');
while 1
    x=1;
    [y,x]=ginput(1);
    if ~isempty(x)
        x=round(x);
        y=round(y);
        if (x>0) && (x<handles.N+1) && (y>0) && (y<handles.N+1)
            %handles.vis_grid(x,y)=~(handles.vis_grid(x,y));
            handles.vis_grid(x,y)=3*~(handles.vis_grid(x,y));
            guidata(hObject, handles);
            plot_grid(handles)
        end
    else
        break
    end
end %while
set(handles.draw_mode, 'visible', 'off')
button_enable(handles, 'on', 'DRAW');
if get(handles.inf_loop, 'value')==1
    button_enable(handles, 'off', 'INF');
end
% Hint: get(hObject, 'Value') returns toggle state of draw_yellow_button

```

```

function draw_black_button_Callback(hObject, eventdata, handles)
% hObject handle to draw_black_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.reset_grid=handles.vis_grid;
set(handles.draw_mode, 'visible', 'on')
button_enable(handles, 'off', 'DRAW');
while 1
    x=1;
    [y,x]=ginput(1);
    if ~isempty(x)
        x=round(x);
        y=round(y);
        if (x>0) && (x<handles.N+1) && (y>0) && (y<handles.N+1)
            %handles.vis_grid(x,y)=~(handles.vis_grid(x,y));

```

```

        handles.vis_grid(x,y)=1*~(handles.vis_grid(x,y));
        guidata(hObject, handles);
        plot_grid(handles)
    end
else
    break
end
end %while
set(handles.draw_mode, 'visible', 'off')
button_enable(handles, 'on', 'DRAW');
if get(handles.inf_loop, 'value')==1
    button_enable(handles, 'off', 'INF');
end
% Hint: get(hObject, 'Value') returns toggle state of draw_black_button

```

```

function draw_blue_button_Callback(hObject, eventdata, handles)
% hObject    handle to draw_blue_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.reset_grid=handles.vis_grid;
set(handles.draw_mode, 'visible', 'on')
button_enable(handles, 'off', 'DRAW');
while 1
    x=1;
    [y,x]=ginput(1);
    if ~isempty(x)
        x=round(x);
        y=round(y);
        if (x>0) && (x<handles.N+1) && (y>0) && (y<handles.N+1)
            %handles.vis_grid(x,y)=~(handles.vis_grid(x,y));
            handles.vis_grid(x,y)=4*~(handles.vis_grid(x,y));
            guidata(hObject, handles);
            plot_grid(handles)
        end
    else
        break
    end
end %while
set(handles.draw_mode, 'visible', 'off')
button_enable(handles, 'on', 'DRAW');
if get(handles.inf_loop, 'value')==1
    button_enable(handles, 'off', 'INF');
end
end

```

```

% Hint: get(hObject,'Value') returns toggle state of draw_blue_button

% --- Executes on button press in clear_grid.
function clear_grid_Callback(hObject, eventdata, handles)
% hObject    handle to clear_grid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.reset_grid=handles.vis_grid;
handles.vis_grid=zeros(size(handles.vis_grid));
guidata(hObject, handles);
plot_grid(handles)

% --- Executes on button press in inverse_grid.
function inverse_grid_Callback(hObject, eventdata, handles)
% hObject    handle to inverse_grid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.vis_grid=~handles.vis_grid;
guidata(hObject, handles);
plot_grid(handles)

% --- Executes on button press in reset_gr.
function reset_gr_Callback(hObject, eventdata, handles)
% hObject    handle to reset_gr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if size(handles.vis_grid)~=size(handles.reset_grid)
    set(handles.N_size,'value',size(handles.reset_grid,1));

set(handles.grid_size_N,'string',num2str(size(handles.reset_grid,1)))
;
    handles.N=size(handles.reset_grid,1);
end
handles.vis_grid=handles.reset_grid;
guidata(hObject, handles);
plot_grid(handles)

function input_signal_Callback(hObject, eventdata, handles)
% hObject    handle to input_signal (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of input_signal as text
%         str2double(get(hObject,'String')) returns contents of
input_signal as a double
n=str2double(get(hObject,'String'));
handles.input_source=n;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function input_signal_CreateFcn(hObject, eventdata, handles)
% hObject handle to input_signal (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function field_effect_Callback(hObject, eventdata, handles)
% hObject handle to field_effect (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of field_effect as text
%         str2double(get(hObject,'String')) returns contents of
field_effect as a double
n=str2double(get(hObject,'String'));
handles.field_effect=n;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function field_effect_CreateFcn(hObject, eventdata, handles)
% hObject handle to field_effect (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB

```

```

% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function decay_rate_Callback(hObject, eventdata, handles)
% hObject    handle to decay_rate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of decay_rate as text
%         str2double(get(hObject,'String')) returns contents of
decay_rate as a double
n=str2double(get(hObject,'String'));
handles.decay_rate=n;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function decay_rate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to decay_rate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function mean_value_Callback(hObject, eventdata, handles)
% hObject    handle to mean_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of mean_value as text
%         str2double(get(hObject,'String')) returns contents of
mean_value as a double
n=str2double(get(hObject,'String'));
handles.mean_value=n;
guidata(hObject, handles);

```

```

% --- Executes during object creation, after setting all properties.
function mean_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mean_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function fluc_value_Callback(hObject, eventdata, handles)
% hObject    handle to fluc_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of fluc_value as text
%         str2double(get(hObject,'String')) returns contents of
fluc_value as a double
n=str2double(get(hObject,'String'));
handles.fluc_value=n;
guidata(hObject, handles);

```

```

% --- Executes during object creation, after setting all properties.
function fluc_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fluc_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function max_weight_value_Callback(hObject, eventdata, handles)
% hObject    handle to max_weight_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of max_weight_value as
text
%         str2double(get(hObject,'String')) returns contents of
max_weight_value as a double
n=str2double(get(hObject,'String'));
handles.max_weight_value=n;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function max_weight_value_CreateFcn(hObject, eventdata, handles)
% hObject    handle to max_weight_value (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
%set(hObject,'String',num2str(handles.max_weight_value));

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in reset_weight.
function reset_weight_Callback(hObject, eventdata, handles)
% hObject    handle to reset_weight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.calc_weight=[2*ones(4,8);2*ones(4,8)];
handles.sd=zeros(8,8);

```

```

handles.stdp=zeros(8,handles.time_window+handles.delay+1);
guidata(hObject, handles);

%Spiking Neural Network Controller Implementation
function [handles]=calc_game(handles)

%PARAMETERS

M=8; % number of synapses per neuron
D=handles.delay; % Conduction delay between 2 neurons D=20

%Presynaptic neurons % Postsynaptic neurons % total number
Ne=4; Ni=4; N=Ne+Ni;

%Neuron parameters
a=[0.02*ones(Ne,1); 0.02*ones(Ni,1)];
d=[ 8*ones(Ne,1); 8*ones(Ni,1)];

%assign handles to matrix
s=handles.calc_weight; %weight
time_window=handles.time_window; %time window

%Matrix that stores STDP value for each neuron from 0 to max time window
STDP=handles.stdp;
v = -65*ones(N,1); % initial values of membrane
potential
u = 0.2.*v; % initial values of recovery variable
firings=handles.firings; % spike timings

%Initial values of Noise Input
mean1=handles.mean_value; %5
fluc1=handles.fluc_value; %15

for t=1:time_window % simulation of 400ms

%INPUT

I=zeros(N,1);
%for h=1:8 %add random noise to each of the 8 neurons
%I(h)=mean1+rand*fluc1;
%end
%I=I+mean1+rand*fluc1;

```

```

I(ceil(N*rand))=mean1+rand*fluc1; %receives some random input

%Directional input from the food, 8 single neurons correspond to 8 different
directions
%The magnitude of the input is 40.

[RFF,CFF]=find(handles.calc_grid==2); %find the location of the food
indrc=[RFF,CFF];
if ~isempty(indrc) %find the location of the robot
[RII,CII]=find(handles.calc_grid==1);

%compare
nn=length(RFF);

for m=1:nn %use for loop to calculate the input (from all the food) to
all the object direction neuron (more than one food)

%Using distance code according the cognitive neural science
%parameter of the field

decay_rate=handles.decay_rate; %0.85
input_source=handles.input_source; %5 %starting input value
in order to make sure the input's effect is greater than noise
field_effect(1)=handles.field_effect; %22 %the maximum value
of the field (when the distance from food to the robot=0, NB, it is 0, not
1)

for dd=2:61 %assign 60 different values to field distance,
using exponential decay, 0.85 is the rate (the larger the slower)
field_effect(dd)=decay_rate*field_effect(dd-1);
end

field=0;

%Adding field to the food %calculate the shortest distance from food
to robot %diagonal grid has the same distance with the straight one.
if abs(RFF(m)-RII)>abs(CFF(m)-CII) %when the vertical distance
from object is greater than horizontal, use the vertical distance as
reference.
field=field_effect(abs(RFF(m)-RII)+1); %add '1' make sure the matrix
dimension fix with each other
end

```

```

if abs(RFF(m)-RII)<abs(CFF(m)-CII)    %when the horizontal distance
from object is greater than vertical, use the horizontal distance as
reference.
field=field_effect(abs(CFF(m)-CII)+1);
end

if abs(RFF-RII)==abs(CFF-CII)
field=field_effect(abs(CFF(m)-CII)+1);
end

%find the nearest neuron and assign input to it

if RFF(m)<RII && CFF(m)==CII
I(1)=I(1)+input_source+field;
end

if RFF(m)<RII && CFF(m)>CII
I(2)=I(2)+input_source+field;
end

if RFF(m)==RII && CFF(m)>CII
I(3)=I(3)+input_source+field;
end

if RFF(m)>RII && CFF(m)>CII
I(4)=I(4)+input_source+field;
end

if RFF(m)>RII && CFF(m)==CII
I(5)=I(5)+input_source+field;
end

if RFF(m)>RII && CFF(m)<CII
I(6)=I(6)+input_source+field;
end

if RFF(m)==RII && CFF(m)<CII
I(7)=I(7)+input_source+field;
end

if RFF(m)<RII && CFF(m)<CII

```

```

I(8)=I(8)+input_source+field;
end
end
end

%Switch off the neuron located at the direction of the obstacles v=-65
%creat a 3*3 chess_gird around the robot to check surround obstacles (ob==4)

[RO,CO]=find(handles.calc_grid==1);
chess_grid=zeros(3);
chess_grid=handles.calc_grid(RO-1:RO+1,CO-1:CO+1);

%switch off the correspond nuerons v=-65 (the rest potential)

if chess_grid(1,2)==4
v(1)=-65;
end

if chess_grid(1,3)==4
v(2)=-65;
end

if chess_grid(2,3)==4
v(3)=-65;
end

if chess_grid(3,3)==4
v(4)=-65;
end

if chess_grid(3,2)==4
v(5)=-65;
end

if chess_grid(3,1)==4
v(6)=-65;
end

if chess_grid(2,1)==4
v(7)=-65;
end

if chess_grid(1,1)==4

```

```

v(8)=-65;
end

%Izhikevich model: resetting all the parameters when neuron fires

fired = find(v>=30);    % indices of fired neuron
v(fired)=-65;          % reset v,u and STDP when neuron fires.
u(fired)=u(fired)+d(fired);
STDP(fired,t+D)=0.1;

%fully connected network(except self connection)
%creat the index of all the neurons in the network
for j=1:N
postmatrix(j)=j;
end

%creat the index of all the postsynaptic neurons
for i=1:N
post{i}=find(postmatrix~=0);

%creat the index of all the pre excitory neurons
pre{i}=find(s(i,:)>0);
end

sd=handles.sd;    %derivatives handles
%STDP implementation    %single neuron potentiation
%if fired==3
%sd(3,2)=sd(3,2)+STDP(2,t);
%end;
%sd(2,3)=0.5;

%potentiaion(when presynaptic spikes arrives before POSTSYNAPTIC
neuron fires)
FL=length(fired);
for h=1:FL

%exchange the column and row of STDP in order to match the matrix
dimension of sd.
RE_STDP=zeros(time_window+20+1,N);

```

```

for i=1:N
RE_STDP(:,i)=reshape(STDP(i,:),time_window+20+1,1);
end

sd(fired(h),pre{fired(h)})=sd(fired(h),pre{fired(h)})+RE_STDP(t,pre{f
ired(h)}); %potentiation

end

firings=[firings;t*ones(length(fired),1),fired]; %the matrix
that stores the time and corresponding spikes

%depression
%Depression(when presynaptic spikes arrives after postsynaptic neuron
fires)

km=find(firings(:,1)==t-D+1); %k index of neuron No in firings that
fire at t-d+1
for dp=1:length(km)

sd(post{firings(km(dp),2)},firings(km(dp),2))=sd(post{firings(km(dp),
2)},firings(km(dp),2))-1.2*STDP(post{firings(km(dp),2)},t+D);

%pulse-coupled neural networks(PCNN),the firing of one neuron
influnce
%the others.

I(post{firings(km(dp),2)})=I(post{firings(km(dp),2)})+s(:,firings(km(
dp),2)); %reshape(s(fired(h),:),M,1)
end

%save data back to handles
handles.firings=firings;
handles.sd=sd;

%calculate the No.of spikes occured at the end of simulation time window
if t==time_window

```

```

    for neuronfired=1:N
        AA=find(firings(:,2)==neuronfired);           %a list of index of the
indicated fired neuron
        BB(neuronfired)=length(AA);                 %record the No. of spikes for
each neuron
    end

    total_fired=max(BB);                            %find the maximum No. of spikes
occured
    sum_fired=find(BB==total_fired) ;              %record the neurons that
fired the maximum No. of spikes

    %final_fired=sum_fired(ceil(rand*length(sum_fired))); %randomly
choose form these neurons, only one neuron will fire finally

    final_fired=0;

%movement calculation
%calculate the overall direction of the movement according to fired
neuron(Monkey Experiment)
moving_calc=3*ones(1,2);

if ~isempty(find(sum_fired==1)) %moving up
moving_calc(1)=moving_calc(1)-1;
end

if ~isempty(find(sum_fired==2))
moving_calc(1)=moving_calc(1)-1;
moving_calc(2)=moving_calc(2)+1;
end

if ~isempty(find(sum_fired==3)) %moving right
moving_calc(2)=moving_calc(2)+1;
end

if ~isempty(find(sum_fired==4))
moving_calc(1)=moving_calc(1)+1;
moving_calc(2)=moving_calc(2)+1;
end

if ~isempty(find(sum_fired==5)) %moving down
moving_calc(1)=moving_calc(1)+1;
end

```

```

if ~isempty(find(sum_fired==6))
moving_calc(1)=moving_calc(1)+1;
moving_calc(2)=moving_calc(2)-1;
end

if ~isempty(find(sum_fired==7))    %moving left
moving_calc(2)=moving_calc(2)-1;
end

if ~isempty(find(sum_fired==8))
moving_calc(1)=moving_calc(1)-1;
moving_calc(2)=moving_calc(2)-1;
end

%additional cases: if the robot moves more than one grid,randomly choose
one from two.

if moving_calc==[1,4]
moving_calc(1)=moving_calc(1)+1;
moving_calc(2)=moving_calc(2)-round(rand);
end

if moving_calc==[2,5]
moving_calc(2)=moving_calc(2)-1;
moving_calc(1)=moving_calc(1)+round(rand);
end

if moving_calc==[4,5]
moving_calc(2)=moving_calc(2)-1;
moving_calc(1)=moving_calc(1)-round(rand);
end

if moving_calc==[5,4]
moving_calc(1)=moving_calc(1)-1;
moving_calc(2)=moving_calc(2)-round(rand);
end

if moving_calc==[5,2]
moving_calc(1)=moving_calc(1)-1;
moving_calc(2)=moving_calc(2)+round(rand);
end

if moving_calc==[4,1]

```

```

moving_calc(2)=moving_calc(2)+1;
moving_calc(1)=moving_calc(1)-round(rand);
end

if moving_calc==[2,1]
moving_calc(2)=moving_calc(2)+1;
moving_calc(1)=moving_calc(1)+round(rand);
end

if moving_calc==[1,2]
moving_calc(1)=moving_calc(1)+1;
moving_calc(2)=moving_calc(2)+round(rand);
end

%Assign the overall direction to the switch case,only one movement
direction each step
%only one neuron will fire finally and make sure the fired neuron's state
is not at rest
if moving_calc==[2,3] & v(1)~=-65
final_fired=1;
end

if moving_calc==[2,4] & v(2)~=-65
final_fired=2;
end

if moving_calc==[3,4] & v(3)~=-65
final_fired=3;
end

if moving_calc==[4,4] & v(4)~=-65
final_fired=4;
end

if moving_calc==[4,3] & v(5)~=-65
final_fired=5;
end

if moving_calc==[4,2] & v(6)~=-65
final_fired=6;
end

```

```

if moving_calc==[3,2] & v(7)~=-65
final_fired=7;
end

if moving_calc==[2,2] & v(8)~=-65
final_fired=8;
end

final_fired;

%creat color property matrix, record the passing times of the robot
[RIL,CIL]=find(handles.calc_grid==1);
handles.numberpass_grid(RIL,CIL)=handles.numberpass_grid(RIL,CIL)+1;

%The deeper the color more times the robot pass a grid

NPA_1=1;
NPA_2=4;
NPA_3=8;
NPA_4=30;
NPA_5=60;
NPA_6=90;

if NPA_1<=handles.numberpass_grid(RIL,CIL) &&
handles.numberpass_grid(RIL,CIL)<NPA_2
TRO=5;
end
if NPA_2<=handles.numberpass_grid(RIL,CIL) &&
handles.numberpass_grid(RIL,CIL)<NPA_3
TRO=6;
end
if NPA_3<=handles.numberpass_grid(RIL,CIL) &&
handles.numberpass_grid(RIL,CIL)<NPA_4
TRO=7;
end
if NPA_4<=handles.numberpass_grid(RIL,CIL) &&
handles.numberpass_grid(RIL,CIL)<NPA_5
TRO=8;
end
if NPA_5<=handles.numberpass_grid(RIL,CIL) &&
handles.numberpass_grid(RIL,CIL)<NPA_6
TRO=9;
end
end

```

```

if NPA_6<=handles.numberpass_grid(RIL,CIL)
TRO=9;
end

% movement decision
switch final_fired
case 1 %moving up
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI-1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO; %the grid value when the robot
left its original place
case 2
[RI,CI]=find(handles.calc_grid==1);
%if the robot is moving diagonally(moving 2 grid at one iteration),
the first grid value need to be recorded in order to avoid overlapping)
RIC=handles.calc_grid(RI-1,CI);
handles.calc_grid(RI-1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI,CI+1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=RIC;
case 3 %moving right
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI,CI+1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
case 4
[RI,CI]=find(handles.calc_grid==1);
RIC=handles.calc_grid(RI,CI+1);
handles.calc_grid(RI,CI+1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI+1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=RIC;
case 5 %moving down
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI+1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
case 6
[RI,CI]=find(handles.calc_grid==1);
RIC=handles.calc_grid(RI+1,CI);
handles.calc_grid(RI+1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
[RI,CI]=find(handles.calc_grid==1);

```

```

handles.calc_grid(RI,CI-1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=RIC;
case 7 %moving left
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI,CI-1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
case 8
[RI,CI]=find(handles.calc_grid==1);
RIC=handles.calc_grid(RI,CI-1);
handles.calc_grid(RI,CI-1)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=TRO;
[RI,CI]=find(handles.calc_grid==1);
handles.calc_grid(RI-1,CI)=handles.calc_grid(RI,CI);
handles.calc_grid(RI,CI)=RIC;
end

end

%IZHIKEVICH MODEL IMPLEMENTATION
v=v+0.5*((0.04*v+5).*v+140-u+I); % for numerical
v=v+0.5*((0.04*v+5).*v+140-u+I); % stability time
u=u+a.*(0.2*v-u); % step is 0.5 ms
STDP(:,t+D+1)=0.95*STDP(:,t+D); % tau = 20 ms
handles.stdp=STDP;

end

function []=plot_grid(handles)
N=handles.N;
axes(handles.axes1);
%colormap hot(2)
MM=[1,1,1;0,0,0;1,0,0;1,1,0;0,0,1;0,1,0;0,0.8,0;0,0.6,0;0,0.4,0;0,0.2
,0;];%creat matrix for colormap storing in future
clims=[0 9]; %assign five colors to the
image:white(0),black(1),red(2),yellow(3),blue(4),green(5-9)
imagesc(handles.vis_grid,clims);
if all(handles.vis_grid) %help colormap
colormap ([0 0 0])
else
colormap(MM)
end
set(gca,'xtick',[1:N])

```

```

-.5 , 'ytick', [1:N]-.5, 'yticklabel', [], 'xticklabel', [], 'xcolor', [.7 .7
.7], 'ycolor', [.7 .7 .7], 'GridLineStyle', '-');
grid on
if get(handles.grid_lines, 'value')==1 axis on
else
    axis off
end

function [a]=artificial_life();
a=zeros(1);
a(1)=1;
%a(1,2)=0;
%a(2,1)=0;
%a(2,2)=0;

function [a]=obstacle1();
a=zeros(1);
a=4+a;

function [a]=food1();
a=zeros(1);
a=2+a;

function [a]=turtle();
a=zeros(14);
a([2,11,12],1+[2:3,10:11])=1;
a(3,1+[3:10])=1;
a(5:6, 1+[3,10])=1;
a([7,12],1+[5,8])=1;
a(8,1+[3:4,6:7,9:10])=1;
a(9,1+[4:5,8:9])=1;
a(10,1+[2,4,9,11])=1;
a(13,1+[6:7])=1;

function []=button_enable(handles,status,situation)
f_inf=['set(handles.increase_iterations,'enable',status);',...
    'set(handles.decrease_iterations,'enable',status);',...
    'set(handles.iterations_str,'enable',status);',...
    'set(handles.units,'enable',status);',...
    'set(handles.hundreds,'enable',status);',...
    'set(handles.tens,'enable',status);'];

```

```

f_other=['set(handles.reset_gr,'enable',status);',...
        'set(handles.draw_red_button,'enable',status);',...
        'set(handles.draw_blue_button,'enable',status);',...
        'set(handles.draw_black_button,'enable',status);',...
        'set(handles.clear_grid,'enable',status);',...
        'set(handles.file,'enable',status);',...
        'set(handles.grid_size_N,'enable',status);',...
        'set(handles.N_size,'enable',status);',...
        'set(handles.N_size,'enable',status);',...
        'set(handles.inf_loop,'enable',status);'];
switch situation
case 'INF'
    eval(f_inf);
case 'DRAW'
    set(handles.speed_slider,'enable',status);
    set(handles.start_stop,'enable',status);
    eval(f_other);
    eval(f_inf);
case 'START_STOP'
    eval(f_other);
    eval(f_inf);
end

% --- Executes on button press in tens.
function tens_Callback(hObject, eventdata, handles)
% hObject    handle to tens (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of tens

% --- Executes on button press in units.
function units_Callback(hObject, eventdata, handles)
% hObject    handle to units (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of units

% --- Executes on button press in hundreds.
function hundreds_Callback(hObject, eventdata, handles)

```

```
% hObject    handle to hundreds (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of hundreds
```

```
%
```

```
-----
function Untitled_1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
```

9. References and Bibliography

9.1 References

1. Wulfram Gerstner, Werner Kistler 'Spiking Neuron Models-Single Neurons, Populations, Plasticity' Cambridge University Press 2002
2. Keith S Elmslie, 'Membrane Potential' , ENCYCLOPEDIA OF LIFE SCIENCE, Nature Publishing Group 2001
3. Peter C Ruben, 'Action Potentials: Generation and Propagation', ENCYCLOPEDIA OF LIFE SCIENCE, Nature Publishing Group 2001
4. Stephen R Williams, Greg J Stuart, 'Synaptic Integration' ENCYCLOPEDIA OF LIFE SCIENCE, Macmillan Publishers Ltd, Nature Publishing Group 2002
5. G.N.Reeke, R.R.Poznanski, K.A.Lindsay, J.R.Rosenberg, and O.Sporns 'Modeling in the Neurosciences-From Biological Systems to Neuromimetic Robotics' second edition by Taylor & Francis Group.LLC 2005
6. Micheal S.Gazzaniga, Richard B.Ivry, George R.Mangun, 'Cognitive Neuroscience-The biology of the mind' third edition, New York.London 1956
7. Eugene M.Izhikevich 'Dynamical Systems in Neuroscience-The geometry of Excitability and Busting' , The MIT Press Cambridge, Massachusetts, London, England 2007
8. Eugene M.Izhikevich 'Simple model of Spiking Neurons' IEEE TRANSACTIONS ON NEURAL NETWORKS,VOL.14,NO.6, NOVEMBER 2003
9. Eugene M. Izhikevich 'Which Model to Use for Cortical Spiking Neurons' IEEE TRANSACTIONS ON NEURAL NETWORKS,VOL.15,NO.5, SEPTEMBER 2004
10. Eugen M.Izhikevich 'Polychronization: Computation with Spikes', The Neuroscience Institute, USA 2006, Neural Computation 18, 245-282, 2006
11. D.O.HEBB (1949), 'The Organization of Behaviour-A Neuropsychological Theory', Lawrence Erlbaum Associates, New Jersey, Reprinted 2002
12. Patrick D. Roberts Curtis C. Bell 'Spike timing dependent synaptic plasticity in biological systems' Neurological Sciences Institute, Oregon Health & Science University, USA 2002
13. Sen Song, Kenneth D.Miller, L.F.Abbott 'Competitive Hebbian learning through spike-timing-dependent synaptic plasticity', Nature Neural Science, Vol.3, No.9 September 2000

14. Di Paolo E.A 'Spike-Timing Dependent Plasticity for Evolved Robots', International Society for Adaptive Behavior, Vol.10(3-4):243-263. 2002
15. Matlab-Central <http://www.mathworks.com/matlabcentral/fileexchange/>
16. Matlab product help
17. RUDRA PRATAP, 'Getting started with Matlab-A Quick Introduction for Scientists and Engineers' New York. Oxford, Oxford University Press 2002
18. K. I. Blum and L. F. Abbott, 'A model of spatial map formation in the hippocampus of the rat', Neural Computation, Vol8, 85-93, 1996
19. W. Gerstner and L. F. Abbott, 'Learning navigational maps through potentiation and modulation of hippocampal place cells', Journal of Computational Neuroscience, Vol4, 79-94, 1997

9.2 Bibliography

Di Paolo E.A 'Spike-Timing Dependent Plasticity for Evolved Robots', International Society for Adaptive Behavior, Vol.10(3-4):243-263. 2002

David M.Halliday and Jay R.Rosenberg 'Time and Frequency Domain Analysis of Spike Train and Time Series Data', Department of Electronics, University of York

David M.Halliday 'Spike train analysis for neural systems', Department of Electronics, University of York

David M.Halliday, D.3rd Year Neural Network Course 2009
Ref Type: Unpublished Work

D.O.HEBB (1949), 'The Organization of Behaviour-A Neuropsychological Theory', Lawrence Erlbaum Associates, New Jersey, Reprinted 2002

Eugene M.Izhikevich 'Dynamical Systems in Neuroscience-The geometry of Excitability and Busting' The MIT Press Cambridge, Massachusetts, London, England, 2007

Eugen M.Izhikevich 'Polychronization: Computation with Spikes', The Neuroscience Institute, USA 2006, Neural Computation 18, 245-282, 2006

G.N.Reeke, R.R.Poznanski, K.A.Lindsay, J.R.Rosenberg,and O.Sporns 'Modeling in the Neurosciences-From Biological Systems to Neuromimetic Robotics' second edition by Taylor & Francis Group.LLC 2005

James A. Anderson 'An Introduction to Neural Networks' Massachusetts Institute of Technology
United States of America 1995

Matlab-Central <http://www.mathworks.com/matlabcentral/fileexchange/>

Micheal N Shadlen and William T Newsom 'Is there a signal in the noise?' Stanford University
School of Medicine, Standford, USA 1995

Micheal S. Gazzaniga, Richard B. Ivry, George R. Mangun, 'Cognitive Neuroscience-The biology of
the mind' third edition, New York. London 1996

Mitchell A. C 'Evolving Spiking neural networks as Robot Controllers " 2000

Maass W and Bishop C. 'Pulsed Neural Networks'. Cambridge, MA: MIT Press 1998.

Renaud Jolivet, Timothy J. Lewis, and Wulfram Gerstner, 'Generalized Integrate-and-Fire Models
of Neuronal Activity Approximate Spike Trains of a Detailed Model to a High Degree of Accuracy'
J Neurophysiol 92: 959-976, The American Physiological Society, 2004

Robert Guitig & Haim Sompolinsky, 'The tempotron: a neuron that learns spike timing-based
decisions' nature publishing group 2006

Alain Destexhe, Michael Rudolph and Denis Pare 'The high-conductance state of neocortical
neurons in vivo' Nature Reviews, Neuroscience, Vol.4, 2003

William R Softky 'Simple codes versus efficient codes' National institutes of Health, Bethesda,
USA, 1995

Wulfram Gerstner, Werner Kistler 'Spiking Neuron Models-Single Neurons, Populations, Plasticity'
Cambridge University Press 2002