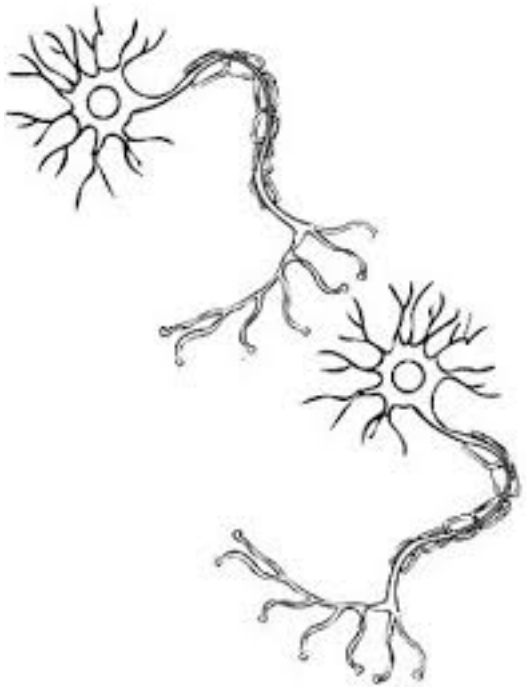
Imperial College London

Department of Electrical & Electronic Engineering

2016 Final Year Project Thesis



Thesis Title: **Machine Learning On The SpiNNaker Platform For The Visual Input Processing Of A Ball Catching Robot**

Student Name: **Hanyi Hu**

CID: **00745392**

Supervisor: **Dr Konstantin Nikolic, Dr Benjamin Evans**

Second Marker: **Dr. Christos Bouganis**

Abstract

Even a simple task for human beings, such as catching a ball in motion, requires basic intelligence and considering multi-factors in the environment. Humans have developed various learning capabilities for performing different tasks through an evolutionary process. However, it is still a great challenge for engineers and scientists to deploy similar abilities to a robot. Driven by this goal and challenge, this report presents an investigation of different ball catching mechanisms and a robotic system design, using a dynamic vision sensor -DVS128- and a neural network simulation platform -SpiNNaker[2][1]. DVS128 is a special bio-inspired motion detection camera, sensitive to the relative luminosity changes and responds asynchronously. The neural network model performs machine learning and decision making functions by the leaky integrate & fire neuron model and Spiking-Timing Dependence Plasticity (STDP) learning rule. By simulating neuron models and the STDP learning rule, a two-layer neural network topology was constructed for the ball catching task.

Acknowledgement

I would like to begin by thanking my project supervisors Dr. Konstantin Nikolic and Dr. Benjamin Evans, who have inspired and encouraged me when I faced technical challenges in the project. It has been a pleasure to work with experts in this field and at the centre of the bio-inspired Technology, Imperial College London, where I developed relevant background knowledge and insights in neural science. I would also like to thank the previous research team and work done by Dario Magliocchetti-Lombi and Wenbo Li, which set the foundation for this project.

I would also like to thank all the teaching, support and administrative staff as well as my fellow colleagues who have assisted me throughout the duration of my course.

Finally, I would like to thank my family wholeheartedly, who have supported my education in this world-class institute, which was the starting point of my journey.

Table of Contents

0. Table of Content

1. Introduction

Different types of intelligent and robotic systems have been serving people in several ways, improving peoples' quality of life. For instance, there are robots available in the market for cleaning, education, surgical operations and entertainment[3]. There is also an increasing demand to implement various robotic systems in new industries. The demand for new systems lead to complex design challenges with regards to software and hardware implementation[4]. More specifically, performance and efficiency related challenges within the system, requiring decreased hardware usage or computational power[5].

With this appealing market potential as well as the need to address design challenges, this report presents the machine learning task and neural network model for the visual input processing of a humanoid robot, which catches a flying ball in motion mimicking human behaviour. In order to perform this task, the robot needs to recognise and track the object in different environments. There are various external factors in this case, including wind, ball size and speed. The task requires real time information monitoring and processing from different sensors. However, analysing this information especially for image processing, requires large computation, massive memory and fast communication at the interface between individual sensors and CPU. Although Moore's law describes the trend of growing computational power by explaining the relationship between time and transistors, it is still a challenge to meet the processing needed for a ball catching task[6]. Even if the hardware can meet the required specification, the system will be power consuming and inefficient. Therefore, it is pertinent to perform the task by using as minimal power and in the most efficient way possible.

Since this robot was designed for performing the task in a humanoid manner, the system can adopt the mechanism and physiological structure of a human, which makes it so called a "bio-inspired" system. The bio-inspired system is less power consuming making it suitable for such a robotic system[7]. In this particular case, the most crucial parts are the visual system and information processing system, corresponding to eyes and brain, similar to that in humans.

A dynamic visual sensor (DVS128) camera was selected for vision in the robotic system. The camera is sensitive and can respond to the luminosity changes in the camera field of the view (FOV) in discrete time. It can also record data in a special format, which will be discussed in detail later on.

In order to deploy similar brain functions such as learning and decision making, a novel neurophorphic platform SpiNNaker was selected, which can emulate and simulate different artificial neural

network (ANN) models and learning mechanisms from human brain. ANN models are particularly suitable and widely applied in applications like pattern recognition in computer vision. The ANN used in this project, is the leaky integrate & fire neuron model while the learning algorithm is spiking-timing dependence plasticity (STDP). With these two elementary neural network models being implemented on the SpiNNaker; the system can perform an unsupervised learning for processing incoming data.

1.1 Principle of Operation:

Figure 1 illustrates the high level design block diagram of the system. For a human catching a ball, we look at the ball through our eyes, the visual system and make decision using our brain. Likewise, the system would sense and track information of the objects from a front-end visual sensor. The data from it would be sent to a central information processing unit (CIPU). The CIPU can then perform decision making and learning tasks.

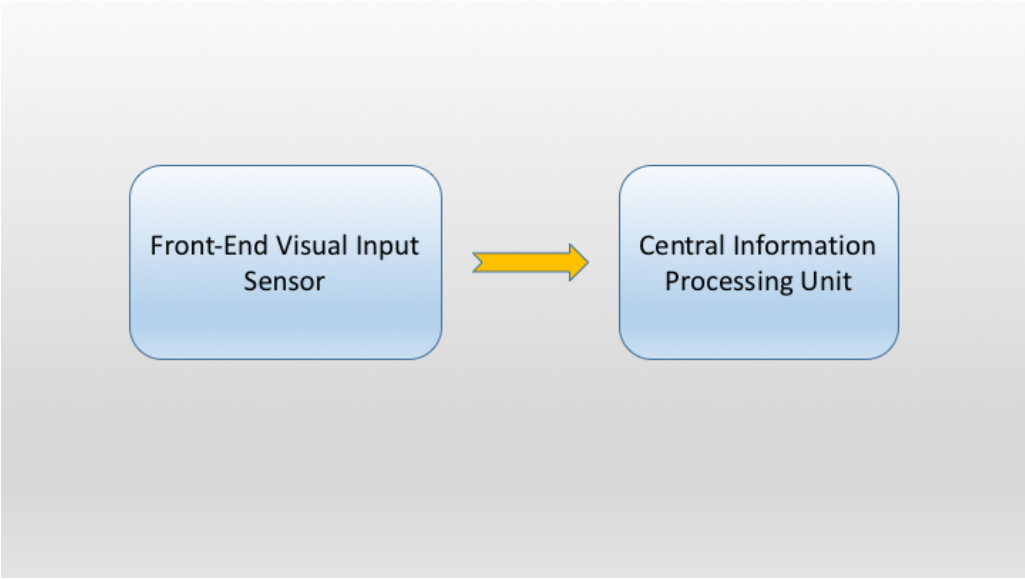


Figure 1.1 System Block Diagram

1.2 Project Scope & Objectives:

The project started with simplifying the problem to make a complete robotic design. The complexity of the system was gradually developed for tackling different cases. The final goal of this project was to build the system capable of determining the direction of the ball motion.

The first deliverable of the project was to verify one existing neural network topology. Li [29] built and trained a two-layer neural network topology, which represents the ball motion by 40 ANN neuron in the output layer. This topology was insufficient for the ball catching task, which will be discussed in Chapter 3. However, it would be ideal, if the new neural network topology could be built on top of this existing one.

Prior to the validation of the existing neural network topology, more data was required from DVS128 camera. Recording more data required emulating ball motion and processing the data in the same way.

As stated above, the second deliverable was to build and train a new neural network system, which determined the direction of ball motion. The following step tests the new system in different situation and determines its robustness and accuracy.

1.3 Report Layout:

The report structure is divided in to the following sections: hypothesis, proposed statement, implementation, testing, result and evaluation.

In Chapter 2, all relevant background research is discussed. The background research includes ball catching methods, real neural network, artificial neural network model and hardware specification. The hardware specification includes DVS128 and the SpiNNaker platform.

In Chapter 3, the system implementation is explored. The first implementation generates a ball from a Matlab animation and processes data recorded by the DVS128 camera. The second implementation simulated neuron models and different settings of the STDP learning rule on SpiNNaker, and

observes the behaviour of each models and the previous two-layer topology. Based on the simulation results, there were two topologies. The first one was a three-layer topology built upon the existing two-layer topology. However, the fact that this three-layer topology could not fulfill the design requirement resulted in changing the design strategy. Thus, a new two-layer topology was implemented and is presented at the end of this chapter.

In order to predict the performance of the trained network, the system was tested under different circumstances. Chapter 4, discusses the purpose, plan, and estimation for 8 different test cases, which will be reviewed in detail.

Chapter 5, presents testing results from the 8 different cases and analyses while Chapter 6 contains the evaluation of the trained neural network based on the results and conclusions in Chapter 5.

Chapter 7, concludes the testing result and summarises the evaluation in chapter 6. Lastly, in Chapter 8 the future work from this project will be discussed.

2. Background

In this chapter, relevant research covers the ball catching model, introduction to the neural network and the hardware specification of DVS128 and SpiNNaker platform.

2.1 Ball Catching Models:

2.1.1 Trajectory Prediction Model:

The most widely used model in various robotic systems is based on the calculation and prediction of the ball position at different times. The system integrates algorithms such as image processing, computer vision object recognition, ball motion and trajectory prediction[8]. In general, the system samples the visual information in continuous frames and extracts the ball position from each frame. It then calculates and tracks the ball trajectory, based on the ball position at different time, and controls actuators to move accordingly.

2.1.2 Ball Catching Model for Human:

Another novel approach to performing such a task was studied from the human ball catching model. Many studies and experiments have shown that human catching a ball in a completely different approach[9][10][11]. Imagine the process of catching a ball. Rather than looking at the spot, where the ball can possibly land, a human can react instantaneously without any calculation and keep staring at the ball throughout the process. Chapman in 1968 stated that if the ball trajectory is parabolic, a human can only catch the ball, where the tangent of elevation angle(α) between flying ball and human is constant. Therefore, the ball catcher needs to maintain the relationship in equation (1).

$$\frac{d(tan(\alpha))}{dt} = 0 \tag{1}$$

Figure 2.1.1, shows a 2 dimensional human ball catching model of the Chapman model.

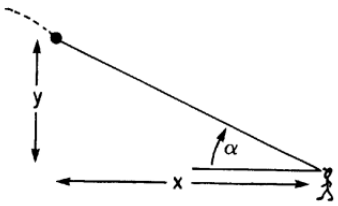


Figure 2.1.1 Angle of Elevation in Human Ball Catching Model. Figure from [13]

However, McLeod and Dienes claim that the model from Chapman assumes the ball trajectory follows a parabolic flight, which is barely the case in reality[13]. Also if the ball catcher is merely maintaining acceleration of tangent of the elevation angle(α) to 0, it is hard for ball catcher to compensate when the ball is landing. With this fact in mind, the ball catcher would keep the acceleration of tangent of elevation angle(α) constant instead. It should now be satisfy the relationship in equation (2).

$$\frac{d^2(tan(\alpha))}{d^2t} = 0 \tag{2}$$

Figure 2.1.2 shows the prediction of the ball catching outcome following the relationship in equation (1) and equation (2) respectively.

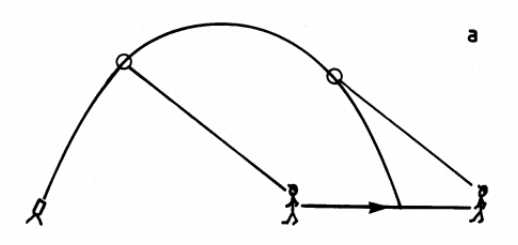


Figure 2.1.2.a A ball catcher maintains the tangent of angle of elevation

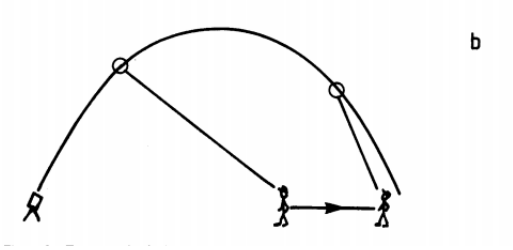


Figure 2.1.2.b A ball catcher maintains the acceleration of tangent of angle of elevation. Figure 2.1.2 From [13]

The relationship in equation (2) is important, which implies that the ball will be moving in the camera field of view in this model. Otherwise it would be difficult for the front end visual sensor, DVS128, to detect the ball motion.

In 1995, McBeath extended Chapman's model to three dimensions and claimed that the relative motion between the ball and catcher would be a upper straight line when tracking ball motion.[14]

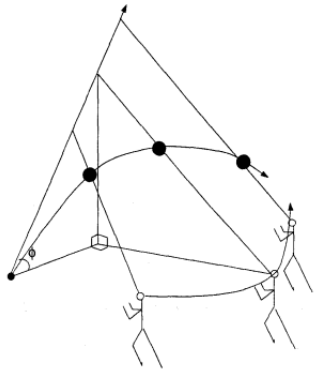


Figure 2.1.3 McBeath Linear Optic Trajectory Model. Figure from[15]

2.2 Introduction to Neural Network:

One of the most important abilities that makes humans different is their ability to adapt to different environments quickly and learn from experiences. Although this is debatable, it is undoubted that the learning ability plays an important role in defining intelligence. This crucial function comes directly from the neural network structure in the brain where it processes all external and internal information.

With regards to the design of the system, it needs to make accurate decisions from the input signal after learning. Moreover, the neural network has been widely used for the pattern recognition, which is one of the most critical functions for ball detection and recognition in the CIPU.

2.2.1 Physiology of Neural Network Structure:

The neural network is the structure and arrangement of neuron cells in the body and brain. The length of neuron cells in the brain are typically around 100 microns. Each neuron cell consists of dendrite, soma and axon.

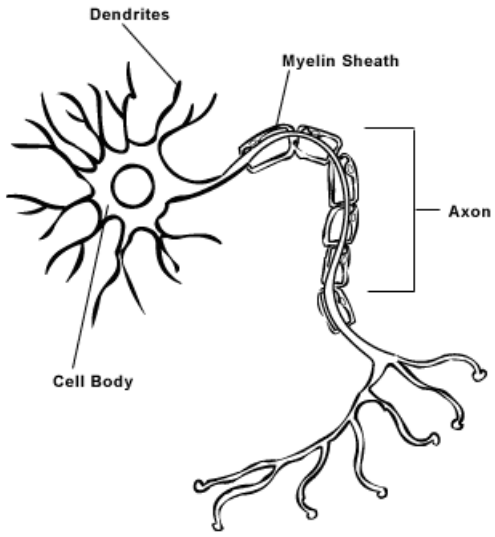


Figure 2.2.1 Neuron Cell Structure

The cellular membrane of a neuron is an ion barrier which separates the ions in the extracellular and intracellular environment. The operation of the neuron results from the flow of the ions across the cell membrane. When neuron is in its resting state, it is in a dynamic electrochemical equilibrium state. The typical bio-potential across cell membrane is around -65mV. There are different ion channels on the cell membrane. An external stimuli causes the sodium channels open (depolarisation), resulting in an inflow of sodium ions and increasing in the bio-potential across cell membrane. The bio-potential reaches its peak after the process saturates. The potassium ion channels then open (hyperpolarization), resulting in an outflow of the potassium ions, which brings the bio-potentials back to a lower bio-potential around -70mV. The sodium and potassium ionic pump then sends these two types of ions back to the resting state, known as the refractory phase of the neuron which lasts about 1-2ms. The process defining the significant change in the bio-potential in a short period of time, usually less than 2ms, is known as “spiking”or “firing”of the neuron.

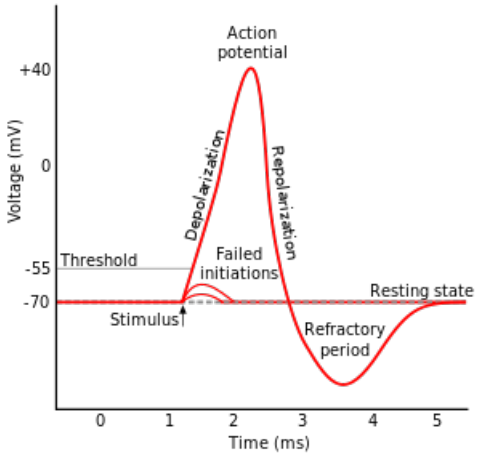


Figure 2.2.2 Action Potential of Neuron Cell

There are over 100 billion neuron in the brain connecting one to another. The connectivity of the neuron cells forms signal paths, which transmit the external stimuli to the brain. Stimulated spiking propagates from the soma to axon in the same neuron cell. Its axon connects to the dendrite of another neuron. The junctions between these axon and dendrite are called the synapse, and the junction gap between them is the synapse cleft. The former neuron is typically defined as the pre-synaptic neuron, while the latter is known as the post-synaptic neuron.

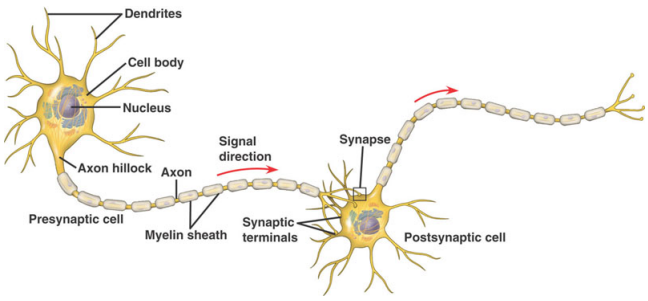


Figure 2.2.3.a Pre- and Post- synaptic neuron Connection

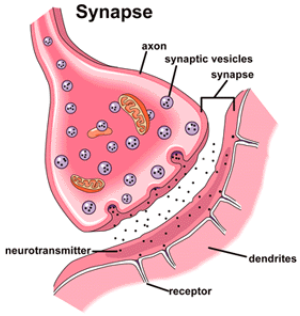


Figure 2.2.3.b Synapse Sturucture

2. Background

There are two types of synapses that impose the activation of the post-synaptic neuron, chemical synapse and electrical synapse.

For chemical synapses, action potential at the end of the axon opens the calcium channel, causing release of neurotransmitters at the synapse. The corresponding receptors on the dendrite of post-synaptic neuron will combine with these neurotransmitters. There are two types of classification, which are important in understanding and simulation in the later chapter. The neurotransmitter and receptor pair can either impose an excitatory post-synaptic potential (EPSP) or an inhibitory post-synaptic potential (IPSP). The EPSP synapse will increase post-synaptic neuron potential, while the IPSP have the opposite effect on the post-synaptic neuron potential. Neurotransmitter and its receptor can also be classified as inotropic and metabotropic. The inotropic receptors have faster response than that of the metabotropic ones. For instances, Glutamate as one type of neurotransmitter is excitatory, while the GABA as the other type is inhibitory. When Glutamate combines with non-NMBA receptor (Inotropic), the membrane potential increases rapidly. When GABA combines with GABAa receptor, the membrane potential responses quickly, but it is relatively slow response for GABAb receptor (metabotropic).[16]

The electrical synapse happens when the cleft between pre-synaptic neuron axon and post-synaptic neuron are within couple of nanometre close, which allows the ions flowing directly from pre-synaptic neuron to post-synaptic neuron. The response of electrical synapse is faster than the chemical type, but it is also very rare.

The role of synapses between neuron is not merely the signal paths, more importantly, it is also essential for the learning ability. Synapse weight or strength has positive correlation with the level of the neurotransmitter release from the pre-synaptic terminal. Although the level of the neurotransmitter release is characterised by multiple factors, it is influenced by the history of previous synaptic action potential. More specifically, it is related to the timing of the pre- and post-synaptic spiking.

2.2.2 Artificial Neural Network Models:

From the inspiration of real neural network in the body and brain, the type of neural network model used in this project is classified as spiking neural network. The particular spiking neural network neuron model and learning model are leaky integrate & fire neuron model and spiking-timing dependence plasticity learning rule. The following sections will discuss in detail the characteristics of these two important models.

2.2.2.1 Leaky Integrate & Fire Model:

The leaky integrate & fire model is fundamentally models and characterises the neuron as an RC electrical circuit, this is because the action potential is due to the flow of the ionic charge inside and outside of neuron cell[17].

From the previous section it can be noted that the neuron is in its electrochemical equilibrium state. The potential across the cell membrane is known as the resting potential, V_{rest}, while the instantaneous value is called the membrane potential. Due to the distribution and flow of ions between ions between the extracellular and intracellular environment, there exists a capacitance and resistance across the cell membrane, denoted as membrane capacitance, C_m, and membrane resistance, R_m respectively. The synaptic current injected from the pre-synaptic neuron can be described as the product of the synaptic conductance, $g_{syn}(t)$ and difference between the synaptic reversal potential (E_{syn}) and instantaneous membrane potential, $V_m(t)$. Figure 2.2.4 below illustrates a basic equivalent electrical circuit of the model.

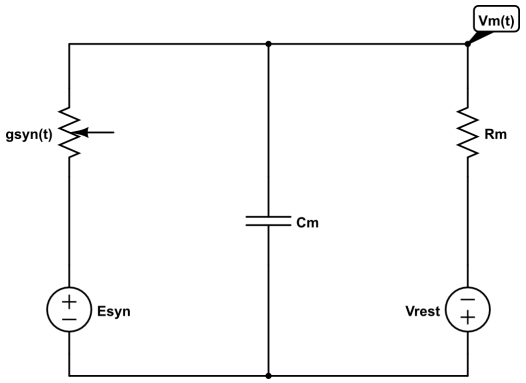


Figure 2.2.4 Equivalent Electrical Circuit of Cell Membrane

By applying Kirchhoff's law, the cell membrane equation can be expressed as in equation (3) and the synaptic conductance $g_{syn}(t)$ can be described by equation (4) below.

$$C\frac{dV_m(t)}{dt} + g_{syn}(t)(V_m(t) - E_{syn}) + \frac{V_m(t) - V_{rest}}{R} = 0 \tag{3}$$

$$g_{syn}(t) = g_o \times \exp\left(-\frac{t}{\tau_{syn}}\right) \tag{4}$$

From equation (3), the membrane potential can be obtained by solving the ordinary differential equation, giving a relationship of $V_m(t)$ from all other quantities. Equation (3) can be rearranged into the following form:

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{RC} = -\frac{g_{syn}(t) \times (V_m(t) - E_{syn})}{C} + \frac{V_{rest}}{RC} \tag{5}$$

Based on this simple membrane equation, there are many variants of the leaky integrate & fire model by making assumptions and tweaking the membrane equation, such as a current based model and conductance based model. Therefore, equation (5) will be solved in Chapter 3, where the variants of leaky integrate & fire model will be discussed.

When $V_m(t)$ reaches a threshold membrane potential, V_{th}, the neuron spikes. After the spiking event, the membrane potential immediately goes to the reset potential, V_{reset}, which is typically lower than V_{rest}. The neuron then experiences a refractory period, remaining at the same potential. After the refractory period, it gradually rises back to V_{rest}.

The other important effect of the model is called lateral inhibition. After one of the post-synaptic neuron spike, it then imposes this effect on the neighbouring neuron. The lateral inhibition effect can either clamp the membrane potential of the neighbour neuron to V_{rest} or inject a negative current, resulting a reduction in membrane potential, V_m. Lateral inhibition causes post-synaptic neuron spiking at different time. This property is important in learning, allowing post-synaptic neuron to learn different information.

2.2.2.2 Spiking-Timing Dependence Plasticity Learning Rule:

The Spiking-Timing Dependence Plasticity (STDP) learning rule is the one of the models, describing the synapse plasticity change relative to the timing of the pre-synaptic and post-synaptic neuron firing[18]. In STDP, the synapse strength is enhanced, if the pre-synaptic neuron fire before post-synaptic neuron, and vice versa. The enhancement and weakening period are known as the long-term potentiation (LTP) and long-term depression (LTD). STDP strengthen and weaken synapse strength by obeying the timing dependence, weight dependence and voltage dependence rule. Voltage dependence is rarely used in the model and will not be discussed.

In general, the timing dependence rule is described by the exponential function as in equation (6) and (7). τ_+ and $\tau_-$are LTP and LTD period respectively, while the A_+ and A_- are amplitude of changing in weighting.

$$W(\Delta t) = A_+ \exp \frac{-\Delta t}{\tau_+}, \text{If } \Delta t < 0 \tag{6}$$

$$W(\Delta t) = A_- \exp \frac{-\Delta t}{\tau_-}, \text{If } \Delta t > 0 \tag{7}$$

In Figure 2.2.5, the mid point of the timing dependence learning curve corresponds to the onset of the post-synaptic neuron spiking time. Intuitively seeing from the figure, the synapses with pre-synaptic neuron firing close to the post-synaptic neuron firing time are influenced more significantly than those spikes far away from the post-synaptic neuron. In principle, STDP tries to correlate with those pre-synaptic spike before post-synaptic spike and have an opposite effect on those pre-synaptic spike after.

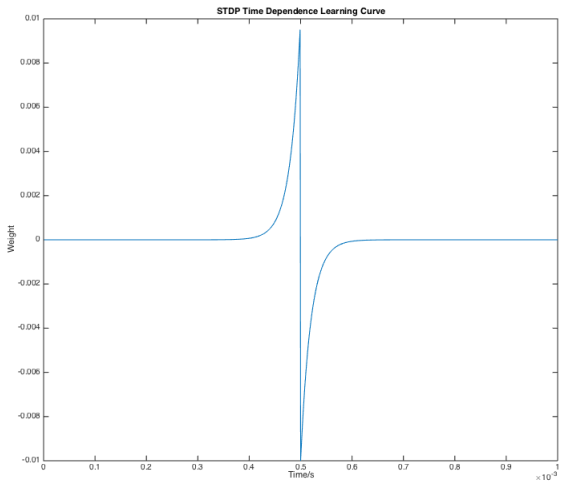


Figure 2.2.5 STDP Learning Curve Timing Dependence From Matlab Generation

In the weight dependence rule, A_+ and A_- can be replaced by functions of relative weight to maximum or minimum weight, rather than a constant. As the synapse weight approaches to the maximum or minimum value, the weight dependence rule decreases the amount of weight adding or subtracting to corresponding synapse. The weight dependence equation is shown in equation (8) and (9), where α and β are constant, and w_{max} and w_{min} represent the maximum and minimum weight respectively[19].

$$\Delta w_+ = \alpha_+ exp(-\beta_+ \frac{w - w_{min}}{w_{max} - w_{min}}) \tag{8}$$

$$\Delta w_- = \alpha_- exp(-\beta_- \frac{w_{max} - w}{w_{max} - w_{min}}) \tag{9}$$

2.2.2.3 Rank order & Polychronization:

In the real neural network structure section, there are different types of neurotransmitters and receptors for synapses. Apart from the synapse strength, the delay of synapses responses varies from one type to another, especially between inotropic and metabotropic ones. The synapse strength and delay can potentially encode the information from the external stimuli, which introduces the rank order and polychronisation neuron encoding in the artificial neural network[20][21].

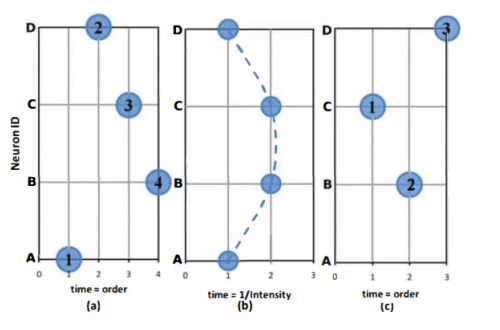


Figure 2.2.6 Rank Order. Figure From [20]

With a sequence of pre-synaptic neuron spiking, rank order emphases the importance of the beginning of the sequence, giving a spatial temporal representation of different input patterns. It coincides with the predicted learning outcome of STDP learning rule. In Figure 2.2.6 case b, neuron A and D are more essential than neuron B and C.

The polychronisation, on the other hand, tries to emphasise and encode the entire input spiking sequence[21]. In polychronisation, the earlier the input spike, the longer the delay of synapse. Therefore, the entire spiking sequence of one pattern will contribute to the membrane potential change in a short period of time, and causes the post-synaptic neuron spike. Spiking in different patterns cannot trigger the same post-synaptic neuron, as the delay of synapses will scatter over a period of time.

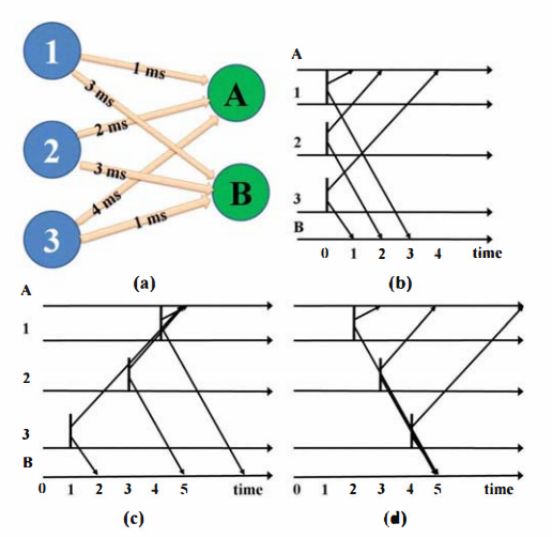


Figure 2.2.7 Polychronization. Figure From [20]

In figure 2.2.7, there are 3 input ANN neuron connecting to 2 other ANN layer in the output layer. In the case b, the spiking diverged to different time and neither of neuron in the higher layer. Whereas, in case c and d, different input pattern triggered the output neuron respectively.

2.3 Hardware Specification:

The hardware design of the robotic system is composed of the dynamic visual sensor camera, DVS128, and neural network simulation platform, SpiNNaker. This section will elaborate on the details of the DVS128 and SpiNNaker platform.

2.3.1 DVS128:

DVS128 is a special type of camera inspired from the human retina, it is sensitive and responds to relative luminosity change in the FOV. The luminosity change will appear as spikes on individual pixels sampling at 1MHz. The spiking event can either be an "ON"event or an "OFF"event, corresponding to increasing and decreasing in the luminosity change. This property of DVS128 makes it particularly good for capturing moving objects and to provide a spatial temporal representation[22].

Figure 2.3.1.a DVS128. Figure From[23]

Figure 2.3.1.b jAER Recording

The DVS128 does not record the spiking event information frame by frame, but uses the asynchronous event representation (AER). It consists of the coordinates of the spiking pixels, the "ONOFF"indication, and the corresponding time stamp of the spike for each event. Because the camera responds asynchronously to the luminosity change, it reduces redundant data. The CIPU can potentially process more data and lower the constraint of the interface between DVS128 to the later stage.

The 128 in the name stands for the resolution of the camera, meaning 128 by 128 pixels in the FOV. The data format currently in use is AEDAT 2.0, which uses 8 bytes for each event. The X address and Y address are 7 bits each, 1 bit for "ONOFF"event and 32 bits for timestamp, with rest of data unused for future development. The developer of DVS128, iniLabs, created the Java program called jAER, which user can record and save the data in AEDAT 2.0[24].

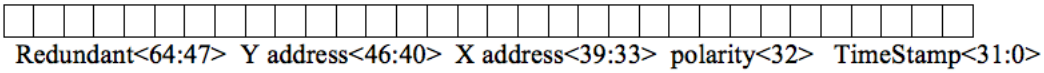


Figure 2.3.2 AEDAT 2.0 Format

The AEDAT data format currently cannot interface with SpiNNaker platform directly at the time of writing this report. Therefore, the data stream from DVS128 needs to be extracted and reformatted, so that it is compatible with SpiNNaker platform.

A python library (python-aer 0.1.3) publicly available on the python package index (PyPI) is able to extract and manipulate the AEDAT file to MAT file, which can be uploaded to SpiNNaker board [25]. The functions in python-aer library can not only reformat AEDAT data but also manipulate reformatted data.

2.3.2 SpiNNaker Platform:

The SpiNNaker platform is inspired by the neurobiology and specifically designed for simulating and emulating the behaviour of different artificial neural networks including leaky integrate & fire neuron model as well as spiking-timing dependence plasticity learning rule. SpiNNaker is a multi-core system and also have multiple chip mounted. Each SpiNNaker chip contains 18 ARM968 processing core with each core emulating up to 256 virtual artificial neuron. Data communication between cores takes place via multicasting packets which contains the information of issuer and destination only. Therefore, the communication in SpiNNaker is also address event represented asynchronously. The multicast router in each chip can replicate the data packets and distribute to different cores, so that one pre-synaptic neuron can connect to multiple post-synaptic neuron[26]. This fast computation protocol allows processing thousands of neuron spiking information in real time which is required in the robotic system.

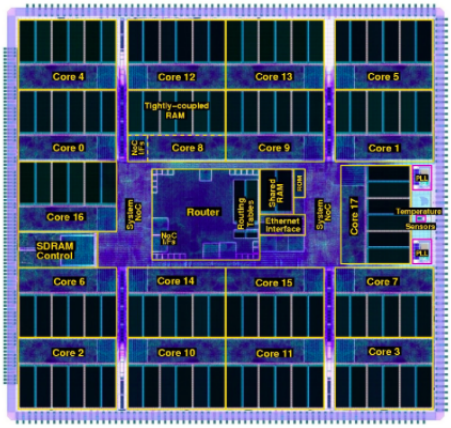


Figure 2.3.3 SpiNNaker ARM968 Chip Structure. Figure From [26]

The high level neural network descriptive and modelling language, such as pyNN and NeNGO, can describe and synthesis the neural network on SpiNNaker. The neural network modelling language used in this project is pyNN, and the descriptive pyNN code can be downloaded via the RJ45 Ethernet port.

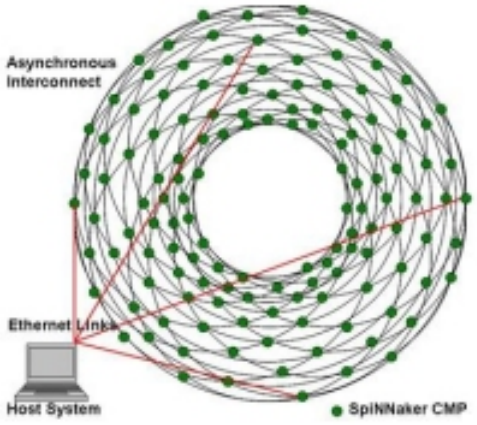


Figure 2.3.4 Host and SpiNNaker Board Connection. Figure From[27]

SpiNNaker supports simulation of many types of neuron model, including various leaky integrate & fire model, and Hodgkin-Huxley model. SpiNNaker platform also models spiking-timing dependence plasticity learning rule. Therefore, SpiNNaker has met the fundamental design requirement and the SpiNNaker platform used in this project is SpiNN-3.

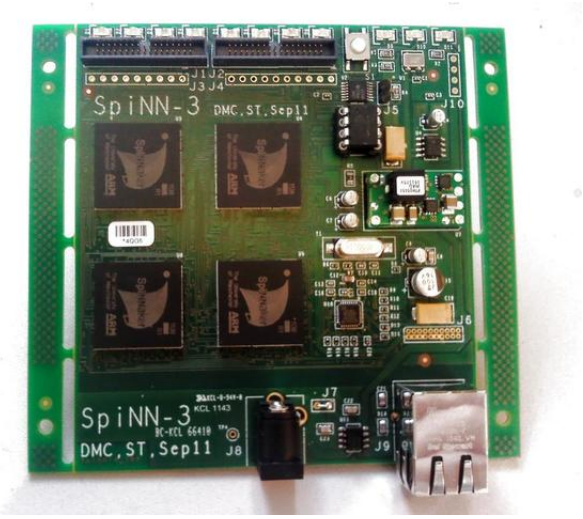


Figure 2.3.5 SpiNN-3. Figure From[28]

3. Implementation

The system was separated into two parts, front end visual sensor and central information processing unit. The implementation was conducted and will be elaborated according to the block diagram in introduction.

3.1 Front End Data Collection & Processing:

The front end data collection includes the emulation of the ball, recording from DVS128 camera, and further data processing. The objective was to verify the data in each stage corresponding to the ball motion and the output data from this functional block is compatible with later stage.

3.1.1 DVS128 Data Collection:

Before collecting the data from DVS128, the first task was to make the ball in motion stable and controllable. For this reason, the ball was emulated from the Matlab animation and DVS128 camera was placed 30cm away from the camera screen. The background colour in the animation is black, while the colour ball is set to white. In this setting, the position of the ball moving will trigger the "ON"event and the ball disappearing will trigger the "OFF"event in the FOV. With the jAER software tool from *initLab*, the data from DVS128 was recorded and exported in the AEDAT 2.0 format.

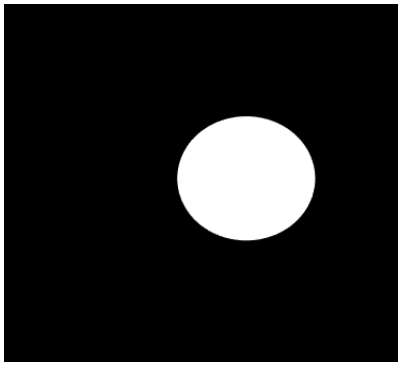


Figure 3.1.1 Matlab Ball Animation

In order to simplify the machine learning task on SpiNNaker, the size of the ball was kept at one third of the side length of FOV. The ball trajectory covers from one end of the FOV to the opposite side.

3.1.2 AEDAT Data Reformatting & Processing:

Since SpiNNaker platform is not compatible with AEDAT format so the information was extracted from the AEDAT file and exported to SpiNNaker. The AEDAT format was represented in ASCII, which was not desirable to be used in the later stage and not readable for verification. Therefore, the ASCII represented AEDAT file was converted to binary representation and AER data packet was then unpacked and separated with event coordinates, ON and OFF indication, and the event time stamp.

In order to relax the computation on SpiNNaker, the resolution of DVS128 is down-sampled from 128 by 128 to 16 by 16, which combines the 8 by 8 pixel block in the original resolution to a single pixel in the down-sampled resolution. In other words, relative luminosity change events within 8 by 8 pixel block triggered at the same time will combine into one event in the down-sampled resolution. Inevitably, downsampling will result in a loss information of input pattern, but it is safe for the neural network on SpiNNaker according to simulation from Li [29].

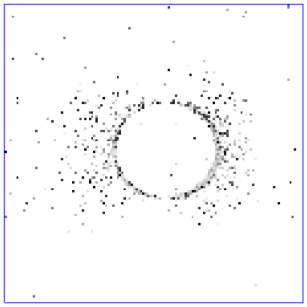


Figure 3.1.2 jAER Raw Data Caption

At this stage, down-sampled recording contained excessive amount of spiking events and the number of events varied from recording to recording. The background activity of spiking was essential in the leaky integrate & fire model. The membrane potential determined the post-synaptic neuron spiking condition. The more the spiking events in the pattern, the more frequent the post-synaptic neuron firing occurred. Therefore, the leaky integrate & fire model based neural network required a pattern in a different direction to be normalised.

3. Implementation

By using python library python-aer 0.1.3, it can extract, unpack, downsample, normalise and convert AEDAT file. The converted data was saved in Matlab data file. However, the function in python-aer 0.1.3 sometimes failed to extract data in AEDAT file, because jAER generated empty newline and spike data events with incorrect time stamps in the data stream. The spike data event always have a monotonically increasing time stamp, which was not the case in those incorrect data event. The third party jAER software in Java was difficult to modify, as the program code is long and lacks comments. Therefore, functions in python-aer library were modified to cope with these issues by skipping empty line and filtering out incorrect data event. The ball trajectories from one side of FOV to the other are normalised within 200ms time window, and the number of events are normalised to 88 for each direction.

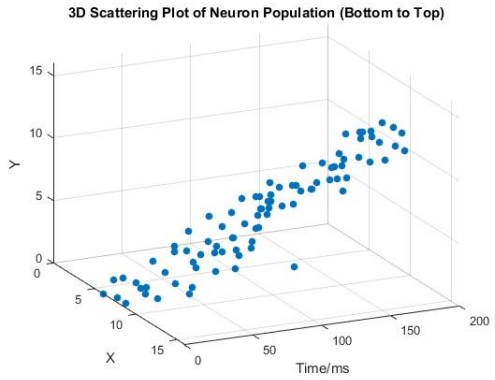


Figure 3.1.3 3D Scattering Plot of Normalised Spiking Event (Bottom to Top)

3.2 Neural Network Simulation on SpiNNaker:

This section of the report examines the behaviour of the neural network model on SpiNNaker, the verification on a two-layer neural network and two neural network topologies design. In machine learning tasks, there are normally two phase, learning period and post-learning period. The implementation chapter was mainly focusing on the learning period.

3.2.1 Behaviour of Neural Network Model on SpiNNaker:

Before discussing different neural network topologies, different neuron models and STDP learning rule were simulated on SpiNNaker to visualize their behaviour intuitively. A clear insight of these simulation results assisted in modelling the neural network topologies.

3.2.1.1 Neuron Model Simulation:

There were 2 types of variant of leaky integrate & fire model supported by SpiNNaker platform, including current based with exponential decay and conductance based with exponential decay.

Recalling the membrane potential equation (5) described in Chapter 2, introduction to the leaky integrate & fire model. In the SpiNNaker platform, the membrane potential equation for the current based model was used by neglecting the $V_m(t)$ in the synaptic current term. The new membrane potential equation can be expressed in equation (10). The deduction from equation (5) to equation (10) is in the appendix.

$$V_m(t) = -A_1 exp(-\frac{t}{\tau_m}) + A_2 exp(-\frac{t}{\tau_{syn}}) \tag{10}$$

To model the behaviour of the membrane potential in current based model, the post-synaptic neuron was connected to only one pre-synaptic neuron and the model parameter list is shown in the table below.

Parameter Name	Parameter Value
First Layer Neuron Population	1
First Layer Neuron Cell Type	SpikeSourceArray
Second Layer Neuron Cell Population	1
Second Layer Neuron Cell Type	Current Based LIF
Leakage Time Constant (τ_m)	$5.0ms$
Refractory Period (τ_{refrac})	$25.0ms$
Synaptic Delay	$1ms$
Membrane Capacitance (C_m)	$0.125nF$
STDP LTP Amplitude	0.02
STDP LTD Amplitude	0.01
STDP LTP Time Constant (τ_{LTP})	$10.0ms$
STDP LTD Time Constant (τ_{LTD})	$20.0ms$

Table 3.2.1 Parametric Settings in Leaky Integrate & Fire Model and STDP Learning Rule

From the membrane potential equation (8), the shape of membrane potential curve above can be characterised by leakage time constant,τ_m, synapse time constant, τ_{syn}, and the synapse strength between pre- and post-synaptic neuron, W. The membrane equation was simulated in the Matlab in figure 3.2.1, which matched with simulation on SpiNNaker.

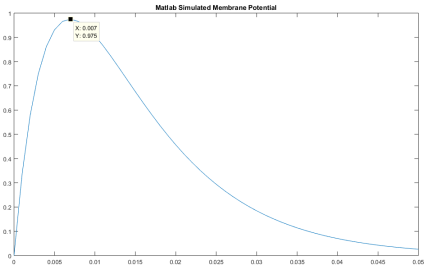


Figure 3.2.1.a Matlab Simulation of Membrane Potential Transient Response From One Pre-Synaptic Spike

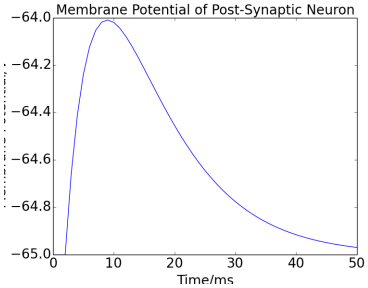


Figure 3.2.1.b SpiNNaker Simulation of Membrane Potential Transient Response From One Pre-Synaptic Spike

The conductance based model came from equation (3) but under different assumption. The conductance based model equation implemented on SpiNNaker is ambiguous, from the simulation result in figure 3.2.2. The comparison of current based and conductance based model on SpiNNaker was illustrated in the figure below with the excitatory reversal potential set at 0mV and inhibitory reversal potential set at -70mV.

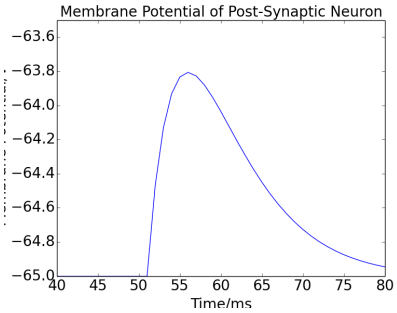


Figure 3.2.2.a Membrane Potential Transient of Current Based LIF Model Response From One Pre-Synaptic Spike

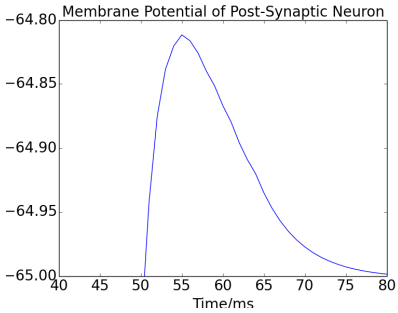


Figure 3.2.2.b Membrane Potential Transient of Conductance Based LIF ModelResponse From One Pre-Synaptic Spike

The shape of the membrane potential curve is very similar to the current based model, with an unnoticeable faster decay to the resting potential, V_{rest}. However, under the same setting, the peak magnitude is less than that for current based, and there was a 30ms delay between the rising time of post-synaptic neuron membrane potential and pre-synaptic neuron spiking time. In order to compare two models, a 30ms delay was added in the current based model.

3.2.1.2 STDP Learning Rule Simulation:

Having modelled different leaky integrate & fire model, the next focus was on the STDP learning rule. As mentioned in the background, the STDP consists of timing dependence rule and weight dependence rule.

The timing dependence rule can be described by equation (6) and (7) for LTP and LTD respectively. The STDP learning curve in figure 3.2.3 was simulated as in the STDP parameter table.

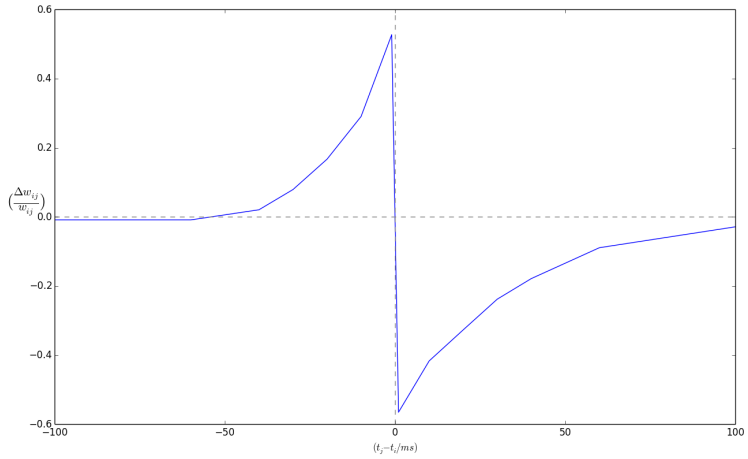


Figure 3.2.3 SpiNNaker Simulation of STDP Timing Dependence Rule

SpiNNaker also allows user specifying the weight dependence rule to either additive or multiplicative as in equation (8) and (9). There are other weight dependence rule supported by SpiNNaker but the most fundamental and important weight dependence are additive and multiplicative dependence rule.

3. Implementation

To illustrate the effect of weight dependence rule, a neural network was used with 256 pre-synaptic neuron connecting to 40 post-synaptic neuron, giving total number of 10240 synapses in between. The initial synapse weight was generated using a uniform distribution and input are ball motion recording from DVS128. In theory, the additive dependence rule pushes the synapse weight histogram to two extreme ends after training, while multiplicative dependence rule train synapse weights shape them as normal distribution.

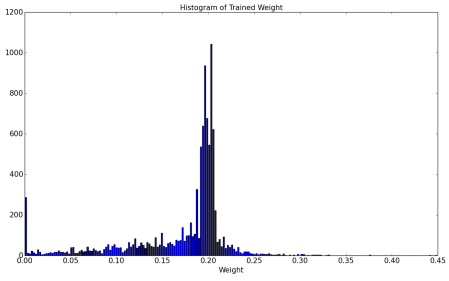


Figure 3.2.4.a Trained Synapse Weight Histogram From Additive Dependence Rule

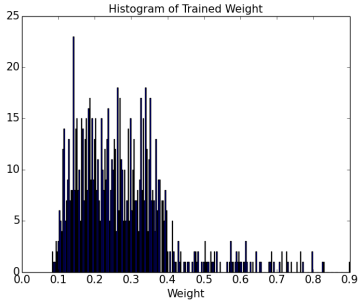


Figure 3.2.4.b Trained Synapse Weight Histogram From Multiplicative Dependence Rule

The additive dependence rule changes the synapse weight by fixed step, according to time dependence. Therefore, it has a faster training rate than multiplicative dependence. From the weight histogram, the additive dependence over-trained synapse weight, with significant amount of them decreased to 0. The reasons that there was a peak at the middle in the additive weight dependence rule trained synapse histogram were that input synapses were all initialised at 0.2 and some of these synapse never spike. On SpiNNaker, if there is no spiking for the synapse, the weight would not be affected by STDP learning rule. On the other hand, multiplicative dependence rule constrain the weight change, according to the difference between current quantity to maximum or minimum value.

3.2.2 Two-Layer Neural Network Topology Verification:

The two-layer neural network topology from Li[29] defined 256 pre-synaptic neuron as the first layer and 40 post-synaptic neuron in the second layer with excitatory synapse between pre- and post-synaptic neuron and inhibitory synapse between post-synaptic neuron. The purpose of simulating this topology was to verify the result and move forward based on this topology. The architecture of this two-layer neural network connectivity has been shown in the figure 3.2.5.

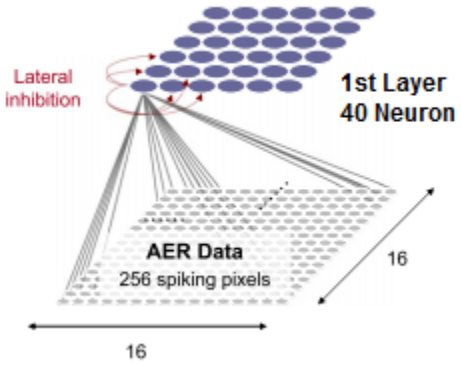


Figure 3.2.5 Architecture of Existing Two-Layer Neural Network Topology

In this topology, neuron in first layer were of type spike array source, which generated spikes according to the DVS input. In the second layer, the model was current based leaky integrate & fire model. For the STDP learning rule, the weight dependence rule was additive. The parameter list of leaky integrate & fire model and STDP learning rule is in table 3.2.2.

Parameter Name	Parameter Value
First Layer Neuron Population	256
First Layer Neuron Cell Type	SpikeSourceArray
Second Layer Neuron Cell Population	40
Second Layer Neuron Cell Type	Current Based LIF
Leakage Time Constant (τ_m)	$7.0ms$
Refractory Period (τ_{refrac})	$25.0ms$
Synaptic Delay	$1.0ms$
Membrane Capacitance (C_m)	$0.3nF$
STDP LTP Amplitude	$15.0ms$
STDP LTD Amplitude	$25.0ms$
STDP LTP Time Constant (τ_{LTP})	0.012
STDP LTD Time Constant (τ_{LTD})	0.01

Table 3.2.2 Parametric Settings in Leaky Integrate & Fire Model and STDP Learning Rule of Existing Two-Layer Topology

The ball trajectories were randomly selected 200 times in 4 directions during training period. Secondly, the generation of initial synapse weight was from a normal distribution.

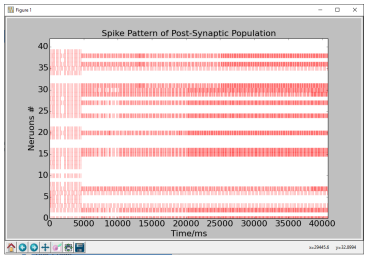


Figure 3.2.6.a Post-Synaptic Neuron Spiking When Training

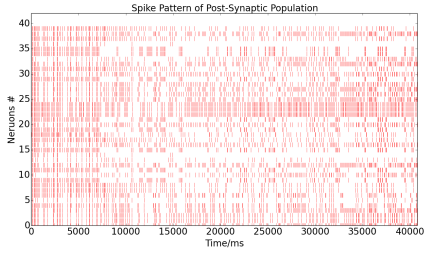


Figure 3.2.6.b Post-Synaptic Neuron Spiking After Training

With several times of training, similar post-synaptic neuron spiking pattern is obtained. However, in most of the training result, the firing pattern in one direction cannot be distinguished from the other, which means that this topology cannot be used directly.

Despite of the result of post-synaptic neuron firing pattern, the topology has over-represented the ball trajectory, as it doesn't hold a surjective relationship between input ball trajectory set and output neuron set. That is, a ball trajectory, not in training set, can have a different post-synaptic neuron firing pattern after training, which cannot be classified as the same direction. For this reason, there were two solutions to tackle this problem. The first option was by adding an extra layer on top, with 4 neuron only in the third layer. The second option was relied on modification of the model parameters and change the neuron population in second layer to 4, instead of 40.

3.2.3 Neural Network Topologies:

From the evaluation in previous topology, input ball trajectory set and the number of neuron in the output layer set needs to be surjective. There are two solutions to tackle this problem, by adding another layer to discriminate the pattern further or modifying the parameters in the model. However, tuning the parameters in the model was a great challenge, because the model is a multifactor non-linear system. This section conveys the simulation and analysis of these two topologies. The simulation result from both topologies have proven that the second topology will be suitable for this system.

3.2.3.1 Three-Layer Topology:

The three-layer topology included input layer, intermediate layer, and output layer. Since the resolution of DVS input has been downsampled to 16 by 16, the neuron population in the input layer has to be 256. There were only 4 ball trajectories including left, right, upwards, and downwards. These direction will be referred as basic direction, and 9 different data sets were collected in each of these basic direction and processed to train the system. The number of neuron population in output layer was set to 4. However, the number of neuron population in the intermediate layer and other parameters were varied in different trials, which will be discussed in the following section.

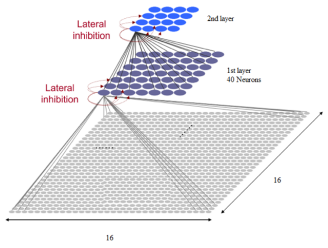


Figure 3.2.7 Architecture of Three-Layer Neural Network Topology

3.2.3.1.1 Input Layer to Intermediate Layer:

Although there were problems in the previous two-layer topology, the configuration was biological plausible. Therefore, the first trial was to tackle the problems in the topology by modifying the parametric settings in the previous two-layer topology. As the model is non-linear and multi-factor, the methodology of tackling the problems was by treating the system as a black box initially and gradually tune parameters for correct response.

Recall the effect of lateral inhibition in background and the simulation earlier on. It imposes a negative current spike to other connected neuron, which naturally prevent other neuron spike and learn from same input pattern. Therefore, weight and time constant of inhibitory current were increased to 1.5 and 20ms. Other parameters were also changed to compensate, which results in approximately 10 post-synaptic neuron firing after training.

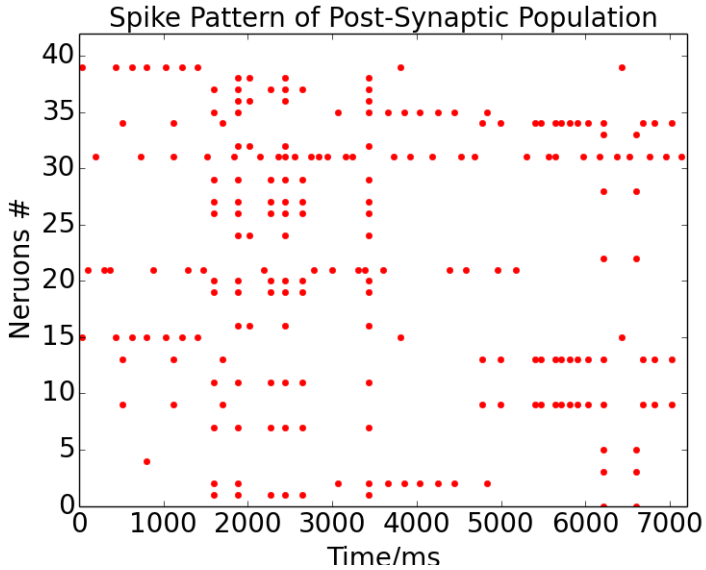


Figure 3.2.8 Post-Synaptic Neuron Spiking After Training

Based on this setting, an expected post-synaptic neuron spiking pattern was obtained. Although the post-synaptic neuron firing pattern was distinguishable for different ball trajectory, an observation on a 2D synapse weight plot showed that the synapses trained were undesirable. First of all, synapses were over-trained by the additive dependence in STDP. Secondly, different post-synaptic neuron can also learn same pattern, even though there were inhibitory connections between neuron in the second layer. Thirdly, one post-synaptic neuron was learning patterns from different ball trajectories. All three of these problem needed to be tackled, in order to build a robust network.

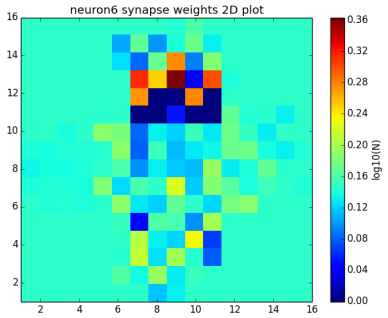


Figure 3.2.9.a Over-Trained Weight Distribution

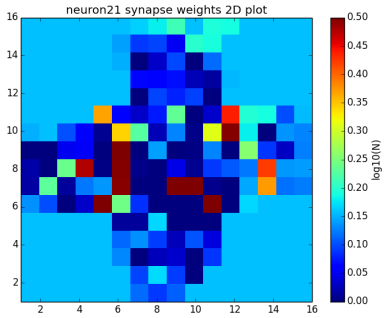


Figure 3.2.9.b Multiple Pattern Learnt Weight Distribution

The solution to the first problem came from property of STDP additive and multiplicative dependence rule. Recall the simulation in section 3.2.1.2. The additive dependence rule tends to pushes the synapse weight to two extreme ends, and hence, over train synapse weight. On the contrary, the multiplicative tries to shapes the trained synapse weight to a Gaussian distribution. Therefore, the multiplicative dependence would be more suitable under this circumstance.

The second problem was vulnerable to the system, meaning increasing in the complexity of neural network with more post-synaptic neuron would not make the system learn sufficient features from the input patterns. Since this problem remained after changing parameters in lateral inhibitory, other important factors can affect the learning outcome as well as lateral inhibition. Principally, lateral inhibition inject negative current to all other post-synaptic neuron. Nevertheless, if multiple post-synaptic neuron spike at the same time, lateral inhibition can have no effect to discriminate these post-synaptic neuron. Consequently, those neuron firing at the same time would learn same pattern from STDP. The post-synaptic neuron spiking was related to the integration from each synaptic current. Hence, the initialization of the synapse weight and the input

pattern determines the convergence of the post-synaptic neuron. Since the synapse weight were initialised by the normal distribution, most post-synaptic neuron would learn the same pattern after training. Hence, this neural network model requires the initial synapse weight as different as possible. Thereby, a uniform distribution was a better solution than the normal distribution.

The learning for post-synaptic neuron occurs, whenever they spike. In other words, the post-synaptic neuron can respond and learn, if the synapse weights are sufficiently high to spike when stimulated by different patterns. A complex neural network, such as the real neural network in the brain, might be capable of coping with neuron learning different features by sorting spiking representing same information. However, in the ANN model, this situation is better to be avoided as much as possible to reduce the complexity of model structure. In this sense, the post-synaptic neuron can only be initialised to be sensitive to specific patterns.

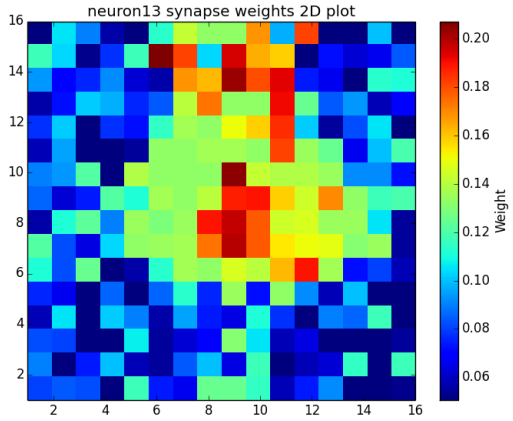


Figure 3.2.10 Trained Weight Distribution When Shuffling Input Trajectory

In the real neural network, the neuron are also connected to finite number of neighbouring neuron. For these reasons, the FOV was divided into 9 smaller square area. The neuron population in the intermediate layer was set to 45 with 5 neuron in a group learning patterns in each area. Synapse of neuron in each group were initialised relatively high in one area and low in all other areas. All 45 neuron in the intermediate layer would cover the whole FOV.

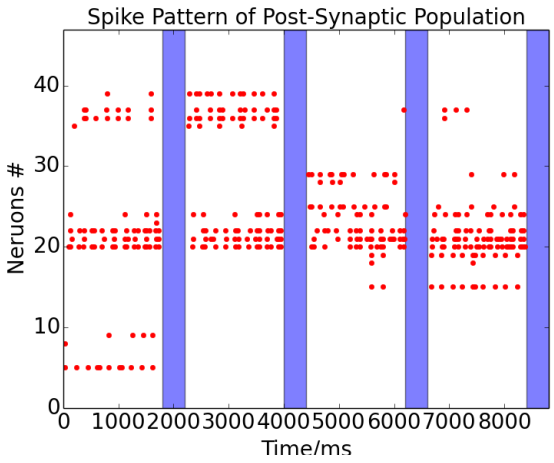


Figure 3.2.11 Post-Synaptic Neuron Spiking Pattern After Training

With this configuration, the post-synaptic neuron spiking pattern were well separated in figure 3.2.11. Each neuron group provided a clear spatial representation of the visual input. However, the neural network lost its fault tolerance ability, since there was not sufficient learning from STDP. Moreover, the most important outcome is that the spatial temporal representation of the input pattern after learning. For this reason, this topology was abandoned and moved on to a new two-layer topology with spatial temporal representation.

3.2.3.1.2 Intermediate Layer to Output Layer:

Although the topology was abandoned, it is possible to achieve the final objective using previous configuration in first two layers and a polychronization network between the intermediate layer and third layer.

The input ball trajectories were all normalized in 200ms in the pre-processing stage. When neuron in output layer connecting to the neuron in intermediate layer, the synapse delay can be initialised linearly decreasing from one side of the FOV to the other side. Only the contribution from entire spiking sequence in the intermediate layer converged at the end can cause the neuron in output layer spike. Thus, the polychronization between intermediate layer and output layer provides a temporal representation on top of the spatial representation in first two-layers. However, as stated earlier, this topology lacks learning process, for which there was no further simulation on this topology.

3.2.3.2 two-layer Topology:

The final design eventually settled to a two-layer topology with 4 neuron in the output layer connecting to 256 neuron in the input layer. After learning, each neuron learnt a spatial temporal representation of pattern in different ball trajectory.

Unlike previous configuration, the neuron in the output layer need to learn the features from entire ball trajectory spiking pattern, instead of distributing the learning outcome to different neuron. To achieve this goal, the synapse weight were weakened, so that the accumulation of membrane potential by approximately two thirds of the spiking event in one ball trajectory could trigger the post-synaptic neuron spiking. The synapses were initialised by a uniform distribution. In order to accumulates the membrane potential within 200ms, the leakage current time constant was set to 110ms. Consequently, the membrane potential decays exponentially over a longer period. The initial parametric list was in the table 3.2.3 below.

Parameter Name	Parameter Value
First Layer Neuron Population	256
First Layer Neuron Cell Type	SpikeSourceArray
Second Layer Neuron Cell Population	4
Second Layer Neuron Cell Type	Current Based LIF
Leakage Time Constant (τ_m)	$110.0ms$
Refractory Period (τ_{refrac})	$40.0ms$
Excitatory Time Constant (τ_{syn_E})	$5.0ms$
Inhibitory Time Constant (τ_{syn_I})	$10.0ms$
Synaptic Delay	$1.0ms$
Membrane Capacitance (C_m)	$12nF$
STDP LTP Amplitude	0.05
STDP LTD Amplitude	0.05
STDP LTP Time Constant (τ_{LTP})	$50.0ms$
STDP LTD Time Constant (τ_{LTD})	$60.0ms$

Table 3.2.3 Parametric Settings in Leaky Integrate & Fire Model and STDP Learning Rule of two-layer Topology

The inhibitory synapse weight was set to 6, so that one post-synaptic neuron firing would bring the membrane potential of all other post-synaptic neuron back to V_{rest}. The model was initially tested by one post-synaptic neuron at the output layer and trajectories in only one direction were imported in the system. By plotting the 2D weight distribution, the STDP learning rule tends to shift the peak of synapse weight earlier in time. However, the ball trajectory set was imported continuously, it caused an "overflow"effect, shifting the peak of synapse weight back to the end of the sequence. Therefore, the synapse weight cannot converge. The solution to this "overflow"effect was by adding a 200ms silent time period between each ball trajectory sequence. In this way, the spiking events at the end of sequence would not affect and shift the peak of synapse weight, as its contribution would decay to V_{rest} in the silent period.

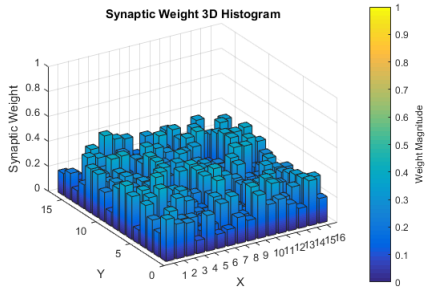


Figure 3.2.12.a Initial Weight Distribution

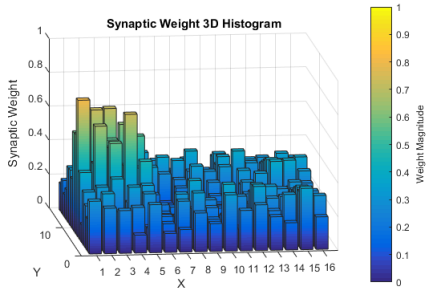


Figure 3.2.12.b Trained For Left to Right Direction Weight Distribution

The network failed in training 4 output neuron in all 4 directions, as the ball trajectory was randomly selected from all basic ball trajectories. Recall the STDP learning curve, the LTP and LTD exponentially decays relative to the onset of post-synaptic neuron spiking. It enhances those synapses with closest pre-synaptic spiking before the post-synaptic spiking, and vice versa. However, if the synapse weight does not converge to the beginning of the sequence and experiences an opposite spiking pattern, the post-synaptic neuron will be more sensitive to the reversed spiking pattern. It is obvious that integrating synapse weight reversely in LTP period can make membrane potential approaching V_{th} faster than integrating synapse weight forward in LTP period. In order to train 4 post-synaptic neuron in output layer together, it needs to import the ball trajectory repeatedly until one of the post-synaptic neuron converges. Once the synapse weight of the post-synaptic neuron has fully converged, the ball trajectories in next direction can be imported.

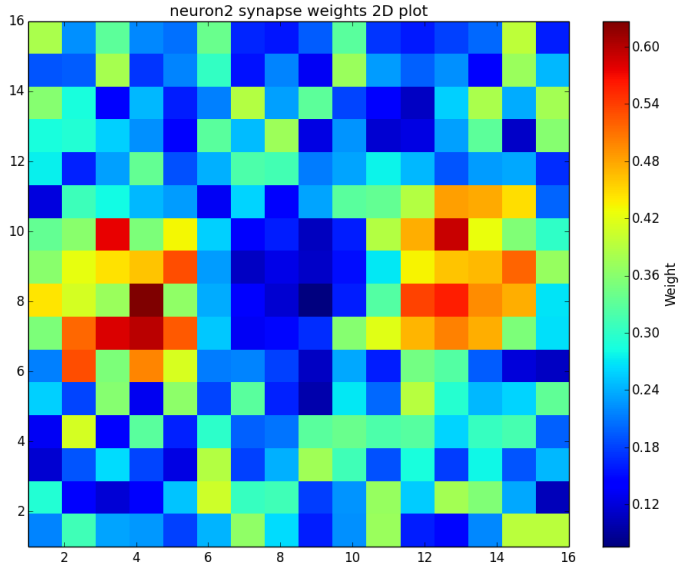


Figure 3.2.13 Trained Weight Distribution of An Output Neuron Learnt Two Direction

With this configuration, there was still one neuron in post-synaptic neuron learning two or more ball trajectory patterns, because the average synapse weight for trained output neuron was higher than that of untrained output neuron. The synapse of these neuron were identical to when it's initialised. Therefore, the range of initial synapse weight for post-synaptic neuron was increased.

After changing the initial synapse weight, 4 output neuron can learn ball trajectory patterns separately.

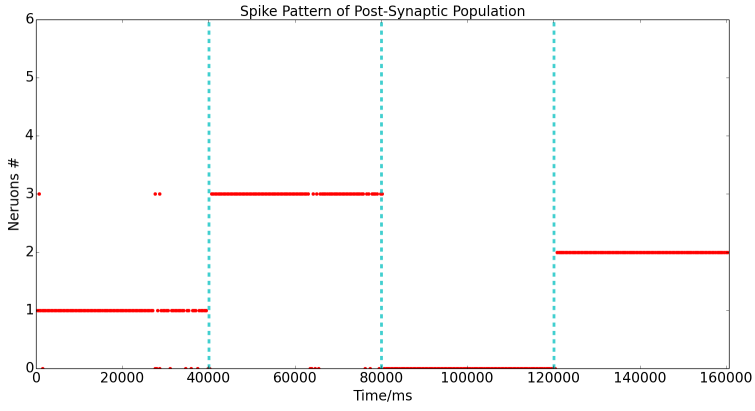


Figure 3.2.14 Post-Synaptic Neuron Spiking Pattern In Training Period

Table 3.2.4 summarises the direction of each output neuron learnt in the training period.

Neuron ID	Direction Learnt
0	Upwards
1	Left
2	Downwards
3	Right

Table 3.2.4 Output Neuron Direction

4. Testing

After training the final two-layer neural network model, there were 8 different tests measuring and anticipating system robustness and accuracy for the ball catching task. Apart from the first test verifying the direction indication ability of the trained network, the rest of tests were designed for measuring the tolerance and accuracy of network. Each tests consist of its testing purpose, testing plan, and estimation of testing result.

In order to minimise the distortion from consecutive trajectories, 200ms silent period were inserted between each trajectory as it was in training period. There were 3 sets of data in each test, unless explicitly claim a different number of data sets were used.

4.1 Post-Synaptic Firing Pattern Verification:

From the previous chapter when the network model was trained, each post-synaptic neuron in the output layer learnt the pattern of one trajectory. The initial test was to verify that the system has the ability to indicate the direction of ball motion.

There were two parts in this verification test, which tested the system by training data sets, as well as extra testing data sets in basic directions. The number of data sets in these two tests were 9 and 6 respectively. The data sets were imported in the order of left, right, upwards, and downwards.

4.2 Angle Variation Test:

Two test cases varying the angle of trajectory were stated below. However, the two angle variation tests were of different purpose. The first one was focused on determining significance of the beginning of the input sequence in the spatial temporal representation of the output neuron. The purpose of second test was to measure the tolerance and ability in correcting indication with an angle variation of trajectory at centre of FOV.

4.2.1 Same Starting Position:

In the first testing case, it varies the angle of trajectories with a pivot at the beginning of the sequence. The angle variation ranged from 10 degree to 80 degree with a step size of 10 degree. All 4 output neuron were tested individually.

From the property of STDP learning rule, the spiking of trained output neuron depends on the beginning of the input sequence. With ball trajectory all having same starting positions, the output neuron can be insensitive to the angle variation in this case.

4.2.2 Crossing Centre of FOV:

In the second test, the pivot of trajectory was at the centre of FOV. The range of angle variation was from 0 to 350 degree. Similarly, the angular difference in each step was 10 degree. In this case, 0 degree refers to the ball moving from left to right.

Since the network was trained only by trajectories in 4 directions, the post-synaptic neuron in the output layer would spike, if a ball is moving in the directions corresponding to the ones from learning process. The ball trajectories with certain angular difference can also possibly cause the same output neuron spiking, with sufficient enough spiking event.

4.3 Ball Trajectory Shifting Test:

Previous two tests was designed for the accuracy and tolerance when changing the angle of trajectory, the next test was of the same purpose, but the variable became the shifted distance of ball trajectory.

Specifically, the ball trajectory were shifted horizontally when ball was moving left or right. Similarly, the shifting was vertically, if a ball was moving upwards and downwards. The diameter of the ball was set to be approximately one third of the length of FOV. The step of shifted distance accords with a quarter of ball radius, ranging from 1/4 of ball radius to 5/4 of ball radius. All 4 output neuron were tested individually in this test. With the prerequisite that ball trajectory and trained synapse are symmetrical, the shifting went further away in one direction only.

As the ball trajectory shifting further away, it overlaps with the adjacent ball trajectory. For instances, the trajectory of a ball moving upwards shifted horizontally by certain distance will overlap with the trajectory of a ball moving from right to left in the FOV. The output neuron sensitive to right to left movement can possibly spike in this case. This test can show the significance of STDP learning outcome leaning on temporal information or spatial information.

4.4 Speed Test:

When a ball is moving in the real world, the speed of a moving ball does not necessarily remain the same. Variation in ball speed will change the number of spiking in the trajectory and also shorten the period of ball trajectory. Two tests were planned specifically for measuring the influence of changing speed. Although the neural network was not trained by ball moving in different speed, it is still worth investigating the ability of trained neural network coping with different ball speed.

4.4.1 Number of Spiking in Ball Trajectory:

In the first speed test, the variable was the number of spiking in the ball trajectory. After normalisation from raw AEDAT recording, there were 88 spiking events within 200ms period for each ball trajectory. Therefore, the lower boundary of the spiking number was set to half of that in the training data sets, corresponding to 44 spiking in one trajectory. The number of spiking was increased by 11, up to 132 spiking in one trajectory. The test was implemented on the trained output neuron separately to determine the behaviour of each neuron.

The trained output neuron can only fire by enough correlated stimuli from input pattern. In estimating testing outcome, a higher number of spiking would trigger the corresponding output neuron in a fast response.

4.4.2 Period of Ball Trajectory:

The second speed test varied the active period of the ball trajectory. In all testing cases, the window of each trajectory was 400ms, with 200ms active period and 200ms silent period. There were input spiking in the active period. On the contrary, there would be no spiking in the silent period.

In this test, it varied the length of active period. The lower boundary and upper boundary of active period in this test were 100ms and 250ms respectively. A 25ms step size was set for each testing value. As the test required normalising active period differently, there was only one data set for each test value.

The spiking concentration is denser in a shorter active period than that in a longer active period. The post-synaptic neuron fires based on its membrane potential, and a higher spiking concentration can hinder the leakage of membrane potential from individual pre-synaptic spiking. The estimation for this test can be addressed that the shorter the active period is, the easier the output neuron can spike.

4.5 Ball Size Test:

In reality, not only the speed of the ball varies, but also physical size of ball as well as the relative size in the camera FOV changes. Hence, the next investigation was on testing the influence of varying size of ball.

The size of ball was controlled by changing the ball radius. The ball radius varied from half of that in training set to 1.5 times larger than that in the training set. The step between each test value was one tenth of the ball radius in training set. Although all four output neuron were tested, there was only one data set.

Although the size of the ball varied, the number of spiking in each trajectory were still normalised to 88. Therefore, it was difficult to predict the outcome for this particular test.

4.6 Background Noise Activity Test:

The last testing case measures immunity and tolerance of trained network against background noise. The ball catching task was simplified by generating the ball trajectory from Matlab animation at this stage. In reality, a ball can potentially be any colour and the ambient luminance level varies, which increases the level of the background noise, therefore, it is worth determining the effect from the background noise.

4. Testing

With the total number of spiking number being 88, the number of noise spiking was increased by 11 between adjacent testing values. The range of spiking number was from 11 to 198.

Certain number of noise spiking can help the network capturing those ball trajectory lacking enough relevant spiking. However, if the number of noise spiking is too high, output neuron might not be able to discriminate the relevant spiking in the trajectory and distinguish from one direction to the other.

5. Results:

In this chapters, it refines and summarises the testing result in previous chapter. With the refined result, a detailed analysis for each scenario will then be discussed.

5.1 Post-Synaptic Firing Pattern Verification Result:

From the verification result, the post-synaptic neuron firing pattern simulated from training data sets and extra testing data sets recorded under the same situation both match with the firing pattern in training period. It proved that trained network is able to indicate the direction of trajectory.

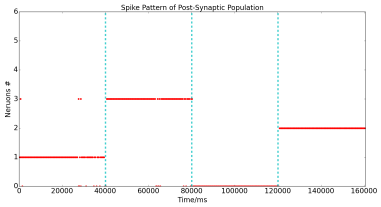


Figure 5.1.1.a Post-Synaptic Neuron Spiking After Training By Training Data Set

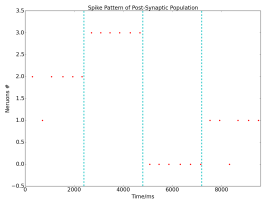


Figure 5.1.1.b Post-Synaptic Neuron Spiking After Training By Extra Testing Data Set

Although the number of data set was insufficient, it has shown the high accuracy of the trained network. When testing using the training data set, there were more missing indication than false indication. However, it was more likely to have false indication in the testing data sets.

	Training Data Set				Testing Data Set			
	Left	Right	Upwards	Downwards	Left	Right	Upwards	Downwards
Correct Indication	8/9	8/9	9/9	8/9	5/6	6/6	6/6	5/6
False Indication	1/9	0/9	0/9	0/9	1/6	0/6	0/6	1/6
Missing Indication	0/9	1/9	0/9	1/9	0/6	0/6	0/6	0/6

Table 5.1 Table of Post-Synaptic Firing Pattern Verification

5.2 Angle Variation Test 1: Same Starting Position

With ball trajectories having a pivot at the beginning, the test gave a quantitative measurement on the significance of the beginning of sequence with respect to the entire input pattern. The post-synaptic neuron spiking pattern in each test cases were plotted and included in the Appendix.

	Test Cases															
	Left				Right				Upwards				Downwards			
Angle	L	R	U	D	L	R	U	D	L	R	U	D	L	R	U	D
0	2	0	1	0	3	0	0	0	0	0	2	1	0	0	0	3
10	3	0	0	0	3	0	0	0	0	1	2	0	0	0	0	3
20	2	0	2	0	3	0	0	0	0	0	3	0	0	0	0	3
30	3	0	0	0	3	0	0	0	0	0	3	0	0	0	0	3
40	3	0	0	0	3	0	0	0	0	0	3	0	0	0	0	3
50	3	0	0	0	3	0	0	0	0	0	3	0	0	0	0	3
60	2	1	0	0	3	0	0	0	0	0	3	0	0	0	0	3
70	0	1	1	2	1	2	0	0	0	0	3	0	1	0	0	2
80	3	0	0	0	0	3	0	0	1	0	2	0	0	0	0	3

Table 5.2 Table of Angle Variation Test 1: Same Starting Position

The table above shows the spiking rate of output neuron in each test case. The direction in the second row are the reference directions, where the variation is 0. The initials in the next row represents the direction of output neuron. The value in each cell represents the number of corresponding output neuron spiking for a trajectory with an angular difference with respect to the reference direction.

From the table, the spiking of output neuron did not differ much when varying the angle in this test. The trained network was insensitive to angle variation with a same starting position.

5.3 Angle Variation Test 2: Crossing Centre of FOV

Recall the purpose of second test, which measures the tolerance and ability in correcting indication with an angle variation of trajectory at centre of FOV. The post-synaptic neuron spiking pattern were plotted. From this spiking pattern, the range of angle variation, each of which output neuron can tolerate or correct, can be extracted. As there were 3 data sets in each angle from 0 degree to 350, the angle would be sorted to an output neuron, if the number of spiking is more than 2.

	Lower Boundary	Upper Boundary	Range
Left	-30	30	70
Upwards	40	140	110
Right	150	210	70
Downwards	220	300	90

Table 5.3 Table of Angle Variation Test 2: Crossing Centre of FOV

The range of each output neuron can tolerate and correct are about a quarter of the revolution, although they differ from each other slightly. However, a large tolerance also means a low accuracy of angle variation.

5.4 Ball Trajectory Shifting Test Result:

Similar to the angle variation test 1, there were 4 testing cases in each basic direction with post-synaptic neuron spiking pattern plotted separately. Those post-synaptic neuron spiking pattern were included in the appendix, but the table below shows a summary of result.

	Test Cases															
	Left				Right				Upwards				Downwards			
Distance	L	R	U	D	L	R	U	D	L	R	U	D	L	R	U	D
r/4	3	1	0	0	0	3	0	1	0	0	4	0	0	0	0	4
2r/4	2	0	0	2	0	3	0	1	0	1	3	0	0	0	0	4
3r/4	2	0	0	2	0	3	0	1	0	4	0	0	0	0	0	4
4r/4	0	0	0	4	0	2	0	2	0	3	1	0	0	2	0	2
5r/4	0	0	0	4	0	0	0	4	0	3	1	0	1	3	0	0

Table 5.4 Table of Ball Trajectory Shifting Test

5. Results:

From the table, output neuron have different immunity in shifting trajectory test. The neuron with best immunity was the one learnt trajectory moving downwards with a maximum shift of three quarters of ball radius. On the contrary, the output neuron learnt ball trajectory moving from left to right cannot determine direction after a half ball radius shift.

5.5 Speed Test 1: Number of Spiking of Ball Trajectory

The first speed test investigate the relationship of the variation in the number of spiking and the accuracy of the direction indication. A table summarising the result of test from 4 different post-synaptic neuron spiking pattern was illustrated below.

Number of Spikes	Test Cases											
	Left			Right			Upwards			Downwards		
	C	F	M	C	F	M	C	F	M	C	F	M
44	0	0	6	0	0	6	0	0	6	0	0	6
55	0	0	6	0	1	5	0	2	4	0	5	1
66	0	5	1	3	1	2	5	2	0	1	4	1
77	2	4	0	5	0	1	6	0	0	4	2	0
88	3	3	0	6	0	0	6	0	0	6	0	0
99	6	0	0	6	0	0	6	0	0	6	0	0
110	6	0	0	6	0	0	6	0	0	6	0	0
121	6	2	0	6	0	0	6	0	0	6	1	0
132	6	2	0	6	2	0	6	2	0	6	2	0

Table 5.5 Speed Test 1: Number of Spiking of Ball Trajectory

In the table, column C, F, and M in each direction represents the correct indication, false indication, and missing indication respectively.

According to the table, the output neuron require at least the same number of spiking in the trajectory to give an accurate indication. A slightly higher number of spiking can even improve the result. However, if the number of spiking is too high, one trajectory can cause multiple output neuron spiking, which confuses the network.

5.6 Speed Test 2: Period of Ball Trajectory

In the previous chapter, the second speed test was designed for investigating the influence of active period within a 400ms window for each ball trajectory.

Active Period	Test Cases											
	Left			Right			Upwards			Downwards		
	C	F	M	C	F	M	C	F	M	C	F	M
100	6	0	0	6	0	0	6	0	0	6	0	0
125	5	1	0	6	0	0	6	0	0	6	0	0
150	5	1	0	6	0	0	6	0	0	6	0	0
175	5	1	0	6	0	0	6	0	0	6	0	0
200	5	1	0	6	0	0	6	0	0	5	1	0
225	4	3	0	6	0	0	6	0	0	5	2	0
250	4	3	0	6	0	0	6	0	0	3	3	0

Table 5.6 Speed Test 2: Period of Ball Trajectory

Again, column C, F, and M in each direction represents the correct indication, false indication, and missing indication respectively.

The active period in training data set was 200ms, while the range of active period varies from 100ms to 250ms. From the table, it is obvious that the shorter the active period was, the accurate the neural network can be.

5.7 Ball Size Test Result:

In this section, it illustrated the result of ball size test. The table below was summarised from the post-synaptic neuron spiking pattern in the output layer.

Radius	Test Cases							
	Left		Right		Upwards		Downwards	
	Correct	False	Correct	False	Correct	False	Correct	False
0.5R	√		√		√		√	
0.6R	√		√		√		√	
0.7R	√		√		√		√	
0.8R	√		√		√		√	
0.9R	√		√			√	√	
1.0R	√		√		√		√	
1.1R	√			√	√		√	
1.2R	√			√	√		√	
1.3R	√			√	√			√
1.4R	√		√		√		√	
1.5R		√		√	√		√	

Table 5.7 Ball Size Test Result:

From the table summarised from the spiking pattern, 3 out of 4 output neuron have managed to indicate the direction of trajectory from 0.5R to 1.5R. In the second test case, there was a high false indication from 1.1R to 1.5R, possibly resulting from lack of data set or not enough relevant spiking due to a wider distribution of larger size ball.

5.8 Background Noise Activity Test Result:

The last test was focused on the distortion to the trained neural network from the background noise. A table was summarised based from the post-synaptic neuron spiking. However, the range of this test was considerably large, the table was included in the appendix.

As a result, the network can tolerate up to 50% of noise spiking, with respect to the number of spiking of trajectory. When the number of noise spiking was 1.5 times larger than that of trajectory, the network can still indicate whether the ball is moving horizontally or vertically in the FOV. From this point onwards, the indication on direction would be completely distorted by noise.

6. Evaluation:

The evaluation chapter will be elaborated, according to the objectives for the ball catching task and simulation result in the previous chapter. As the implementation of data collection from DVS128 and processing later were based on the machine learning task on SpiNNaker platform, the elaboration for these parts would be discussed while evaluating the performance of trained network on SpiNNaker.

From the result in all testing cases, the trained two-layer neural network is able to determine the direction of trajectories. The second angle variation shows that it can indicate the direction of ball, if the ball is moving from one side of FOV towards the opposite side. This behaviour was expected and desirable, because the input ball trajectories were recorded so.

However, performance varies under different circumstances. In the first angle variation test, output neuron had difficulty in distinguishing direction of trajectories, even though the angle variation was large. Theoretically, the output neuron would spike less, as the angle variation increases. This phenomena can reflect the fact that synapse weight corresponding to spiking in the beginning of the sequence were over-trained. The result from ball shifting test also validated the over-training of synapse weight, otherwise shifting trajectory cannot cause spiking of output neuron learning adjacent direction. Hence, the synapse weight corresponding to the beginning of the trajectory needs to be weakened such that a spiking from output neuron requires more spiking, which increases sensitivity of network.

One of the advantage of neural network is that it can tolerate and correct fault so that the system is capable of recognising and interpreting same input in slightly different situation. For this ball catching task, the speed and size of ball as well as noise level operating in the environment vary in different situation.

A variation of ball speed, consequently, changes the number of spiking as well as the period of ball trajectory. Specifically, increasing speed will increase the number of spiking and shorten the period of ball trajectory, and vice versa. Both speed test results show that a ball moving in faster speed up to certain point would make it more detectable and accurate for the trained network. On the contrary, a slower ball increases the error in performing the same task.

6. Evaluation:

From the ball size test, it illustrated the fact that the trained network can determine the direction of ball trajectory irrespective to the size of ball, which is a desirable functionality in the ball catching task. However, this feature results from the over-trained synapses, which means the trained network is not accurate in determining the shape of objects. Not only the network needs to weaken the synapse weight but also it needs to be trained by different size of ball moving in different motion to actually cope with this problem. The training data sets have lost information of ball shape significantly. In order to train a more reliable network, the number of spiking in the trajectory needs to be increased as well.

From the background noise result, the trained network is noise immune up to 50% noise spiking with respect to the total number of spiking in the trajectory. In order to improve the noise immunity of the system, it needs to increases the relative potential difference between threshold voltage and rest potential in the leaky integrate & fire model. The relative potential can be measured as the ratio of actual potential difference relative to the average potential change from the pre-synaptic neuron. Alternatively, the neural network decrease the synapse weight for those irrelevant to the ball trajectory. This could be done by increasing the number of noise spiking in training. It is obvious that there would be a limit in increasing the number of spiking and also it leads to a longer training period.

7. Conclusion:

The designed two-layer neural network topology has met the objective of this project, which requires developing a system to determine direction of a moving ball. In the evaluation chapter, it illustrated that this topology can perform the task within limitation in different situation.

The trained network is accurate in indicating the trajectory that a ball moving from one side to the opposite side, but insensitive to those having the same starting position. It can also tolerate if ball trajectory gets shifted. The performance will better and more accurate for a ball moving in a faster speed. The system also has a good tolerance in noise, up to 50% of spiking in the training set.

Although the STDP learning rule deployed the multiplicative dependence rule, the synapse for each output neuron were over-trained, which caused undesirable behaviour in the tests, including varying angle with same starting position, shifting ball trajectory, and even varying the ball size.

In order to deal with those undesirable behaviour, the parameters in the network needs to be tuned slightly. Firstly, the intial synapses needs to be weakened further. Secondly, the number of spiking in each trajectory needs to be normalised with more spiking events to preserve more information of moving ball. Thirdly, the output neuron should be trained in various situation, for instances, different speed, size, or noise level.

8. Future Work:

At this point, the project has a completed parts for the robotic system, though it requires improvement in each building blocks to make the robotic to system perform the ball catching task in reality. The work to be done in the future can be listed in hardware and software separately.

8.1 Hardware:

1. At the moment, the ball trajectory were pre-recorded and imported from the PC to SpiNNaker platform. In order to implement the ball catching task, the system has to import and process the data from DVS128 in real time. Therefore, it requires an interface between DVS128 and SpiNNaker platform, as they are not directly compatible with each other.

2. In reality, it might need more information for tracking the ball or processing power for SpiNNaker board. Therefore, depending on the growth of complexity of system, there might be a demand in having a latest version of DVS camera or SpiNNaker platform.

8.2 Software:

1. The data from the DVS128 were heavily compressed in the down sampling and normalisation process. The characteristic and information of the ball and the trajectory can be lost, which can limit the performance of neural network on SpiNNaker platform, as the complexity of system increases later. Therefore, it is important to determine the optimum number of spiking in one trajectory, as project continues.

2. The neural network needs to be tuned slightly so that relevant synapse weights for output neuron will not over-trained. The polychronization can be added on top of the existing two-layer topology, which emphasis the spiking not at the beginning of trajectory. It can then possibly cope with testing cases discussed in the evaluation chapter and also characterise the entire ball trajectory.

3. The system can be more robust, if there are more output neuron learning different input patterns or alternatively making each of them learning the same patterns with different configuration, such as varying the speed and size of the ball, as well as the noise level.

9. Bibliography:

References

[1] SpiNNaker home page. University of Manchester. Available at: http://www.apt.cs.man.ac.uk/projects/SpiNNaker/

[2] H. S. Leow1, and K. Nikolic,"Machine Vision Using Combined Frame-based and Event-based Vision Sensor," in 2015 IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, 2015

[3] Z.Gu, I.Yamamoto, el al. "Development of forceps robot for surgical operation by biomechanism application," In 2011 4th International Conference on Biomedical Engineering and Informatics (BMEI), 2011

[4] J.Saleh, F.Karray, et al "A Qualitative Evaluation Criterion for Human-Robot Interaction System in Achieving Collective Tasks," in Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference, 2012

[5] M.Chemnitz,G.Schreck, et al "Analyzing energy consumption of industrial robots," in Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference, 2011

[6] S. Furber, and A.Brown, "Biologically-Inspired Massively-Parallel Architectures - Computing Beyond a Million Processors," In Application of Concurrency to System Design, ACSD '09. Ninth International Conference, 2009

[7] B. Liu, M.Hu, et al, "Bio-Inspired Ultra Lower-Power Neuromorphic Computing Engine for Embedded Systems," in Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference, 2013

[8] Y.Asano, K. Baba, et al. "A prediction method considering object motion for humanoid robot with visual sensor," In 2014 IEEE 13th International Workshop on Advanced Motion Control (AMC),2014

[9] K.Hashimoto, R.Mori, et al. "Tracking and catching of 3d flying target based on gag strateg," In:Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference, 2004

[10] O. Birbach, U. Frese, et al. "Realtime Perception for Catching a Flying Ball with a Mobile Humanoid," in IEEE International Conference on Robotics and Automation Shanghai International Conference Center, May 9-13, 2011, Shanghai, China, 2011

[11] F. Takagi, H. Sakahara, et al. "Navigation Control for Tracking and Catching a Moving Target," in The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems October 11-15, 2009 St. Louis, USA, 2009

[12] S. Chapmen, "Catching a baseball," *American Journal of Physics*. vol. 36, no.10, pp. 868-870,1968

[13] P.McLeod, and N.Reed, et al. "Do Fielders Know Where to Go to Catch the Ball or Only How to Get There? "*Journal of Experimental Psychology:Human Perception and Performance*. VoL 22, No. 3, 531-543, 1996

[14] M. McBeath and D. Shaffer, et al. "How baseball outfielders determine where to run to catch fly balls," *Science (New York, N.Y.)*. vol.268, no. 5210, pp.569-573, 1995

[15] M. Gillies and N.Dodgson, "Ball catching: An example of physiologically-based behavioural animation," *Eurographics UK*, 1999

[16] C. Koch "The Synaptic Input" in Biophysics of Computation: Information Processing in Single neuron Oxford, UK: Oxford University Press, Inc., pp85-112, 1999

[17] C. Koch "The Membrane Equation" in Biophysics of Computation: Information Processing in Single neuron Oxford, UK: Oxford University Press, Inc., pp5-14, 1999

[18] C. Koch "Synaptic Plasticity" in Biophysics of Computation: Information Processing in Single neuron Oxford, UK: Oxford University Press, Inc., pp308-329, 1999

[19] O. Bichler, D. Querlioz, et al. "Unsupervised features extraction from asynchronous silicon retina through Spike-Timing-Dependent Plasticity," in: Neural Networks (IJCNN), The 2011 International Joint Conference on, 2011

[20] F. Gallupi, and S. Furber, "Representing and decoding rank order codes using polychronization in a network of spiking neuron" in Neural Networks (IJCNN), The 2011 International Joint Conference, 2011.

[21] E. M. Izhikevich. "Polychronization: Computation with Spikes," in Neural Computation (Volume:18 , Issue: 2), pp 245 - 282, 2006.

[22] Dynamic Vision Sensor Overview. iniLabs. Available at: http://inilabs.com/products/dynamic-vision-sensors/

[23] User Guide: DVS128 page. iniLabs. Available at: http://inilabs.com/support/hardware/dvs128/

[24] File format (AEDAT). AEDAT formats. iniLabs. Available at: http://inilabs.com/support/software/fileformat/

[25] D. Magliocchetti-Lombi, "Comparison of machine learning algorithms for use on the spinnaker platform," *Final Year Project Thesis*, Imperial College London, 2015

[26] SpiNNaker Project - The SpiNNaker Chip. APT Advanced Processor Technologies Research Group. Available at: http://apt.cs.manchester.ac.uk/projects/SpiNNaker/SpiNNchip/

[27] Biologically-Inspired Massively Parallel Computation (BIMPC) Home Page. APT Advanced Processor Technologies Research Group. Available at: http://apt.cs.manchester.ac.uk/projects/BIMPC/

[28] SpiNNaker Project - Boards and Machines. APT Advanced Processor Technologies Research Group. Available at: http://apt.cs.manchester.ac.uk/projects/SpiNNaker/hardware/

[29] W. Li, "Machine Learning Tasks on SpiNNaker Platform," *Final Year Project Thesis*, Imperial College London, 2015

10. Appendix:

10.1 Cell Membrane Potential Equation:

$$C\frac{dV_m(t)}{dt} + g_{syn}(t)(V_m(t) - E_{syn}) + \frac{V_m(t) - V_{rest}}{R} = 0 \tag{11}$$

Rearrange equation (11):

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{RC} = \frac{g_{syn}(t)(V_m(t) - E_{syn})}{C} + \frac{V_{rest}}{RC} \tag{12}$$

Omit $V_m(t)$ in the first term on the right hand side of equation (12):

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{RC} = \frac{g_{syn}(t) - E_{syn}}{C} + \frac{V_{rest}}{RC} \tag{13}$$

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{RC} = \frac{g_{syn}(t) - E_{syn}}{C} + \frac{V_{rest}}{RC} \tag{14}$$

$$\tau_m = RC \tag{15}$$

$$g_{syn}(t) = g_o \times \exp\left(-\frac{t}{\tau_{syn}}\right) \tag{16}$$

$V_m(t)$ can be characterised as the exact solution and possible solution.

The exact solution is solved when the right hand side is 0.

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{\tau_m} = 0 \tag{17}$$

This gives the membrane potential in the following form.

$$\frac{dV_m(t)}{dt} + \frac{V_m(t)}{\tau_m} = 0 \tag{18}$$

$$V_m(t) = C \times \exp\left(-\frac{t}{\tau_m}\right) \tag{19}$$

The analytic possible solution of equation (14) is complex, but since g_{syn} is in a exponential relationship, therefore, the trial solution is as following.

10. Appendix:

$$V_m(t) = A \times \exp\left(-\frac{t}{\tau_{syn}}\right) \tag{20}$$

Adding the possible solution and exact solution together, we can obtain equation (10).

10.2 Testing Results:

10.2.1 Angle Variation Test 1:

10.2.1 Angle Variation Test 1: Same Starting Position

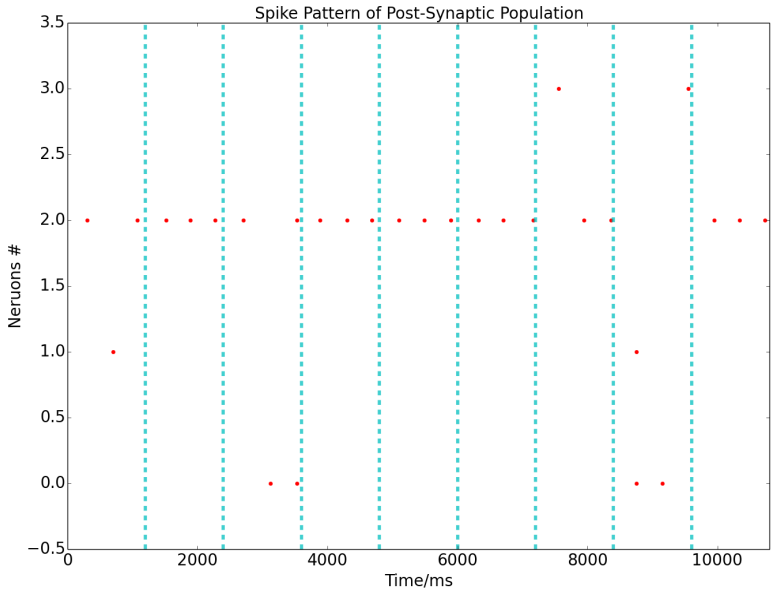


Figure 10.2.1.1 Angle Variation Test 1: Left to Right

10. Appendix:

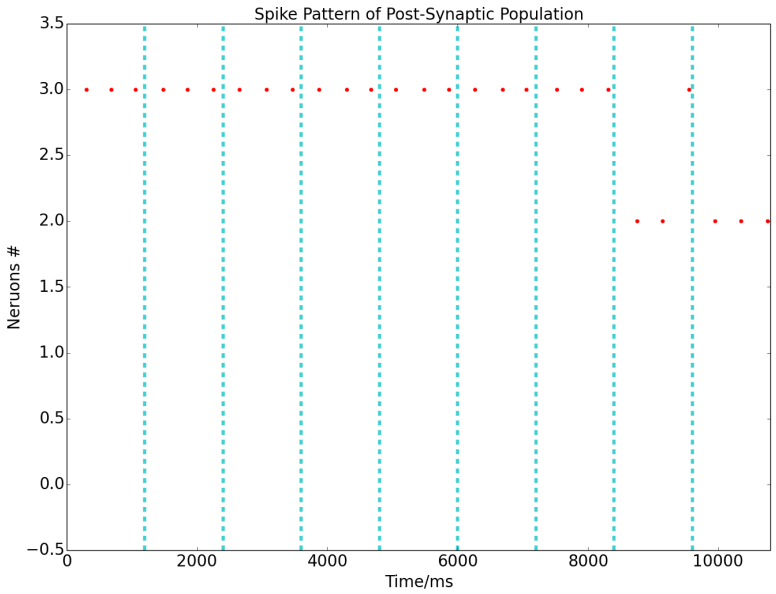


Figure 10.2.1.2 Angle Variation Test 1: Right to Left

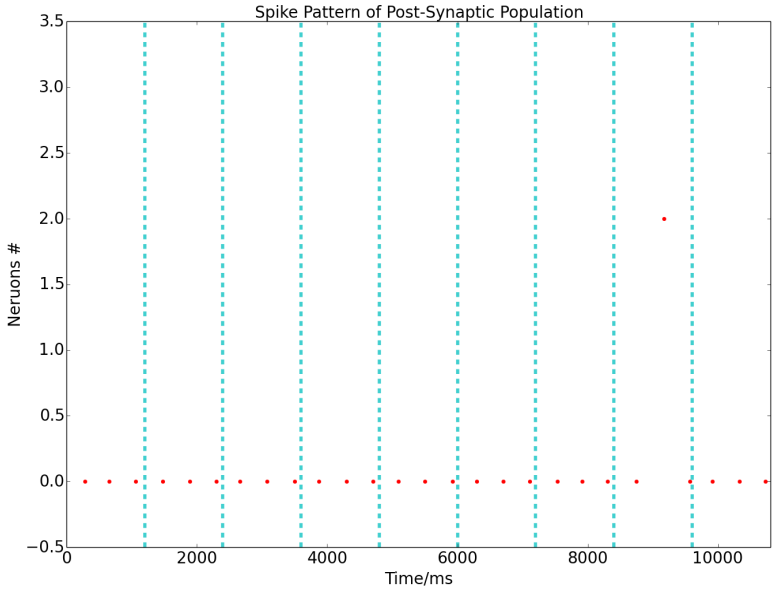


Figure 10.2.1.3 Angle Variation Test 1: Top to Bottom

10. Appendix:

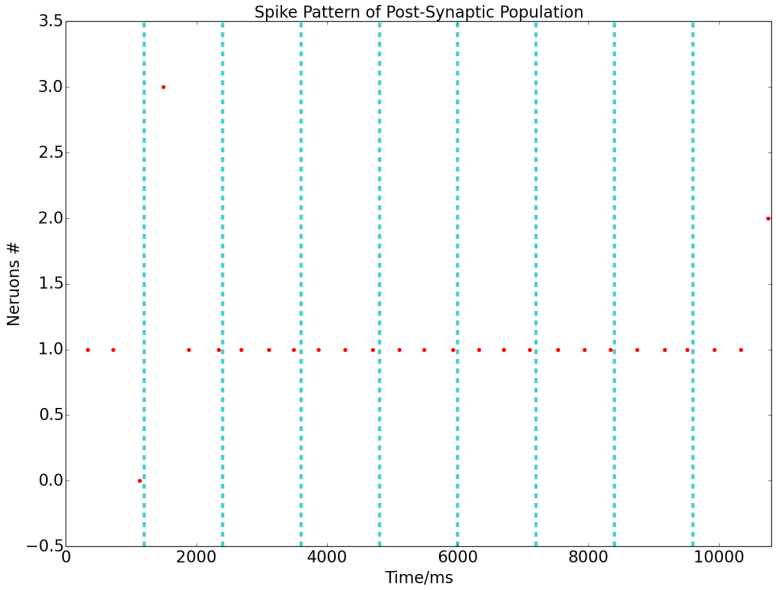


Figure 10.2.1.4 Angle Variation Test 1: Bottom to Top

10.2.2 Angle Variation Test 2: Crossing Centre of FOV

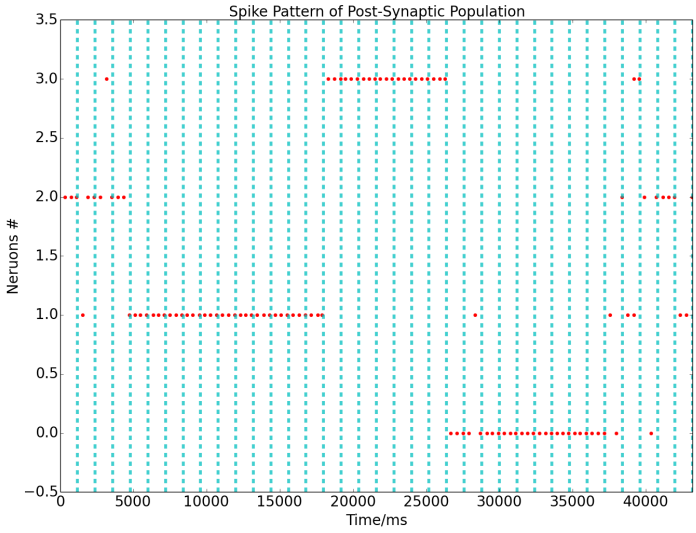


Figure 10.2.2.1 Angle Variation Test 2: Crossing Centre of FOV

10.2.3 Ball Trajectory Shifting Test

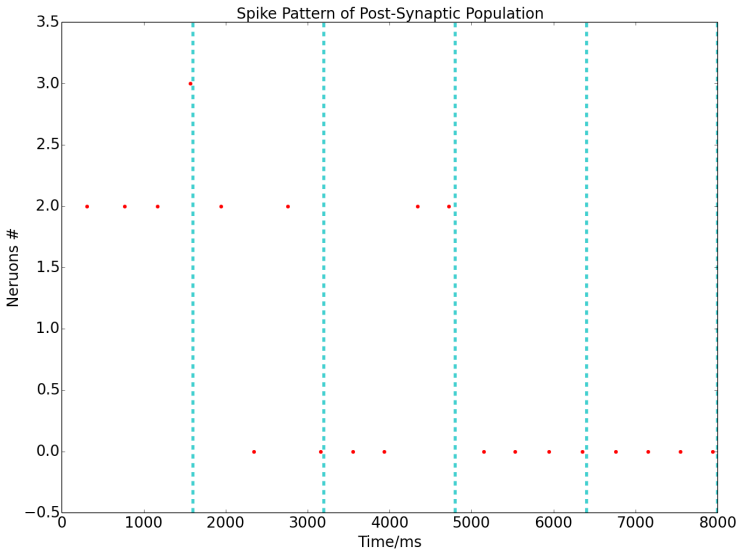


Figure 10.2.3.1 Ball Trajectory Shifting Test: Left to Right

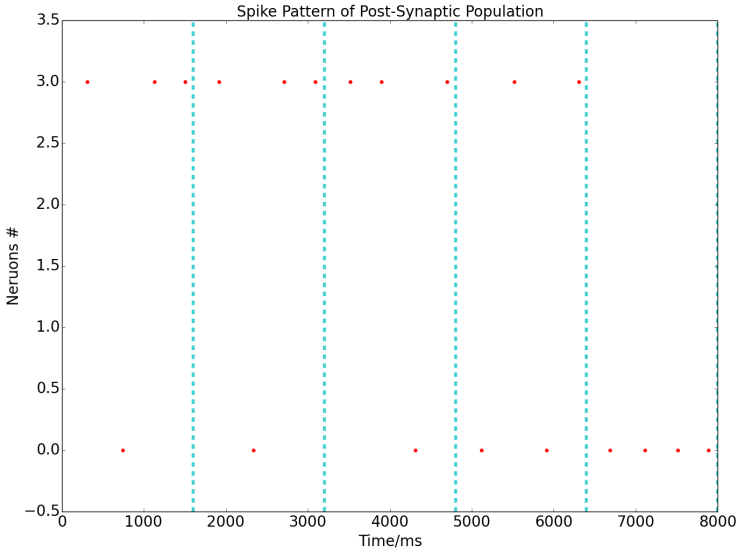


Figure 10.2.3.2 Ball Trajectory Shifting Test: Right to Left

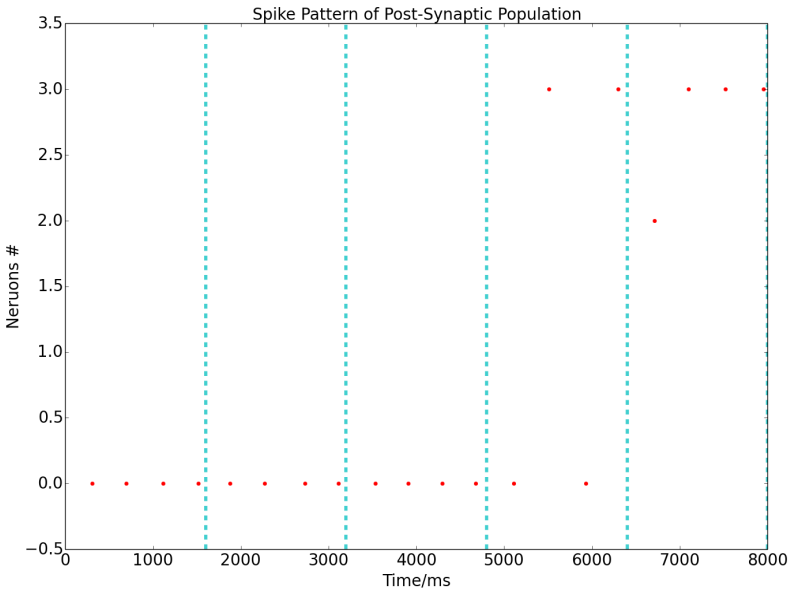


Figure 10.2.3.3 Ball Trajectory Shifting Test: Top to Bottom

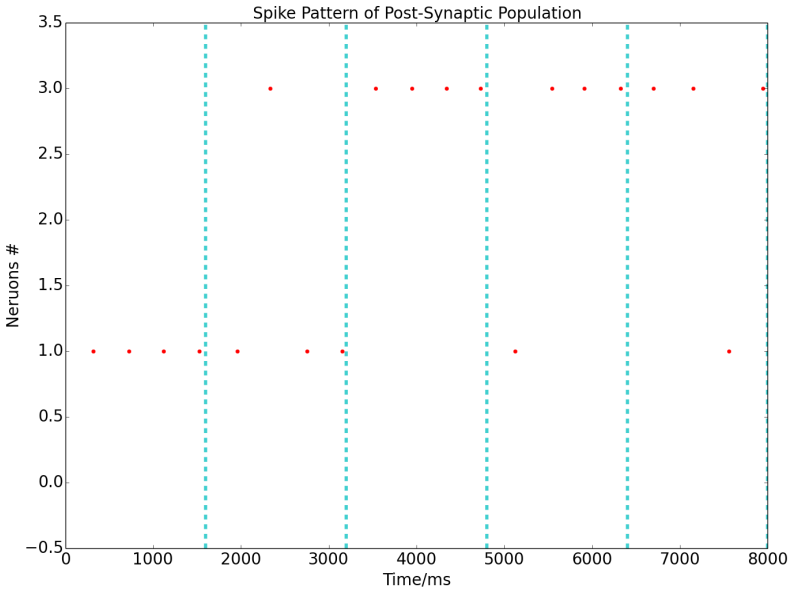


Figure 10.2.3.4 Ball Trajectory Shifting Test: Bottom to Top

10. Appendix:

10.2.4 Speed Test 1: Number of Spiking of Ball Trajectory

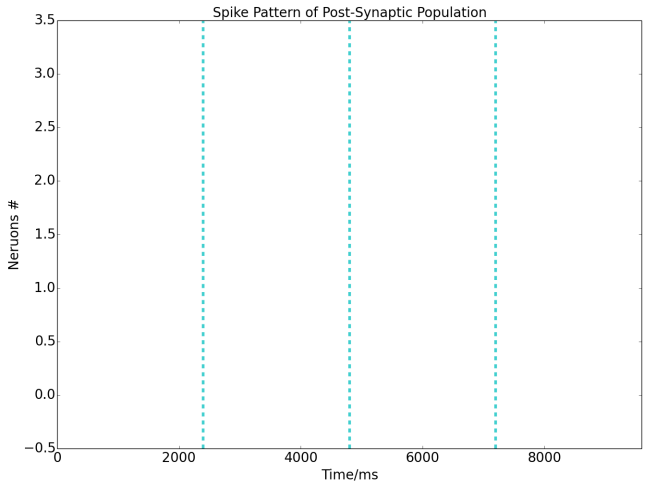


Figure 10.2.4.1 Speed Test 1: 44 Spiking in Trajectory

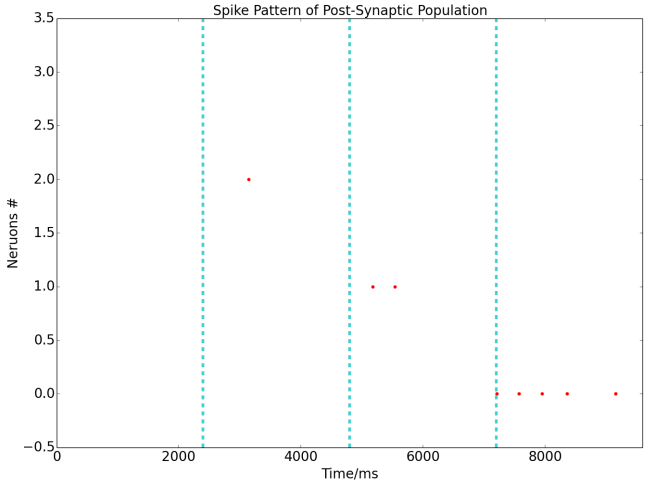


Figure 10.2.4.2 Speed Test 1: 55 Spiking in Trajectory

10. Appendix:

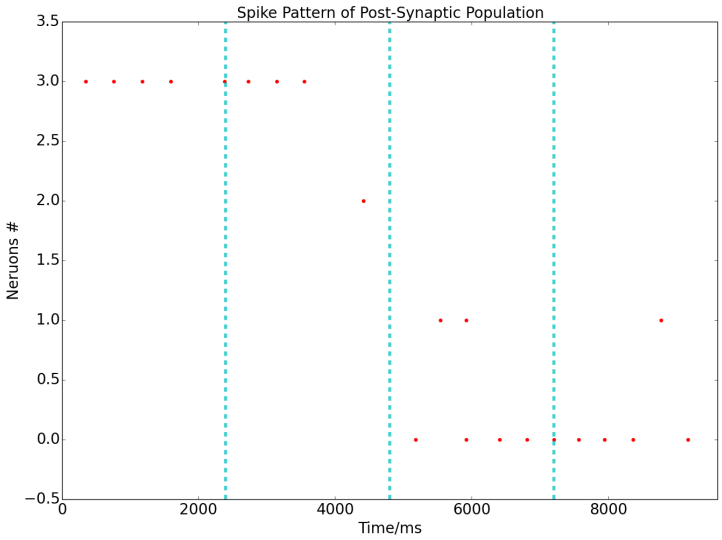


Figure 10.2.4.3 Speed Test 1: 66 Spiking in Trajectory

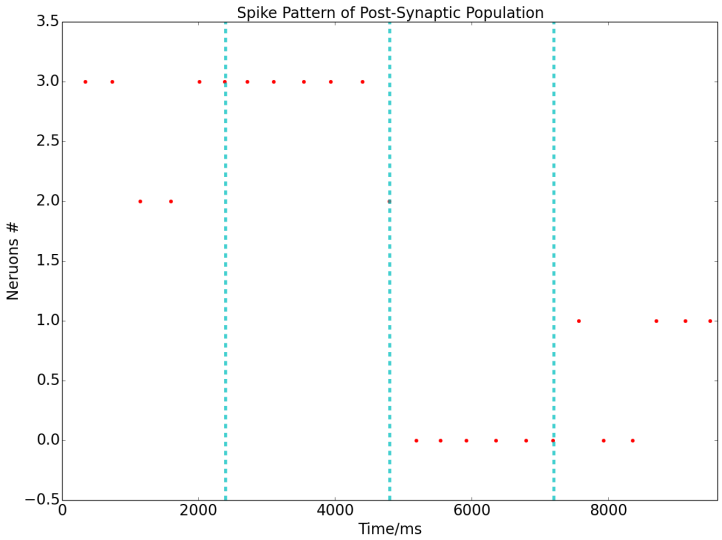


Figure 10.2.4.4 Speed Test 1: 77 Spiking in Trajectory

10. Appendix:

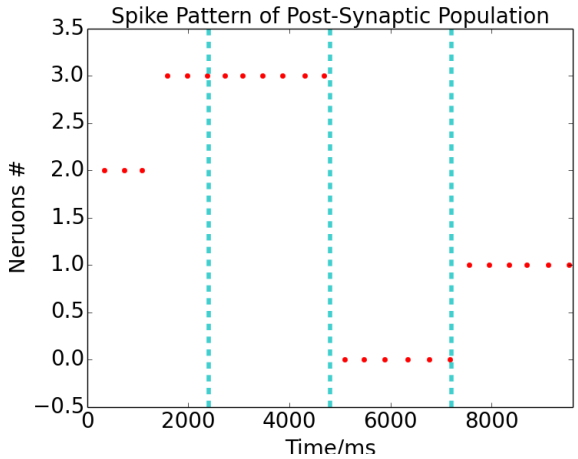


Figure 10.2.4.5 Speed Test 1: 88 Spiking in Trajectory

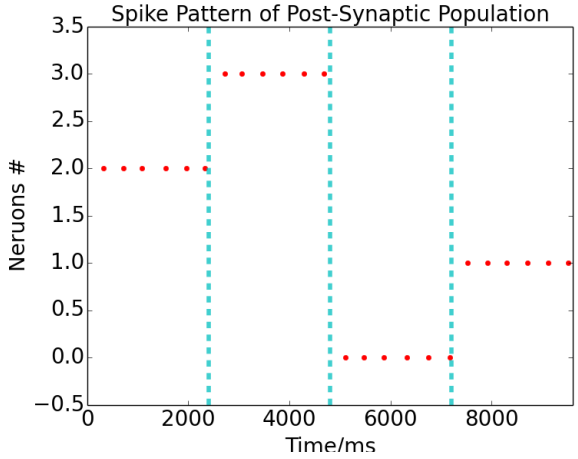


Figure 10.2.4.6 Speed Test 1: 99 Spiking in Trajectory

10. Appendix:

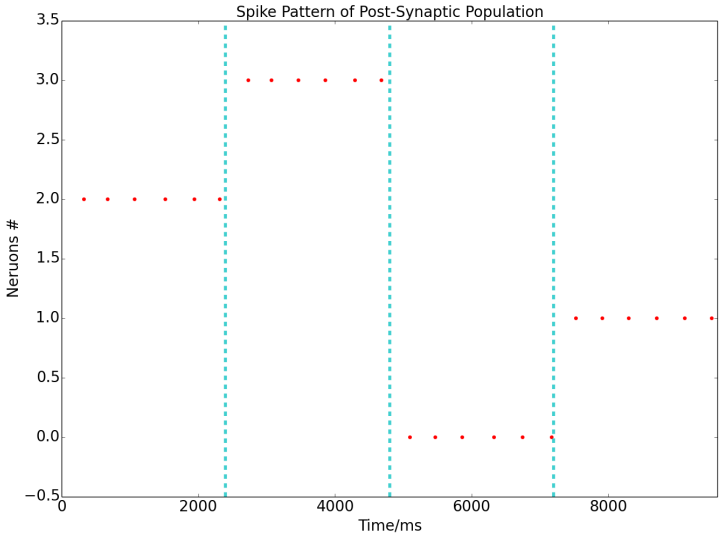


Figure 10.2.4.7 Speed Test 1: 110 Spiking in Trajectory

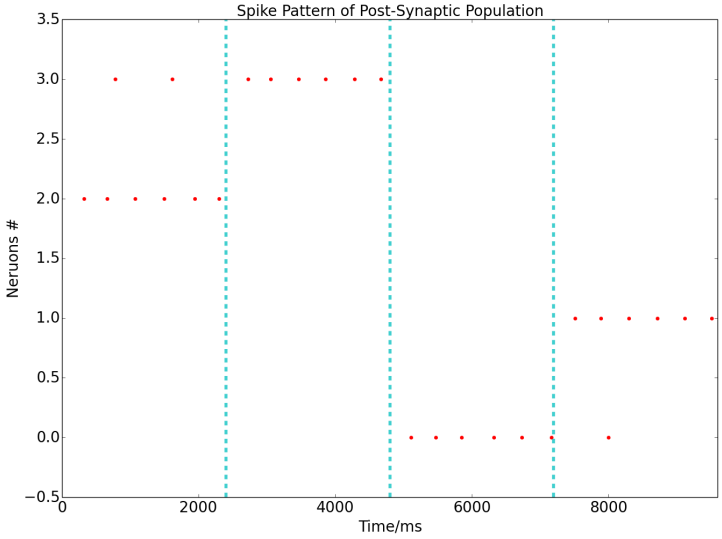


Figure 10.2.4.8 Speed Test 1: 121 Spiking in Trajectory

10. Appendix:

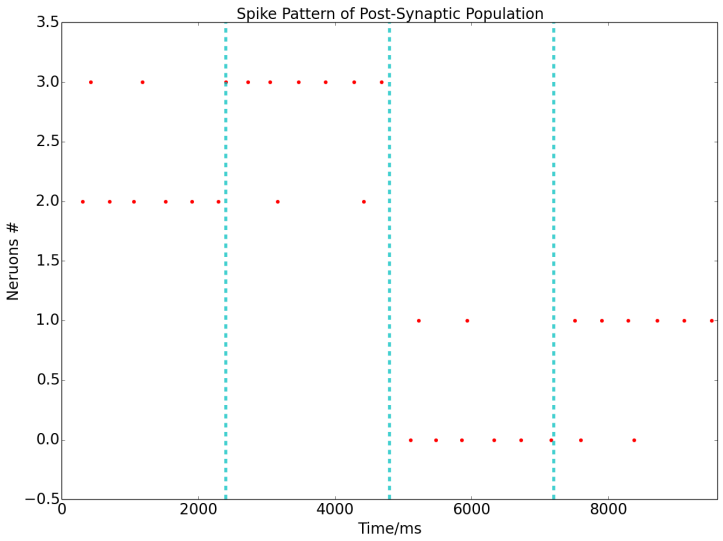


Figure 10.2.4.9 Speed Test 1: 132 Spiking in Trajectory

10.2.5 Speed Test 2: Period of Ball Trajectory

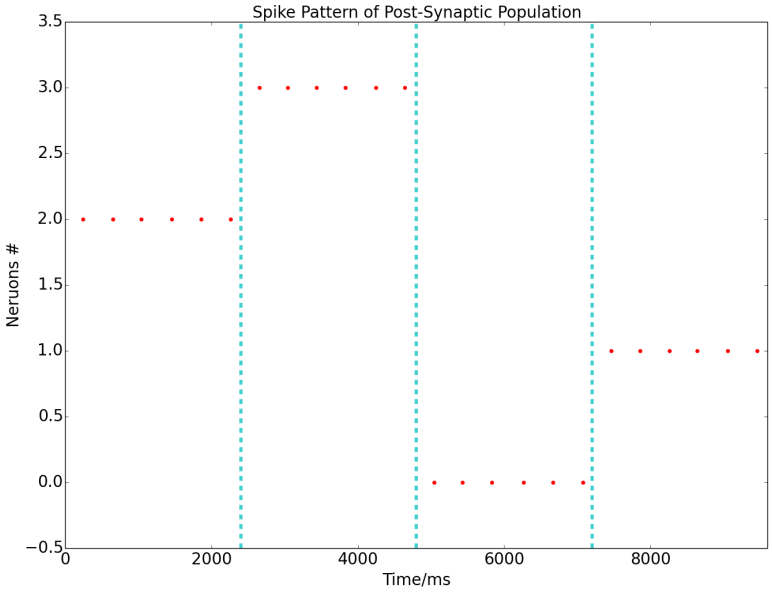


Figure 10.2.5.1 Speed Test 2: 100ms active within 400ms

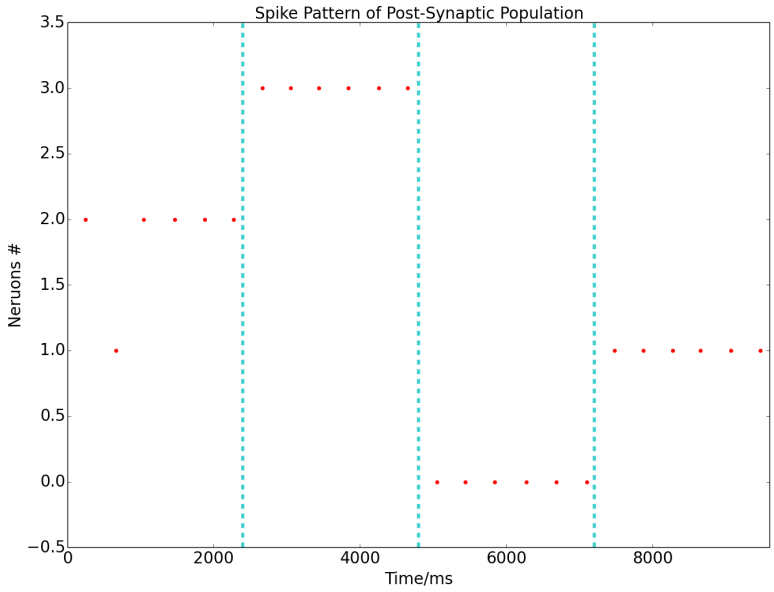


Figure 10.2.5.2 Speed Test 2: 125ms active within 400ms

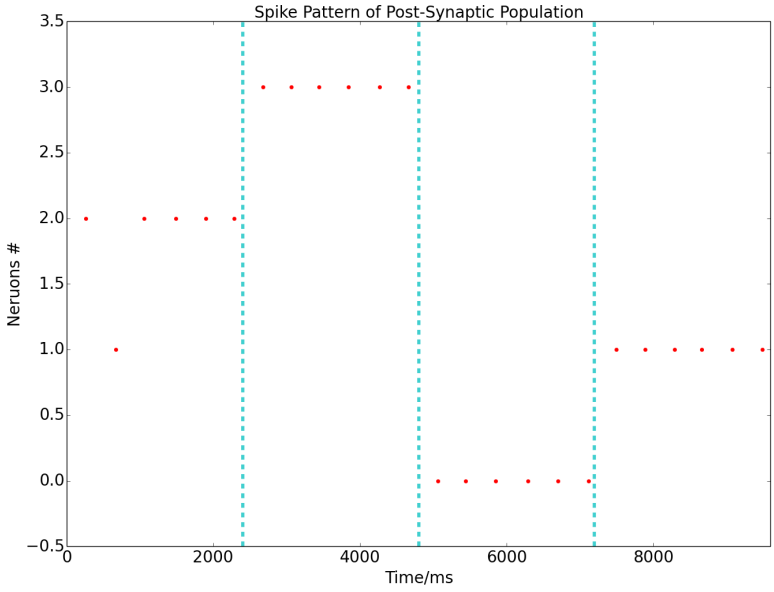


Figure 10.2.5.3 Speed Test 2: 150ms active within 400ms

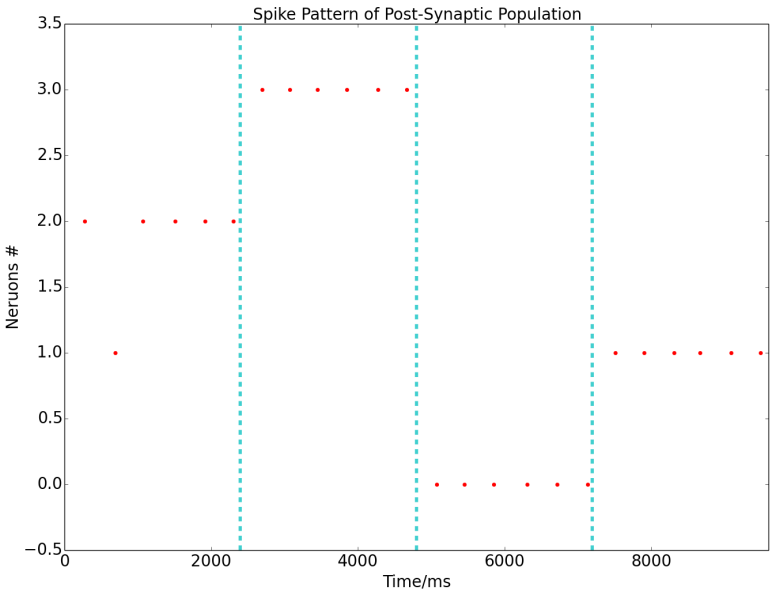


Figure 10.2.5.4 Speed Test 2: 175ms active within 400ms

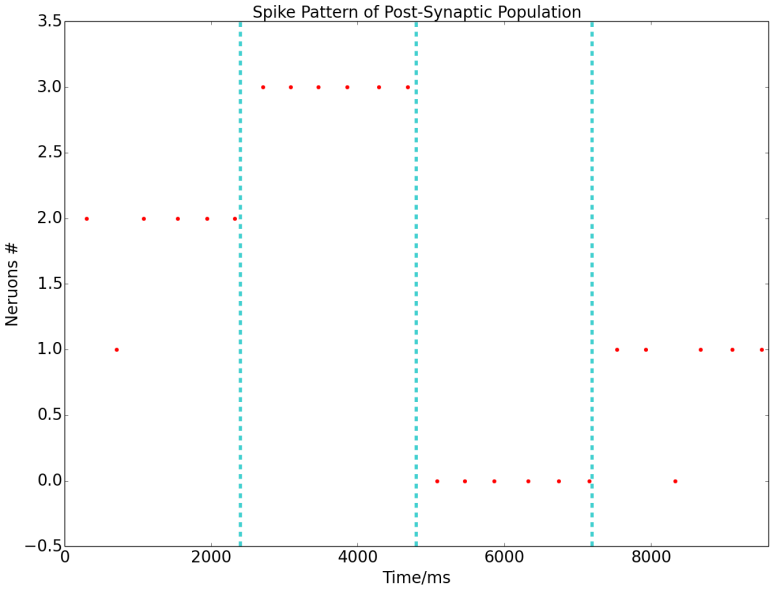


Figure 10.2.5.5 Speed Test 2: 200ms active within 400ms

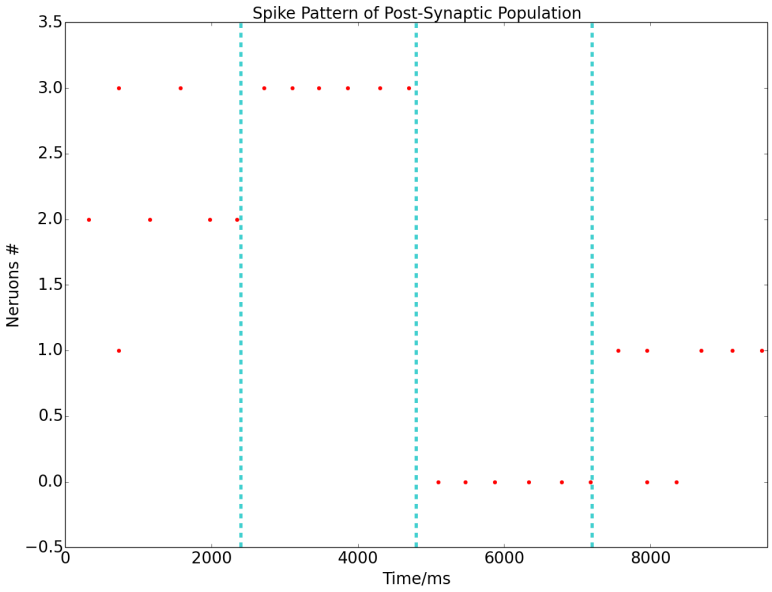


Figure 10.2.5.6 Speed Test 2: 225ms active within 400ms

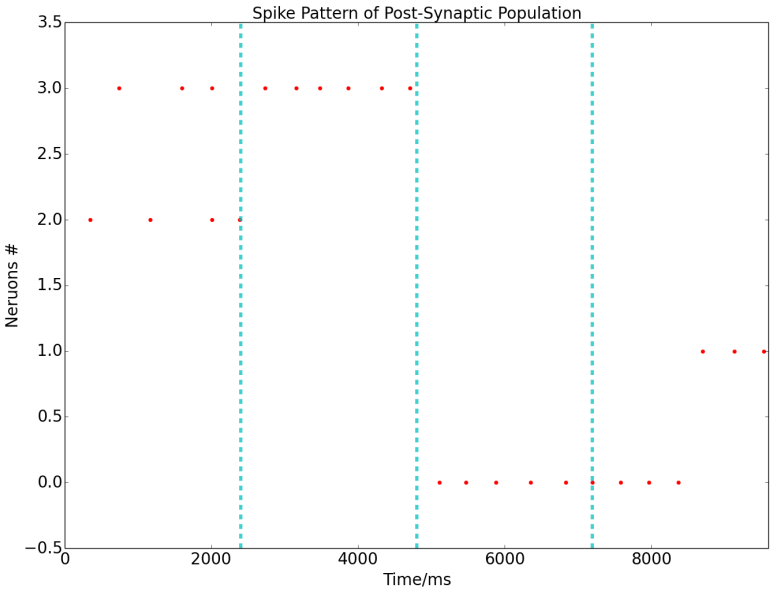


Figure 10.2.5.7 Speed Test 2: 250ms active within 400ms

10.2.6 Ball Size Test

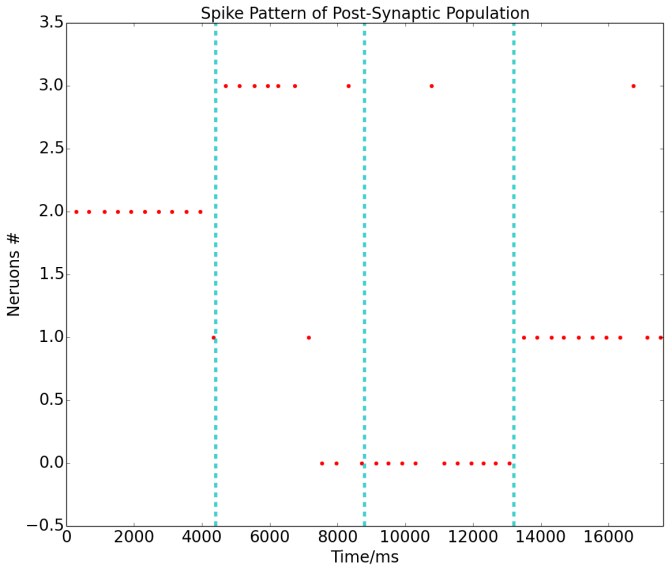


Figure 10.2.6.1 Ball Size Test

10.2.7 Background Noise Activity Test

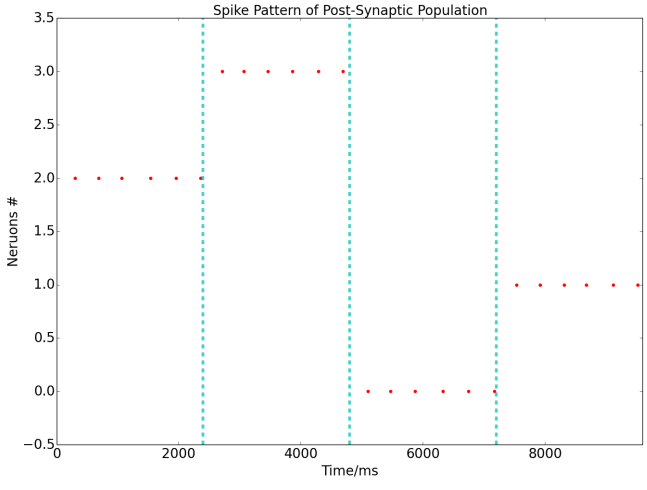


Figure 10.2.7.1 11 Noise Spiking in Active Period

10. Appendix:

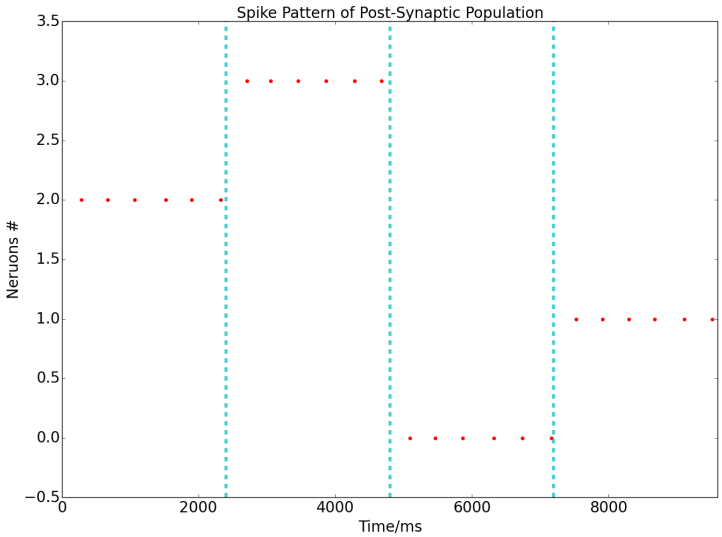


Figure 10.2.7.2 22 Noise Spiking in Active Period

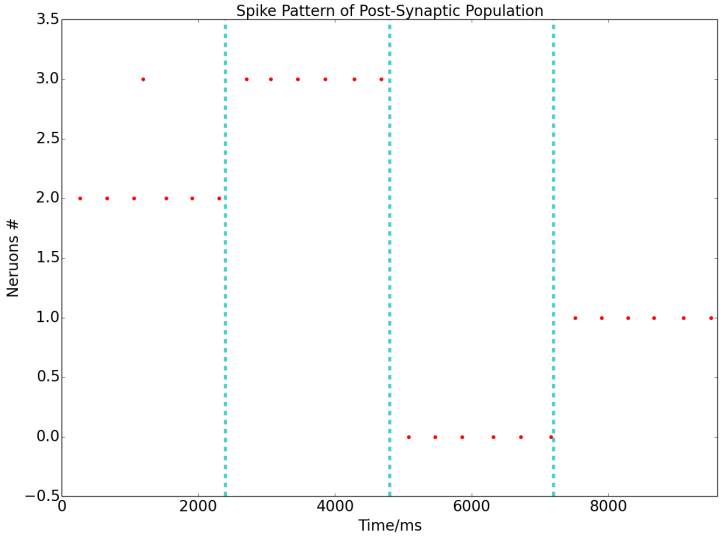


Figure 10.2.7.3 33 Noise Spiking in Active Period

10. Appendix:

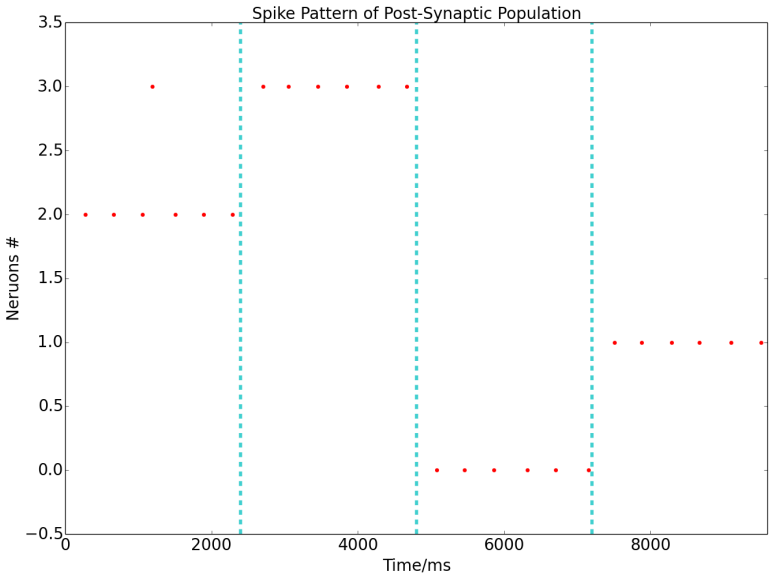


Figure 10.2.7.4 44 Noise Spiking in Active Period

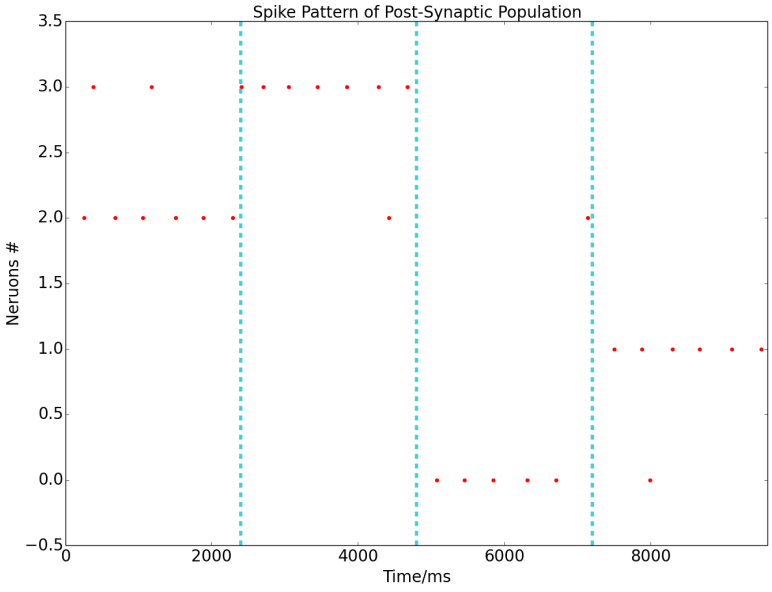


Figure 10.2.7.5 55 Noise Spiking in Active Period

10. Appendix:

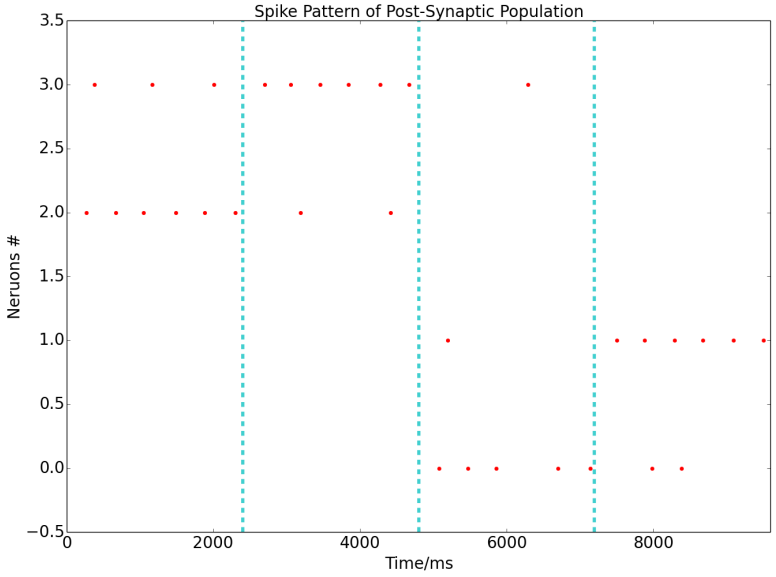


Figure 10.2.7.6 66 Noise Spiking in Active Period

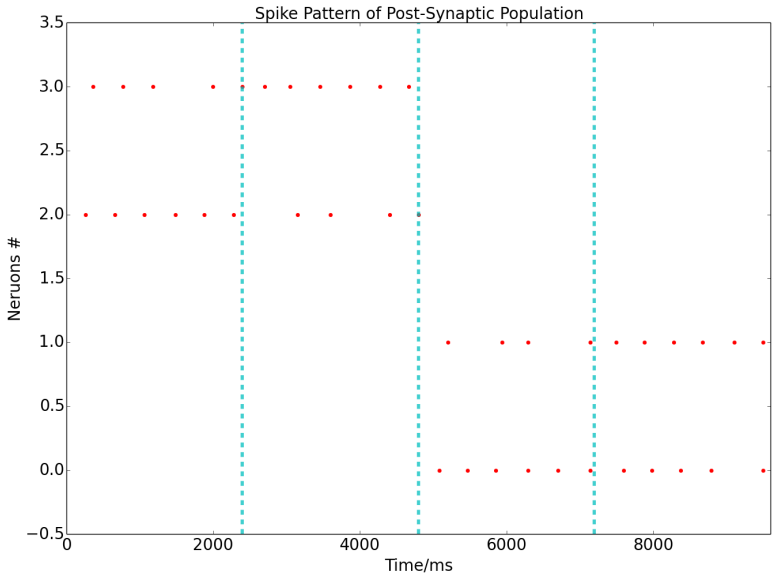


Figure 10.2.7.7 77 Noise Spiking in Active Period

10. Appendix:

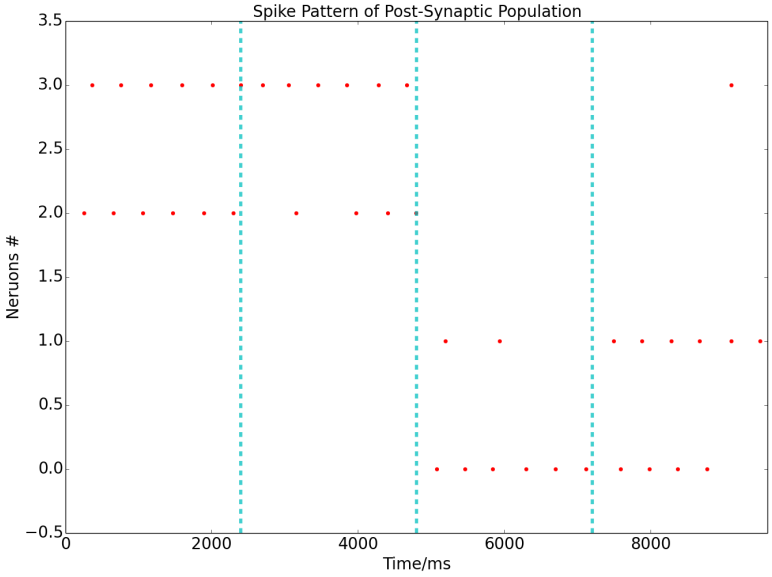


Figure 10.2.7.8 88 Noise Spiking in Active Period

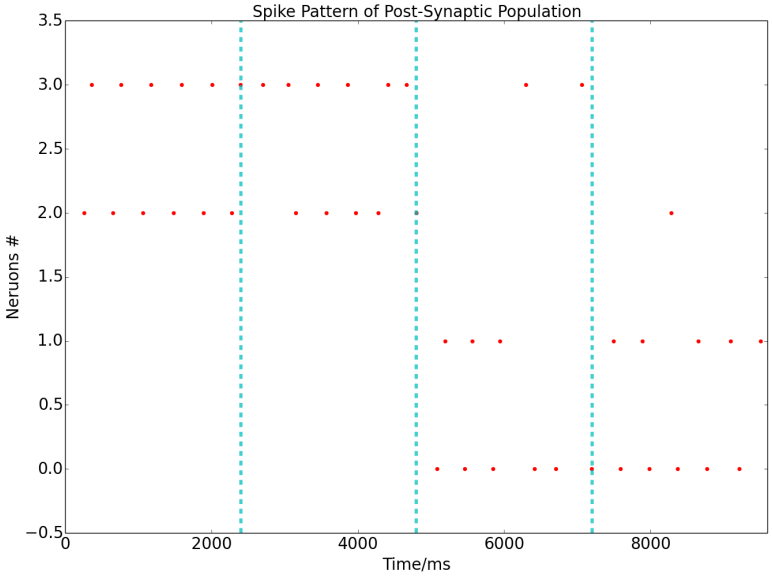


Figure 10.2.7.9 99 Noise Spiking in Active Period

10. Appendix:

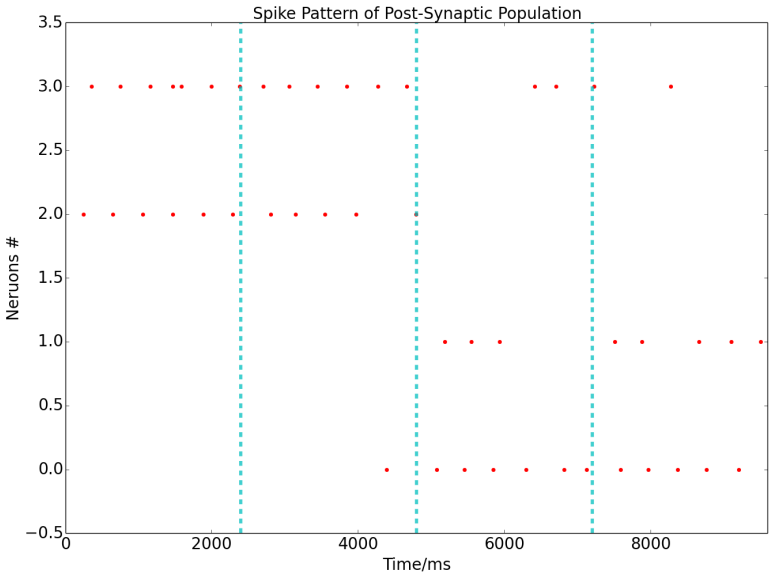


Figure 10.2.7.10 110 Noise Spiking in Active Period

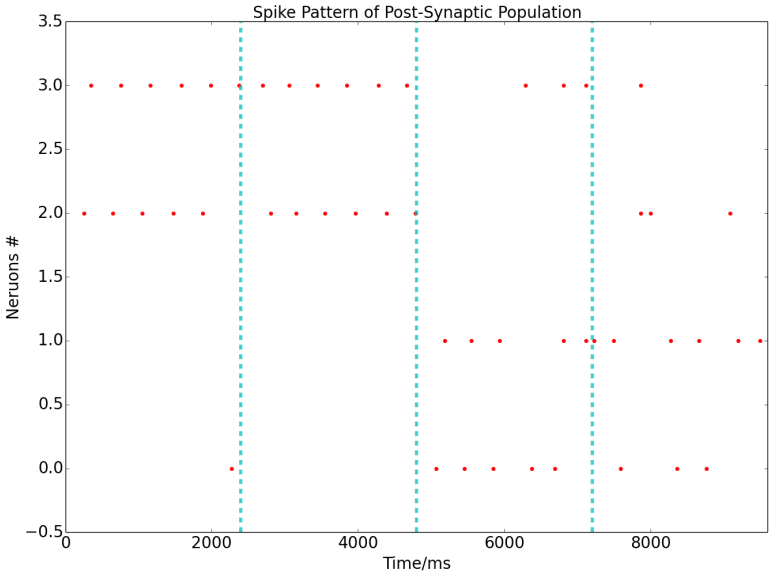


Figure 10.2.7.11 121 Noise Spiking in Active Period

10. Appendix:

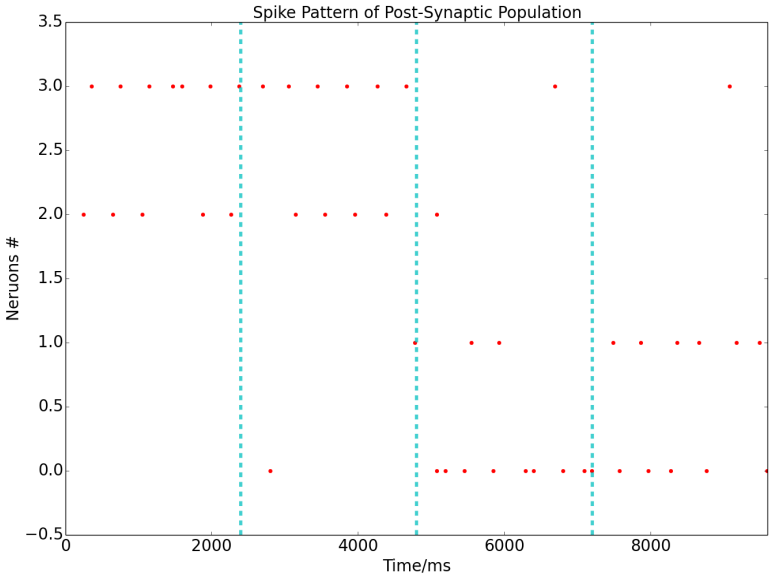


Figure 10.2.7.12 132 Noise Spiking in Active Period

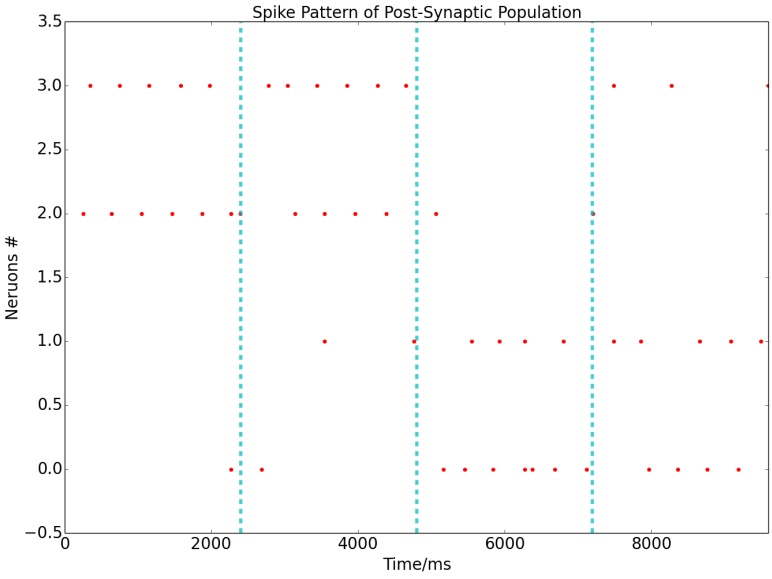


Figure 10.2.7.13 143 Noise Spiking in Active Period

10. Appendix:

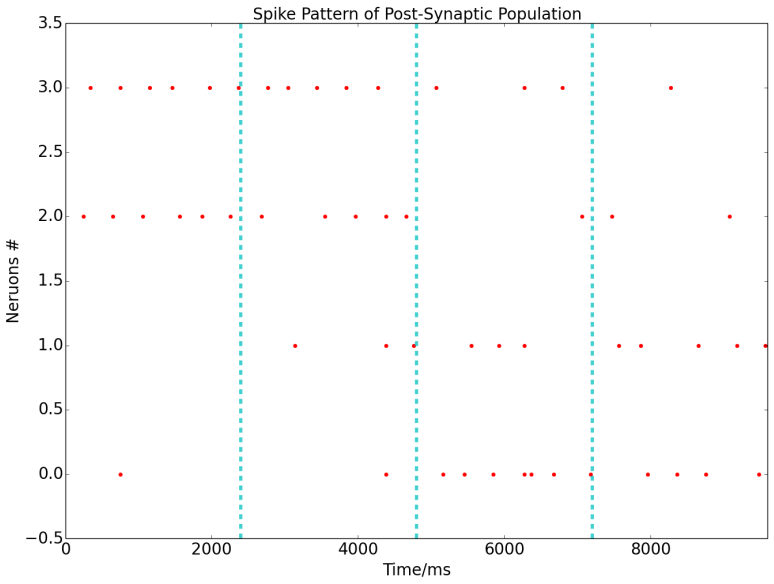


Figure 10.2.7.14 154 Noise Spiking in Active Period

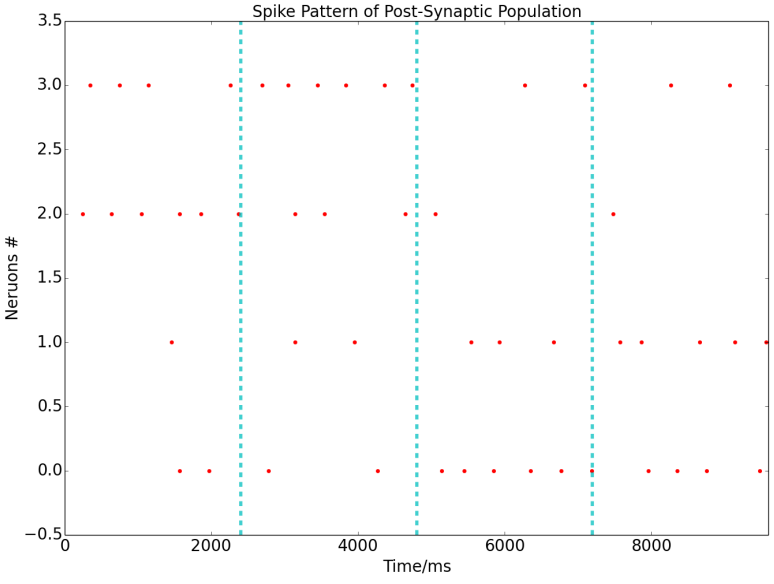


Figure 10.2.7.15 165 Noise Spiking in Active Period

10. Appendix:

10.3 Code:

10.3.1 all_four_directions.py:

```
import paer
import numpy as np

base_dir = 'C:\Python27\python-aer-0.1.3'
file1 = 'left_to_right_1.aedat'
file2 = 'right_to_left_1.aedat'
file3 = 'top_to_bottom_1.aedat'
file4 = 'bottom_to_top_1.aedat'

# Each ball movement should be .5s long
animation_time = 0.2

# 3,280 events per second for 16*16 is reasonable for ball movement (might be even
num_events_p_s = 3280

# Helper function to read a file. Given (min,max) which are data ranges for extract
#  suitably sparse output.
def get_data(file, min, max, animation_time=animation_time, num_events=num_events_p
    aefile = paer.aefile(file, max_events=max+1)
    aedata = paer.aedata(aefile)
    #print 'Points: %i, Time: %0.2f. End Time: %0.2f. Start Time: %0.2f. Sparsity: %
    #                                                    np.floor(len(aefile.data)/num_e

    sparse = aedata[min:max].make_sparse(np.floor(len(aefile.data)/num_events))
    for i in range(1,len(sparse.ts)):
        if(sparse.ts[-i]!=0):
            last_index = -i
```

```
                break
        print "last index: %d, last time stamp %d"%(last_index, sparse.ts[last_index])
        actual_time = (sparse.ts[last_index]-sparse.ts[0])/1000000
        scale = actual_time/animation_time
        sparse.ts = (offset * 1000000) + np.round((sparse.ts-sparse.ts[0])/scale)
        # print sparse_ts[0], sparse_ts[-1], sparse_ts[-1]-sparse_ts[0], (sparse_ts[-1]
        return sparse[0:len(sparse.ts)+last_index+1]
        #return aedata

# Loop through all files - indexes are extrapolated.
x1 = get_data(file1, 340001, 748073, offset=0*animation_time)
x2 = get_data(file2, 299247, 589011, offset=1*animation_time)
x3 = get_data(file3, 83625,  139803, offset=2*animation_time)
x4 = get_data(file4, 69529,  127081, offset=3*animation_time)

# Need to pre-load a file, to get the correct headers when writing!
#lib = paer.aefile(file1, max_events=1)

final = paer.concatenate( (x1,x2,x3,x4) )
final_16 = final.downsample((16,16))

#lib.save(final, 'test.aedat')
#lib.save(final_16, 'test_16.aedat')
d1 = x1.downsample((16,16))
d2 = x2.downsample((16,16))
d3 = x3.downsample((16,16))
d4 = x4.downsample((16,16))

d1.save_to_mat('downsampled_left_to_right_2.mat')
d2.save_to_mat('downsampled_right_to_left_2.mat')
d3.save_to_mat('downsampled_top_to_bottom_2.mat')
d4.save_to_mat('downsampled_bottom_to_top_2.mat')
```

10.3.2 __init__.py:

```
"""
Author: Dario ML
Program: src/__init__.py
Description: main file for python-ae
"""

from PIL import Image
import math
import numpy as np
import scipy.io
import os,time
import matplotlib.pyplot as plt
from matplotlib import cm

class aefile(object):
    def __init__(self, filename, max_events=1e6):
        self.filename = filename
        self.max_events = max_events
        self.header = []
        self.data, self.timestamp = self.read()

    # alias for read
    def load(self):
        return self.read()

    def read(self):
        with open(self.filename, 'r') as f:
            line = f.readline()
            while line[0] == '#':
                self.header.append(line)
                if line[0:9] == '#!AER-DAT':
```

```
                aer_version = line[9];
            current = f.tell()
            line = f.readline()



        if aer_version != '2':
            raise Exception('Invalid AER version. Expected 2, got %s' % aer_ver

        f.seek(0,2)
        numEvents = math.floor( ( f.tell() - current ) / 8 )

        if numEvents > self.max_events:
            print 'There are %i events, but max_events is set to %i. Will only
            numEvents = self.max_events

        f.seek(current)

        timestamps = np.zeros( numEvents )
        data       = np.zeros( numEvents )

        for i in range(int(numEvents)):
            data[i] = int(f.read(4).encode('hex'), 16)
            timestamps[i] = int(f.read(4).encode('hex'), 16)

        return data, timestamps

def save(self, data=None, filename=None, ext='aedat'):
    if filename is None:
        filename = self.filename
    if data is None:
        data = aedata(self)
    if ext is 'aedat':
```

```
            # unpack our 'data'
            ts = data.ts
            data = data.pack()

            with open(filename, 'w') as f:
                for item in self.header:
                    f.write(item)
                print
                print
                no_items = len(data)
                for i in range(no_items):
                    f.write(hex(int(data[i]))[2:].zfill(8).decode('hex'))
                    f.write(hex(int(ts[i]))[2:].zfill(8).decode('hex'))

    def unpack(self):
        noData = len(self.data)

        x = np.zeros(noData)
        y = np.zeros(noData)
        t = np.zeros(noData)

        for i in range(noData):

            d = int(self.data[i])

            t[i] = d & 0x1
            x[i] = 128-((d >> 0x1) & 0x7F)
            y[i] = (d >> 0x8) & 0x7F
        return x,y,t


class aedata(object):
```

```
def __init__(self, ae_file=None):
    self.dimensions = (128,128)
    if isinstance(ae_file, aefile):
        self.x, self.y, self.t = ae_file.unpack()
        self.ts = ae_file.timestamp
    elif isinstance(ae_file, aedata):
        self.x, self.y, self.t = aedata.x, aedata.y, aedata.t
        self.ts = ae_file.ts
    else:
        self.x, self.y, self.t, self.ts = np.array([]),np.array([]),np.array([]

def __getitem__(self, item):
    rtn = aedata()
    rtn.x = self.x[item]
    rtn.y = self.y[item]
    rtn.t = self.t[item]
    rtn.ts= self.ts[item]
    return rtn

def __setitem__(self, key, value):
    self.x[key] = value.x
    self.y[key] = value.y
    self.t[key] = value.t
    self.ts[key] = value.ts

def __delitem__(self, key):
    self.x = np.delete(self.x,  key)
    self.y = np.delete(self.y,  key)
    self.t = np.delete(self.t,  key)
    self.ts = np.delete(self.ts,  key)

def save_to_mat(self, filename):
```

```
        scipy.io.savemat(filename, {'X':self.x, 'Y':self.y, 't': self.t, 'ts': self

    def pack(self):
        noData = len(self.x)
        packed = np.zeros(noData)
        for i in range(noData):
            packed[i] =  (int(self.t[i]) & 0x1)
            packed[i] += (int(128-self.x[i]) & 0x7F) << 0x1
            packed[i] += (int(self.y[i]) & 0x7F) << 0x8

        return packed

    # TODO
    # performance here can be improved by allowing indexing in the AE data.
    # For now, I expect this not to be done often
    def make_sparse(self, ratio):
        indexes = np.random.randint(0,len(self.x),math.floor(len(self.x)/ratio))
        indexes.sort()

        rtn = aedata()
        rtn.x = self.x[indexes]
        rtn.y = self.y[indexes]
        rtn.t = self.t[indexes]
        rtn.ts = self.ts[indexes]

        return rtn

    def __repr__(self):
        return "%i total [x,y,t,ts]: [%s, %s, %s, %s]" % (len(self.x), self.x, self

    def __len__(self):
        return len(self.x)
```

```
def interactive_animation(self, step=5000, limits=(0,128), pause=0):
    plt.ion()
    fig = plt.figure(figsize=(6,6))
    plt.show()
    ax = fig.add_subplot(111)

    start = 0
    end = step-1
    while(start < len(self.x)):
        ax.clear()
        ax.scatter(self.x[start:end],self.y[start:end],s=20,c=self.t[start:end]
        ax.set_xlim(limits)
        ax.set_ylim(limits)
        start += step
        end += step
        plt.draw()
        time.sleep(pause)

def downsample(self,new_dimensions=(16,16)):
    # TODO
    # Make this cleaner
    assert self.dimensions[0]%new_dimensions[0] is 0
    assert self.dimensions[1]%new_dimensions[1] is 0

    rtn = aedata()

    rtn.ts = self.ts
    rtn.t = self.t
    rtn.x = np.floor(self.x / (self.dimensions[0] / new_dimensions[0]))
    rtn.y = np.floor(self.y / (self.dimensions[1] / new_dimensions[1]))
```

```
        return rtn

    def to_matrix(self, dim=(128,128)):
        return make_matrix(self.x, self.y, self.t, dim=dim)

def make_matrix(x, y, t, dim=(128,128)):
    image = np.zeros(dim)
    events= np.zeros(dim)

    for i in range(len(x)):
        image[y[i]-1,x[i]-1] -= t[i]-0.5
        events[y[i]-1,x[i]-1] += 1

    # http://stackoverflow.com/questions/26248654/numpy-return-0-with-divide-by-zer
    np.seterr(divide='ignore', invalid='ignore')

    result = 0.5+(image / events)
    result[events == 0] = 0.5
    return result

def create_pngs(data,prepend,path="",step=3000,dim=(128,128)):
    if not os.path.exists(path):
        os.makedirs(path)

    idx = 0
    start = 0;
    end = step-1;
    while(start < len(data.x)):
        image = make_matrix(data.x[start:end],data.y[start:end],data.t[start:end],
        img_arr = (image*255).astype('uint8')
        im = Image.fromarray(img_arr)
        im.save(path+'/'+prepend+("%05d" % idx)+".png")
```

```
        idx += 1

        start += step
        end += step

def concatenate(a_tuple):
    rtn = aedata()
    rtn.x = np.concatenate(tuple([a_tuple[i].x for i in range(len(a_tuple))]))
    rtn.y = np.concatenate(tuple([a_tuple[i].y for i in range(len(a_tuple))]))
    rtn.t = np.concatenate(tuple([a_tuple[i].t for i in range(len(a_tuple))]))
    rtn.ts = np.concatenate(tuple([a_tuple[i].ts for i in range(len(a_tuple))]))
    return rtn

    # np.concatenate(a_tuple)
```

10.3.3 input_normaliser.py:

```
__author__ = "hanyi"
import scipy.io
import math
argw = []

prefix_l_t_r = 'l_to_r'
prefix_r_t_l = 'r_to_l'
prefix_t_to_b = 't_to_b'
prefix_b_to_t = 'b_to_t'
post_fix = '.mat'

def get_data(filename):
    dvs_data = scipy.io.loadmat(filename)
    ts = dvs_data['ts'][0]
    ts = (ts - ts[0])/1000 #from us to ms
    x = dvs_data['X'][0]
```

```
    y = dvs_data['Y'][0]
    p = dvs_data['t'][0]
    return x,y,p,ts

for i in range(1,10):
    argw.append(get_data(prefix_l_t_r+str(i)+post_fix))

for i in range(1,10):
    argw.append(get_data(prefix_r_t_l+str(i)+post_fix))

for i in range(1,10):
    argw.append(get_data(prefix_t_to_b+str(i)+post_fix))

for i in range(1,10):
    argw.append(get_data(prefix_b_to_t+str(i)+post_fix))

ONOFF = 1
count1 = 0
count2 = 0
count3 = 0
count4 = 0
for i in range(36):
    count = 0
    x1 = []
    y1 = []
    p1 = []
    ts1 = []
    for j in range(len(argw[i][2])):
        if (argw[i][2][j] == ONOFF):
            count = count + 1
            x1.append(argw[i][0][j])
            y1.append(argw[i][1][j])
```

```
            p1.append(argw[i][2][j])
            ts1.append(argw[i][3][j])
    x2 = []
    y2 = []
    p2 = []
    ts2 = []
    k = 0
    step = count/89.0
    j = 0.0
    temp = 0
    while(j<count):
        if(temp != int(math.floor(j))):
            x2.append(x1[int(math.floor(j))])
            y2.append(y1[int(math.floor(j))])
            p2.append(p1[int(math.floor(j))])
            ts2.append(ts1[int(math.floor(j))])
            temp = int(math.floor(j))
        j = j+step
        k = k+1


    if(i<9):
        scipy.io.savemat(prefix_l_t_r+str(i+1)+post_fix,{'ts':ts2,
                                            'X':x2,
                                            'Y':y2,
                                            't':p2
                                            })
    if((i<18) and (i>=9)):
        scipy.io.savemat(prefix_r_t_l+str(i-8)+post_fix,{'ts':ts2,
                                    'X':x2,
                                    'Y':y2,
                                    't':p2
                                    })
```

```
    if((i<27) and (i>=18)):
        scipy.io.savemat(prefix_t_to_b+str(i-17)+post_fix,{'ts':ts2,
                                    'X':x2,
                                    'Y':y2,
                                    't':p2
                                    })
    if((i<36) and (i>=27)):
        scipy.io.savemat(prefix_b_to_t+str(i-26)+post_fix,{'ts':ts2,
                                    'X':x2,
                                    'Y':y2,
                                    't':p2
                                    })
```

10.3.4 wrapper2.py:

```
__author__ = "hanyi"
import I_based_model as I_sim
import numpy as np
from types import FunctionType
import random
import scipy.io
import pylab
import matplotlib.pyplot as plt

inhibitory_mode = False
test_mode = False
Normal_training_mode = True

if(Normal_training_mode):
    pre_pop_size = 256
    post_pop_size = 4
    test_STDP = False
    STDP_mode = True
```

```
    inhibitory_spike_mode = False
    allsameweight = False
    self_import = False
else:
    test_STDP = True
    if(inhibitory_mode):
        pre_pop_size = 4
        post_pop_size = 1
        STDP_mode =False
        inhibitory_spike_mode = True
        allsameweight = True
        self_import = False
    if(test_mode):
        pre_pop_size = 256
        post_pop_size = 1
        STDP_mode =True
        inhibitory_spike_mode = False
        allsameweight = False
        self_import = True

#E_syn_weight = np.random.normal(0.2, 0.003,pre_pop_size*post_pop_size).tolist()
E_syn_weight = [0.2,0.1,0.2,0]
I_syn_weight = 15

setup_cond = {
                    'timestep': 1,
                    'min_delay':1,
                    'max_delay':144
}

stdp_param = {
                    'tau_plus': 50.0,
```

```
                    'tau_minus': 60.0,
                    'w_min': 0,
                    'w_max': 1,
                    'A_plus': 0.05,
                    'A_minus': 0.05

}

cell_params_lif = {
                    'cm': 12,
                    'i_offset': 0.0,
                    'tau_m': 110,
                    'tau_refrac': 40.0,
                    'tau_syn_E': 5.0,
                    'tau_syn_I': 10.0,
                    'v_reset': -70.0,
                    'v_rest': -65.0,
                    'v_thresh': -61.0
}

sim1 = I_sim.IF_curr_exp_s(pre_pop_size = pre_pop_size, post_pop_size = post_pop_si
                           E_syn_weight = E_syn_weight, I_syn_weight = I_syn_weigh
                           cell_params_lif = cell_params_lif, setup_cond = setup_c
                           stdp_param = stdp_param,STDP_mode = STDP_mode,inhibitor

#-------------------------------------------------------#
# the following input spike are recorded from DVS
#-------------------------------------------------------#
last_episodes = np.repeat(np.array([0,9,18,27]),1)
order = []
animation_time = 200
```

```
def create_shuffle_seq(num_copy):
    order = np.array([1])
    order_temp = []
    for i in range (4):
        for j in range (4):
            if (i!=j):
                for k in range (4):
                    if(i!=k and j!=k):
                        for z in range (4):
                            if(z!=i and z!=j and z!=k):
                                order_temp.append(9*i+random.randrange(0, 9))
                                order_temp.append(9*j+random.randrange(0, 9))
                                order_temp.append(9*k+random.randrange(0, 9))
                                order_temp.append(9*z+random.randrange(0, 9))
    np_order_temp =  np.array(order_temp)
    #1 in the np.array to initialise the data type to integer
    for i in range (num_copy):
        order = np.concatenate((order, np_order_temp))
    return order


def create_order(num_copy):
    order = np.array([1])
    order_temp = []
    for j in range(num_copy):
        order_temp.append(random.randrange(0, 9))
    for j in range(num_copy):
        order_temp.append(random.randrange(9, 18))
    for j in range(num_copy):
        order_temp.append(random.randrange(18, 27))
    for j in range(num_copy):
        order_temp.append(random.randrange(27, 36))
```

```
    order = np.concatenate((order,order_temp))
    return order

order = create_order(30)
#order = create_order(4)

#order = create_shuffle_seq(12)
#order1 = range(27,36)
'''for i in range(50):
    order = np.concatenate((order,order1))'''

print "_____"
#order = np.concatenate((order,last_episodes))
print order

def get_data(filename):
    dvs_data = scipy.io.loadmat(filename)
    ts = dvs_data['ts'][0]
    ts = (ts - ts[0]) #from us to ms
    x = dvs_data['X'][0]
    y = dvs_data['Y'][0]
    p = dvs_data['t'][0]
    return x,y,p,ts


(x_r1, y_r1, p_r1, ts_r1) = get_data('l_to_r1.mat')
(x_r2, y_r2, p_r2, ts_r2) = get_data('l_to_r2.mat')
(x_r3, y_r3, p_r3, ts_r3) = get_data('l_to_r3.mat')
(x_r4, y_r4, p_r4, ts_r4) = get_data('l_to_r4.mat')
(x_r5, y_r5, p_r5, ts_r5) = get_data('l_to_r5.mat')
(x_r6, y_r6, p_r6, ts_r6) = get_data('l_to_r6.mat')
(x_r7, y_r7, p_r7, ts_r7) = get_data('l_to_r7.mat')
```

```
(x_r8, y_r8, p_r8, ts_r8) = get_data('l_to_r8.mat')
(x_r9, y_r9, p_r9, ts_r9) = get_data('l_to_r9.mat')

(x_l1, y_l1, p_l1, ts_l1) = get_data('r_to_l1.mat')
(x_l2, y_l2, p_l2, ts_l2) = get_data('r_to_l2.mat')
(x_l3, y_l3, p_l3, ts_l3) = get_data('r_to_l3.mat')
(x_l4, y_l4, p_l4, ts_l4) = get_data('r_to_l4.mat')
(x_l5, y_l5, p_l5, ts_l5) = get_data('r_to_l5.mat')
(x_l6, y_l6, p_l6, ts_l6) = get_data('r_to_l6.mat')
(x_l7, y_l7, p_l7, ts_l7) = get_data('r_to_l7.mat')
(x_l8, y_l8, p_l8, ts_l8) = get_data('r_to_l8.mat')
(x_l9, y_l9, p_l9, ts_l9) = get_data('r_to_l9.mat')

(x_b1, y_b1, p_b1, ts_b1) = get_data('t_to_b1.mat')
(x_b2, y_b2, p_b2, ts_b2) = get_data('t_to_b2.mat')
(x_b3, y_b3, p_b3, ts_b3) = get_data('t_to_b3.mat')
(x_b4, y_b4, p_b4, ts_b4) = get_data('t_to_b4.mat')
(x_b5, y_b5, p_b5, ts_b5) = get_data('t_to_b5.mat')
(x_b6, y_b6, p_b6, ts_b6) = get_data('t_to_b6.mat')
(x_b7, y_b7, p_b7, ts_b7) = get_data('t_to_b7.mat')
(x_b8, y_b8, p_b8, ts_b8) = get_data('t_to_b8.mat')
(x_b9, y_b9, p_b9, ts_b9) = get_data('t_to_b9.mat')

(x_t1, y_t1, p_t1, ts_t1) = get_data('b_to_t1.mat')
(x_t2, y_t2, p_t2, ts_t2) = get_data('b_to_t2.mat')
(x_t3, y_t3, p_t3, ts_t3) = get_data('b_to_t3.mat')
(x_t4, y_t4, p_t4, ts_t4) = get_data('b_to_t4.mat')
(x_t5, y_t5, p_t5, ts_t5) = get_data('b_to_t5.mat')
(x_t6, y_t6, p_t6, ts_t6) = get_data('b_to_t6.mat')
(x_t7, y_t7, p_t7, ts_t7) = get_data('b_to_t7.mat')
(x_t8, y_t8, p_t8, ts_t8) = get_data('b_to_t8.mat')
(x_t9, y_t9, p_t9, ts_t9) = get_data('b_to_t9.mat')
```

```
argw =((x_r1, y_r1, p_r1, ts_r1),(x_r2, y_r2, p_r2, ts_r2),
       (x_r3, y_r3, p_r3, ts_r3),(x_r4, y_r4, p_r4, ts_r4),
       (x_r5, y_r5, p_r5, ts_r5),(x_r6, y_r6, p_r6, ts_r6),
       (x_r7, y_r7, p_r7, ts_r7),(x_r8, y_r8, p_r8, ts_r8),
       (x_r9, y_r9, p_r9, ts_r9),(x_l1, y_l1, p_l1, ts_l1),
       (x_l2, y_l2, p_l2, ts_l2),(x_l3, y_l3, p_l3, ts_l3),
       (x_l4, y_l4, p_l4, ts_l4),(x_l5, y_l5, p_l5, ts_l5),
       (x_l6, y_l6, p_l6, ts_l6),(x_l7, y_l7, p_l7, ts_l7),
       (x_l8, y_l8, p_l8, ts_l8),(x_l9, y_l9, p_l9, ts_l9),
       (x_b1, y_b1, p_b1, ts_b1),(x_b2, y_b2, p_b2, ts_b2),
       (x_b3, y_b3, p_b3, ts_b3),(x_b4, y_b4, p_b4, ts_b4),
       (x_b5, y_b5, p_b5, ts_b5),(x_b6, y_b6, p_b6, ts_b6),
       (x_b7, y_b7, p_b7, ts_b7),(x_b8, y_b8, p_b8, ts_b8),
       (x_b9, y_b9, p_b9, ts_b9),(x_t1, y_t1, p_t1, ts_t1),
       (x_t2, y_t2, p_t2, ts_t2),(x_t3, y_t3, p_t3, ts_t3),
       (x_t4, y_t4, p_t4, ts_t4),(x_t5, y_t5, p_t5, ts_t5),
       (x_t6, y_t6, p_t6, ts_t6),(x_t7, y_t7, p_t7, ts_t7),
       (x_t8, y_t8, p_t8, ts_t8),(x_t9, y_t9, p_t9, ts_t9))

def raster_plot_4_dir():
    pylab.figure()
    pylab.xlabel('Time/ms')
    pylab.ylabel('spikes')
    for ii in range(0,4):
        pylab.plot(argw[9*ii][3]+200*(2*ii+1),argw[9*ii][0]+argw[9*ii][1]*16,".")
    pylab.title('Raster Plot of Virtual Retina Neuron Population in 4 Direction')
    pylab.xlim((0, 1800))
    pylab.ylim((0, 270))
    pylab.show()

#raster_plot_4_dir()
```

```
def raster_plot():
    pylab.figure()
    pylab.xlabel('Time/ms')
    pylab.ylabel('spikes')
    for ii in range(0,len(argw)):
        pylab.plot(argw[ii][3]+200*(ii+ii/9),argw[ii][0]+argw[ii][1]*16,".")
    pylab.axvline(1900,0,1,linewidth = 4, color = 'r',alpha = 0.75,linestyle = 'das
    pylab.axvline(3900,0,1,linewidth = 4, color = 'r',alpha = 0.75,linestyle = 'das
    pylab.axvline(5900,0,1,linewidth = 4, color = 'r',alpha = 0.75,linestyle = 'das
    pylab.title('Raster Plot of 36 Neuron Population Training Sets')
    pylab.ylim((0, 270))
    pylab.show()

raster_plot()

NetworkInfo = scipy.io.loadmat('trained_weight4dir.mat')
weights_import = NetworkInfo['trained_weight']
delay = 1

def convert_weights_to_list(matrix, delay):
    def build_list(indices):
        # Extract weights from matrix using indices
        weights = matrix[indices]
        # Build np array of delays
        delays = np.repeat(delay, len(weights))
        # Zip x-y coordinates of non-zero weights with weights and delays
        return zip(indices[0], indices[1], weights, delays)

    # Get indices of non-nan i.e. connected weights
    connected_indices = np.where(~np.isnan(matrix))
    # Return connection lists
```

```
    return build_list(connected_indices)



def BuildTrainingSpike(order,ONOFF):
    complete_Time = []
    for nid in range(0,pre_pop_size):
        SpikeTimes = []
        for tid in range(0,len(order)):
                #print dead_zone_cnt
                temp=[]
                loc = order[tid]
                j = np.repeat(nid,len(argw[loc][1]))
                p = np.repeat(ONOFF,len(argw[loc][1]))
                temp = 200*(2*tid+1)+argw[loc][3][(j%16==argw[loc][0])&
                                                  (j/16==argw[loc][1])&(p==argw[loc][2])]
                if temp.size>0:
                    SpikeTimes = np.concatenate((SpikeTimes,temp))
        if type(SpikeTimes) is not list:
           complete_Time.append(SpikeTimes.tolist())
        else:
            complete_Time.append([])
    return complete_Time

def BuildTrainingSpike_with_noise(order,ONOFF,noise_spikes):
    noise_nid = []
    for i in range(0,len(order)):
        noisetemp = np.random.randint(0,256,noise_spikes)
        noisetemp.sort()
        noisetemp = noisetemp.tolist()
        noise_nid.append(noisetemp)
    #print len(noise_nid)
```

```
    complete_Time = []
    for nid in range(0,pre_pop_size):
        SpikeTimes = []
        for tid in range(0,len(order)):
                #print dead_zone_cnt
                temp=[]
                loc = order[tid]
                j = np.repeat(nid,len(argw[loc][1]))
                p = np.repeat(ONOFF,len(argw[loc][1]))
                temp = 200*(2*tid+1)+argw[loc][3][(j%16==argw[loc][0])&
                                                  (j/16==argw[loc][1])&(p==argw[loc][2])]
                if(nid in noise_nid[tid]):
                    t_noise = 200*(2*tid+1) + np.random.uniform(0,200,1)
                    temp = np.concatenate((temp,t_noise))
                    temp.sort()
                if temp.size>0:
                   SpikeTimes = np.concatenate((SpikeTimes,temp))
        if type(SpikeTimes) is not list:
           complete_Time.append(SpikeTimes.tolist())
        else:
           complete_Time.append([])
    return complete_Time

#in_spike = BuildTrainingSpike(order,1)
#-------------------------------------------------------#

if(test_STDP):
    sim_time = 4000
    if(inhibitory_mode):
        in_spike = [[10],[],[],[]]
    else:
        #in_spike = [[0,10],[5],[20],[13]]
```

```
        #in_spike = [[0,10,100,110],[5,105],[20,120],[13,113]]
        #in_spike = [[0,10,100,110,200,210],[5,105,205],[20,120,220],[13,113,213]]
        #order = [1,2,3,4,5]
        in_spike = BuildTrainingSpike_with_noise(order,1,11)
        sim_time = (2*len(order)+1)*animation_time
else:
    in_spike = BuildTrainingSpike_with_noise(order,1,11)
    sim_time = (2*len(order)+1)*animation_time


sim1.input_spike(in_spike)
#list = [(0, 0, 0.2, 1), (1, 0, 0.1, 1), (2, 0, 0.25, 1), (3, 0, 0.3, 1)]
list = convert_weights_to_list(weights_import, delay)
#sim1.connection_list_converter_normal(self_import = self_import,conn_list = list,
sim1.connection_list_converter_uniform(self_import = self_import,conn_list = list,
#sim1.connection_list_converter_modified_uniform(self_import = self_import,conn_lis
sim1.start_sim(sim_time)
#sim1.display_weight()
sim1.plot_spikes("input",pre_pop_size , "Spike Pattern of Pre-Synaptic Population")
sim1.plot_spikes("output",post_pop_size, "Spike Pattern of Post-Synaptic Population
#sim1.display_membrane_potential("Membrane Potential of one Post-Synaptic Neuron",x
#sim1.display_membrane_potential2("Membrane Potential of Post-Synaptic Neuron",xmin
sim1.Plot_WeightDistribution(256,'Histogram of Trained Weight')
#sim1.Plot_WeightDistribution0(256,'Histogram of Trained Weight of neuron 0')
#sim1.Plot_WeightDistribution1(256,'Histogram of Trained Weight of neuron 1')
```

10.3.5 I_based_model.py:

```
__author__ = "hanyi"
import spynnaker.pyNN as sim
import pylab
import numpy as np
import random
import matplotlib.pyplot as plt
```

```
from pyNN.random import NumpyRNG, RandomDistribution
from types import FunctionType
import scipy.io

class IF_curr_exp_s:
    def __init__(self,pre_pop_size = None,post_pop_size = None,E_syn_weight= None,I
        setup_cond = None,stdp_param = None,STDP_mode = True,inhibitory_spike_mode
        #print "hello"
        #initialise all parameters for simulation
        self.pre_pop_size = pre_pop_size
        self.post_pop_size = post_pop_size
        self.E_syn_weight = E_syn_weight
        self.I_syn_weight = I_syn_weight
        self.cell_params_lif = cell_params_lif
        self.setup_cond = setup_cond
        self.stdp_param = stdp_param
        self.STDP_mode = STDP_mode
        self.inhibitory_spike_mode = inhibitory_spike_mode
        self.allsameweight = allsameweight

    def input_spike(self,in_spike):
        #construct own spike for now
        self.in_spike = in_spike;
        #print self.in_spike

    def connection_list_converter_normal(self,self_import,conn_list,mean,var,delay)
        if(self_import):
            self.conn_list = conn_list
        else:
            conn_list = []
            #self.init_weights = [[None]*self.post_pop_size]*self.pre_pop_size
            for i in range(self.post_pop_size):
```

```
                for j in range(self.pre_pop_size):
                    rand_num = np.random.normal(mean,var,1)
                    if(rand_num[0]<=0):
                        rand_num[0] = -rand_num[0]
                    if(rand_num[0]<0.1):
                        rand_num[0] = 0.1
                    conn_list.append((j,i,rand_num[0],delay))
            self.conn_list = conn_list
            '''for k in range(self.post_pop_size*self.pre_pop_size):
                self.init_weights[j][i] ='''


    def connection_list_converter_uniform(self,self_import,conn_list,min,max,delay):
        print "-------------------------------------------------------------------------------"
        print "uniform distribution used"
        print "-------------------------------------------------------------------------------"

        if(self_import):
            self.conn_list = conn_list
        else:
            conn_list = []
            #self.init_weights = [[None]*self.post_pop_size]*self.pre_pop_size
            for i in range(self.post_pop_size):
                for j in range(self.pre_pop_size):
                    rand_num = np.random.uniform(min,max,1)
                    if(rand_num[0]<0.05):
                        rand_num[0] = 0.05
                    conn_list.append((j,i,rand_num[0],delay))
            self.conn_list = conn_list
            '''for k in range(self.post_pop_size*self.pre_pop_size):
                self.init_weights[j][i] ='''
            #print self.conn_list
```

```
def connection_list_converter_modified_uniform(self,self_import,conn_list,min_h
    if(self_import):
        self.conn_list = conn_list
    else:
        conn_list = []
        for i in range(self.post_pop_size):
            for j in range(self.pre_pop_size):
                if(i<5):
                    #x[0:4] y[0:4]
                    if((j/16)<5 and (j%16)<5):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=5) and i<10):
                    #x[5:9] y[0:4]
                    if((j/16)>=5 and (j/16)<10  and (j%16)<5):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=10) and i<15):
                    #x[10:15] y[0:4]
                    if((j/16)>=10 and (j/16)<16 and (j%16)<5):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=15) and i<20):
                    #x[0:4] y[5:9]
                    if((j/16)<5 and (j%16)>=5 and (j%16)<10):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
```

```
                if((i>=20) and i<25):
                    #x[5:9] y[5:9]
                    if((j/16)>=5 and (j/16)<10 and (j%16)>=5 and (j%16)<10):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=25) and i<30):
                    #x[10:15] y[5:9]
                    if((j/16)>=10 and (j/16)<16 and (j%16)>=5 and (j%16)<10):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=30) and i<35):
                    #x[0:4] y[10:15]
                    if((j/16)<5 and (j%16)>=10 and (j%16)<16):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=35) and i<40):
                    #x[5:9] y[10:15]
                    if((j/16)>=5 and (j/16)<10 and (j%16)>=10 and (j%16)<16):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                if((i>=40) and i<45):
                    #x[10:15] y[10:15]
                    if((j/16)>=10 and (j/16)<16 and (j%16)>=10 and (j%16)<16):
                        rand_num = np.random.uniform(min_h,max_h,1)
                    else:
                        rand_num = np.random.uniform(min_l,max_l,1)
                conn_list.append((j,i,rand_num[0],delay))
        self.conn_list = conn_list
```

```
def start_sim(self,sim_time):
    #simulation setup
    self.simtime = sim_time
    sim.setup(timestep=self.setup_cond["timestep"], min_delay=self.setup_cond["
    #initialise the neuron population
    spikeArrayOn = {'spike_times': self.in_spike}
    pre_pop = sim.Population(self.pre_pop_size, sim.SpikeSourceArray,
                      spikeArrayOn, label='inputSpikes_On')
    post_pop= sim.Population(self.post_pop_size,sim.IF_curr_exp,
                       self.cell_params_lif, label='post_1')
    stdp_model = sim.STDPMechanism(timing_dependence=sim.SpikePairRule(tau_plus
                                                        tau_minus= self.stdp_param["tau
                                                        nearest=True),
                                                weight_dependence=sim.MultiplicativeWeightD
                                                                w_max= self.stdp_para
                                                                A_plus= self.stdp_pa
                                                                A_minus= self.stdp_p
    #initialise connectiviity of neurons
    #exitatory connection between pre-synaptic and post-synaptic neuron populat
    if(self.inhibitory_spike_mode):
        connectionsOn  = sim.Projection(pre_pop, post_pop, sim.AllToAllConnecto
                                            allow_self_connections=False),
                                            target='inhibitory')
    else:
        if(self.STDP_mode):
            if(self.allsameweight):
                connectionsOn = sim.Projection(pre_pop, post_pop,
                                                sim.AllToAllConnector(weights = self
                                                 synapse_dynamics=sim.SynapseDynamic
```

```
                else:
                    connectionsOn = sim.Projection(pre_pop, post_pop,
                                                   sim.FromListConnector(self.conn_list
                                                    synapse_dynamics=sim.SynapseDynamic
            else:
                if(self.allsameweight):
                    connectionsOn  = sim.Projection(pre_pop, post_pop, sim.AllToAll
                                                                       target='ex
                else:
                    connectionsOn  = sim.Projection(pre_pop, post_pop, sim.FromList
                                                                       target='ex
                #sim.Projection.setWeights(self.E_syn_weight)

        #inhibitory between the neurons post-synaptic neuron population
        connection_I  = sim.Projection(post_pop, post_pop, sim.AllToAllConnector(w
                                                           allow_self_connections
                                                           target='inhibitory')

        pre_pop.record()
        post_pop.record()
        post_pop.record_v()
        sim.run(self.simtime)
        self.pre_spikes = pre_pop.getSpikes(compatible_output=True)
        self.post_spikes = post_pop.getSpikes(compatible_output=True)
        self.post_spikes_v = post_pop.get_v(compatible_output=True)
        self.trained_weights = connectionsOn.getWeights(format='array')
        sim.end()
        #print self.conn_list
        #print self.trained_weights
        '''scipy.io.savemat('trained_weight.mat',{'initial_weight':self.init_weight
____________________________________________________'trained_weight':self.trained_weight
_____________________________________________})'''
        scipy.io.savemat('trained_weight.mat',{'trained_weight':self.trained_weight
```

```
                                    })

    def display_weight(self):
        for i in range(self.post_pop_size):
            print ([x[i] for x in self.weights_trained])

    def plot_spikes(self, spike_type, size, title):
        if (spike_type == "input"):
            spikes = self.pre_spikes
        if (spike_type == "output"):
            spikes = self.post_spikes
        #print spikes
        if spikes is not None:
            pylab.figure()
            ax = plt.subplot(111, xlabel='Time/ms', ylabel='Neruons #',
                             title=title)
            pylab.xlim((0, self.simtime))
            pylab.ylim((-0.5, size-0.5))
            lines = pylab.plot([i[1] for i in spikes], [i[0] for i in spikes],".")
            pylab.axvline(32500,0,1,linewidth = 4, color = 'c',alpha = 0.75,linesty
            pylab.axvline(64500,0,1,linewidth = 4, color = 'c',alpha = 0.75,linesty
            pylab.axvline(96500,0,1,linewidth = 4, color = 'c',alpha = 0.75,linesty
            pylab.setp(lines,markersize=10,color='r')
            for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                 ax.get_xticklabels() + ax.get_yticklabels()):
                 item.set_fontsize(20)
            pylab.show()
        else:
            print "No spikes received"


    def Plot_WeightDistribution(self,bin_num,title):
```

```
        hist,bins = np.histogram(self.trained_weights,bins=bin_num)
        center = (bins[:-1]+bins[1:])/2
        width = (bins[1]-bins[0])*0.7
        ax = pylab.subplot(111,xlabel='Weight',title =title)
        plt.bar(center,hist,align='center',width =width)
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                     ax.get_xticklabels() + ax.get_yticklabels()):
            item.set_fontsize(15)
        plt.show()

    def Plot_WeightDistribution0(self,bin_num,title):
        hist,bins = np.histogram([i[0] for i in self.trained_weights],bins=bin_num)
        center = (bins[:-1]+bins[1:])/2
        width = (bins[1]-bins[0])*0.7
        ax = pylab.subplot(111,xlabel='Weight',title =title)
        plt.bar(center,hist,align='center',width =width)
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                     ax.get_xticklabels() + ax.get_yticklabels()):
            item.set_fontsize(15)
        plt.show()

    def Plot_WeightDistribution1(self,bin_num,title):
        hist,bins = np.histogram([i[1] for i in self.trained_weights],bins=bin_num)
        center = (bins[:-1]+bins[1:])/2
        width = (bins[1]-bins[0])*0.7
        ax = pylab.subplot(111,xlabel='Weight',title =title)
        plt.bar(center,hist,align='center',width =width)
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                     ax.get_xticklabels() + ax.get_yticklabels()):
            item.set_fontsize(15)
        plt.show()
```

```
def display_membrane_potential(self,title,xmin=0,xmax=50,ymin=-70,ymax=-63):
    post_spikes_v = self.post_spikes_v
    #print post_spikes_v
    if post_spikes_v is not None:
        pylab.figure()
        ax = plt.subplot(111, xlabel='Time/ms', ylabel='Membrane Potential/V',
                         title=title)
        pylab.xlim((xmin, xmax))
        pylab.ylim((ymin, ymax))
        pylab.plot([i[1] for i in post_spikes_v], [i[2] for i in post_spikes_v]
        #pylab.setp(lines,markersize=10,color='r')
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
             ax.get_xticklabels() + ax.get_yticklabels()):
             item.set_fontsize(20)
        pylab.show()
    else:
        print "No spikes received"

def display_membrane_potential2(self,title,xmin=0,xmax=50,ymin=-70,ymax=-63):
    post_spikes_v = self.post_spikes_v
    #print post_spikes_v
    if post_spikes_v is not None:
        pylab.figure()
        ax = plt.subplot(111, xlabel='Time/ms', ylabel='Membrane Potential/V',
                         title=title)
        pylab.xlim((xmin, xmax))
        pylab.ylim((ymin, ymax))
        time = []
        potential = []
        cur_neuron_id = 0
        ii = 0
        while(ii!=len(post_spikes_v)):
```

```
                if(post_spikes_v[ii][0]==cur_neuron_id):
                    time.append(post_spikes_v[ii][1])
                    potential.append(post_spikes_v[ii][2])
                else:
                    pylab.plot(time,potential)
                    cur_neuron_id = cur_neuron_id+1
                    time = []
                    potential = []
                    time.append(post_spikes_v[ii][1])
                    potential.append(post_spikes_v[ii][2])
                ii =ii+1
            #pylab.plot([i[1] for i in post_spikes_v], [i[2] for i in post_spikes_v
            #pylab.setp(lines,markersize=10,color='r')
            for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                 ax.get_xticklabels() + ax.get_yticklabels()):
                 item.set_fontsize(20)
            pylab.show()
        else:
            print "No spikes received"
```

10.3.6 Post_training.py:

```
__author__ = 'hanyi'
import spynnaker.pyNN as sim
import pylab
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
import time
import sys


cell_params_lif = {
```

```
                    'cm': 12,
                    'i_offset': 0.0,
                    'tau_m': 110,
                    'tau_refrac': 40.0,
                    'tau_syn_E': 5.0,
                    'tau_syn_I': 10.0,
                    'v_reset': -70.0,
                    'v_rest': -65.0,
                    'v_thresh': -61.0
}

NetworkInfo = scipy.io.loadmat('trained_weight_65.mat')
weights_import = NetworkInfo['trained_weight']

def convert_weights_to_list(matrix, delay):
    def build_list(indices):
        # Extract weights from matrix using indices
        weights = matrix[indices]
        # Build np array of delays
        delays = np.repeat(delay, len(weights))
        # Zip x-y coordinates of non-zero weights with weights and delays
        return zip(indices[0], indices[1], weights, delays)

    # Get indices of non-nan i.e. connected weights
    connected_indices = np.where(~np.isnan(matrix))
    # Return connection lists
    return build_list(connected_indices)

def Plot_WeightDistribution(weight,bin_num,title):
    hist,bins = np.histogram(weight,bins=bin_num)
    center = (bins[:-1]+bins[1:])/2
    width = (bins[1]-bins[0])*0.7
```

```
    plt.bar(center,hist,align='center',width =width)
    plt.xlabel('Weight')
    plt.title(title)
    plt.show()
#Plot_WeightDistribution(weights_import,200,'trained weight')

sim.setup(timestep=1, min_delay=1, max_delay=144)
synapses_to_spike = 1
delay = 1
prepop_size = 256
postpop_size = 4
animation_time = 200
episode = 200
order = np.array(range(36))
test_order = np.array([0,1,2,3])
simtime = len(order)*animation_time+8*animation_time

def concatenate_time(time,iter):
    temp_time = []
    spike_time= []
    for kk in range(0,iter):
        spike_time = np.concatenate((temp_time,time+kk*animation_time*4))
        temp_time = spike_time
    return temp_time
#Train_time =  concatenate_time(firing_time,len(order)/4)
#NeuronID = np.tile(NetworkInfo['NeuronID'][0],len(order)/4)

def get_data(filename):
    dvs_data = scipy.io.loadmat(filename)
    ts = dvs_data['ts'][0]
    ts = (ts - ts[0]) #from ns to ms
    x = dvs_data['X'][0]
```

```
    y = dvs_data['Y'][0]
    p = dvs_data['t'][0]
    return x,y,p,ts

def ReadSpikeTime(NeuronID,x,y,ts,p,ONOFF):
    timeTuple=[]
    for idx in range(0,len(x)):
        if NeuronID == (x[idx]+y[idx]*16) and p[idx]==ONOFF:
            timeTuple.append(ts[idx])
    return timeTuple

def BuildSpike(x,y,ts,p,ONOFF):
    SpikeTimes = []
    for i in range(0,prepop_size):
        SpikeTimes.append(ReadSpikeTime(i,x,y,ts,p,ONOFF))
    return SpikeTimes

def BuildTrainingSpike(order,ONOFF):
    complete_Time = []
    for nid in range(0,prepop_size):
        SpikeTimes = []
        for tid in range(0,len(order)):
                dead_zone_cnt = int(tid/9)
                #print dead_zone_cnt
                temp=[]
                loc = order[tid]
                j = np.repeat(nid,len(argw[loc][1]))
                p = np.repeat(ONOFF,len(argw[loc][1]))
                temp = 200*(tid+2*dead_zone_cnt)+argw[loc][3][(j%16==argw[loc][0])&
                                                    (j/16==argw[loc][1])&(p==argw[loc][2])]
                if temp.size>0:
                    SpikeTimes = np.concatenate((SpikeTimes,temp))
```

```
        if type(SpikeTimes) is not list:
            complete_Time.append(SpikeTimes.tolist())
        else:
            complete_Time.append([])
    return complete_Time

def compare_spikes(spikes, title):
    if spikes is not None:
        pylab.figure()
        ax = plt.subplot(111, xlabel='Time/ms', ylabel='Neruons #', title=title)
        pylab.xlim((0, simtime))
        pylab.ylim((0, postpop_size+2))
        line1 = pylab.plot([i[1] for i in spikes], [i[0] for i in spikes],
                           'r|',label='post-train spikes')
        line2 = pylab.plot(Train_time,NeuronID,'b|',label='trained spikes')
        pylab.setp(line1,markersize=10,linewidth=25)
        pylab.setp(line2,markersize=10,linewidth=25)
        pylab.legend()
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
             ax.get_xticklabels() + ax.get_yticklabels()):
             item.set_fontsize(20)
        pylab.show()
    else:
        print "No spikes received"

def plot_spikes(spikes, title):

    if spikes is not None:
        pylab.figure()
        ax = plt.subplot(111, xlabel='Time/ms', ylabel='Neruons #', title=title)
        pylab.xlim((0, simtime))
        pylab.ylim((-0.5, postpop_size-0.5))
```

```
        lines = pylab.plot([i[1] for i in spikes], [i[0] for i in spikes],".")
        pylab.axvline(2000,linewidth = 4, color = 'c',alpha = 0.75,linestyle = 'das
        pylab.axvline(4200,0,1,linewidth = 4, color = 'c',alpha = 0.75,linestyle =
        pylab.axvline(6400,0,1,linewidth = 4, color = 'c',alpha = 0.75,linestyle =
        '''pylab.axvspan(1800,2200, facecolor='b', alpha=0.5)
        pylab.axvspan(4000,4400, facecolor='b', alpha=0.5)
        pylab.axvspan(6200,6600, facecolor='b', alpha=0.5)
        pylab.axvspan(8400,8800, facecolor='b', alpha=0.5)'''
        pylab.setp(lines,markersize=10,color='r')
        for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
             ax.get_xticklabels() + ax.get_yticklabels()):
             item.set_fontsize(20)
        pylab.show()

    else:
        print "No spikes received"
'''(x_r, y_r, p_r, ts_r) = get_data('downsampled_left_to_right_2.mat')
(x_l, y_l, p_l, ts_l) = get_data('downsampled_right_to_left_2.mat')
(x_d, y_d, p_d, ts_d) = get_data('downsampled_top_to_bottom_2.mat')
(x_u, y_u, p_u, ts_u) = get_data('downsampled_bottom_to_top_2.mat')'''
(x_r1, y_r1, p_r1, ts_r1) = get_data('l_to_r1.mat')
(x_r2, y_r2, p_r2, ts_r2) = get_data('l_to_r2.mat')
(x_r3, y_r3, p_r3, ts_r3) = get_data('l_to_r3.mat')
(x_r4, y_r4, p_r4, ts_r4) = get_data('l_to_r4.mat')
(x_r5, y_r5, p_r5, ts_r5) = get_data('l_to_r5.mat')
(x_r6, y_r6, p_r6, ts_r6) = get_data('l_to_r6.mat')
(x_r7, y_r7, p_r7, ts_r7) = get_data('l_to_r7.mat')
(x_r8, y_r8, p_r8, ts_r8) = get_data('l_to_r8.mat')
(x_r9, y_r9, p_r9, ts_r9) = get_data('l_to_r9.mat')

(x_l1, y_l1, p_l1, ts_l1) = get_data('r_to_l1.mat')
(x_l2, y_l2, p_l2, ts_l2) = get_data('r_to_l2.mat')
```

```
(x_l3, y_l3, p_l3, ts_l3) = get_data('r_to_l3.mat')
(x_l4, y_l4, p_l4, ts_l4) = get_data('r_to_l4.mat')
(x_l5, y_l5, p_l5, ts_l5) = get_data('r_to_l5.mat')
(x_l6, y_l6, p_l6, ts_l6) = get_data('r_to_l6.mat')
(x_l7, y_l7, p_l7, ts_l7) = get_data('r_to_l7.mat')
(x_l8, y_l8, p_l8, ts_l8) = get_data('r_to_l8.mat')
(x_l9, y_l9, p_l9, ts_l9) = get_data('r_to_l9.mat')

(x_b1, y_b1, p_b1, ts_b1) = get_data('t_to_b1.mat')
(x_b2, y_b2, p_b2, ts_b2) = get_data('t_to_b2.mat')
(x_b3, y_b3, p_b3, ts_b3) = get_data('t_to_b3.mat')
(x_b4, y_b4, p_b4, ts_b4) = get_data('t_to_b4.mat')
(x_b5, y_b5, p_b5, ts_b5) = get_data('t_to_b5.mat')
(x_b6, y_b6, p_b6, ts_b6) = get_data('t_to_b6.mat')
(x_b7, y_b7, p_b7, ts_b7) = get_data('t_to_b7.mat')
(x_b8, y_b8, p_b8, ts_b8) = get_data('t_to_b8.mat')
(x_b9, y_b9, p_b9, ts_b9) = get_data('t_to_b9.mat')

(x_t1, y_t1, p_t1, ts_t1) = get_data('b_to_t1.mat')
(x_t2, y_t2, p_t2, ts_t2) = get_data('b_to_t2.mat')
(x_t3, y_t3, p_t3, ts_t3) = get_data('b_to_t3.mat')
(x_t4, y_t4, p_t4, ts_t4) = get_data('b_to_t4.mat')
(x_t5, y_t5, p_t5, ts_t5) = get_data('b_to_t5.mat')
(x_t6, y_t6, p_t6, ts_t6) = get_data('b_to_t6.mat')
(x_t7, y_t7, p_t7, ts_t7) = get_data('b_to_t7.mat')
(x_t8, y_t8, p_t8, ts_t8) = get_data('b_to_t8.mat')
(x_t9, y_t9, p_t9, ts_t9) = get_data('b_to_t9.mat')

argw =((x_r1, y_r1, p_r1, ts_r1),(x_r2, y_r2, p_r2, ts_r2),
       (x_r3, y_r3, p_r3, ts_r3),(x_r4, y_r4, p_r4, ts_r4),
       (x_r5, y_r5, p_r5, ts_r5),(x_r6, y_r6, p_r6, ts_r6),
       (x_r7, y_r7, p_r7, ts_r7),(x_r8, y_r8, p_r8, ts_r8),
```

```
        (x_r9, y_r9, p_r9, ts_r9),(x_l1, y_l1, p_l1, ts_l1),
        (x_l2, y_l2, p_l2, ts_l2),(x_l3, y_l3, p_l3, ts_l3),
        (x_l4, y_l4, p_l4, ts_l4),(x_l5, y_l5, p_l5, ts_l5),
        (x_l6, y_l6, p_l6, ts_l6),(x_l7, y_l7, p_l7, ts_l7),
        (x_l8, y_l8, p_l8, ts_l8),(x_l9, y_l9, p_l9, ts_l9),
        (x_b1, y_b1, p_b1, ts_b1),(x_b2, y_b2, p_b2, ts_b2),
        (x_b3, y_b3, p_b3, ts_b3),(x_b4, y_b4, p_b4, ts_b4),
        (x_b5, y_b5, p_b5, ts_b5),(x_b6, y_b6, p_b6, ts_b6),
        (x_b7, y_b7, p_b7, ts_b7),(x_b8, y_b8, p_b8, ts_b8),
        (x_b9, y_b9, p_b9, ts_b9),(x_t1, y_t1, p_t1, ts_t1),
        (x_t2, y_t2, p_t2, ts_t2),(x_t3, y_t3, p_t3, ts_t3),
        (x_t4, y_t4, p_t4, ts_t4),(x_t5, y_t5, p_t5, ts_t5),
        (x_t6, y_t6, p_t6, ts_t6),(x_t7, y_t7, p_t7, ts_t7),
        (x_t8, y_t8, p_t8, ts_t8),(x_t9, y_t9, p_t9, ts_t9))

#Let us only use the ON events
TrianSpikeON = BuildTrainingSpike(order,1)
#print TrianSpikeON
spikeArrayOn = {'spike_times': TrianSpikeON}
ON_pop = sim.Population(prepop_size, sim.SpikeSourceArray, spikeArrayOn,
                        label='inputSpikes_On')
post_pop= sim.Population(postpop_size,sim.IF_curr_exp, cell_params_lif,
                         label='post_1')


connectionsOn = sim.Projection(ON_pop, post_pop, sim.FromListConnector(
    convert_weights_to_list(weights_import, delay)))
#inhibitory between the neurons
connection_I  = sim.Projection(post_pop, post_pop, sim.AllToAllConnector(
    weights = 15,delays=1,allow_self_connections=False), target='inhibitory')
post_pop.record()
```

```
sim.run(simtime)

# == Get the Simulated Data ==========================================
post_spikes = post_pop.getSpikes(compatible_output=True)
sim.end()

def GetFiringPattern(spike,low,high):
    spikeT = np.transpose(spike)
    time_stamp = spikeT[1]
    target_index = ((time_stamp-low)>=0) & ((time_stamp-high)<0)
    firingTable = np.unique(spikeT[0][target_index])
    firingRate = len(np.unique(spikeT[0][target_index]))
    return firingRate,firingTable

sec_layer_firing_rate = []
sec_layer_firing_table= []
for jj in range(0,44):
    rate, table =  GetFiringPattern(post_spikes,200*jj,200*(jj+1))
    print table,jj
    sec_layer_firing_rate.append(rate)
    sec_layer_firing_table.append(table)

scipy.io.savemat('trained_firing_info.mat',{'firing_rate':sec_layer_firing_rate,
                                            'firing_table':sec_layer_firing_table})

def check_uniq(p_subset,p_superset):
    flag = 0;#when flag is 1, it means two sets are not subset and superset
    j = 0
    i = 0
    while(i != len(p_subset)):
        if(p_subset[i]==p_superset[j]):
            i += 1
```

```
        else:
            j += 1
            if(j==len(p_superset)):
                flag = 1
                break

    if(flag == 0 ):
        print "there exists subset"
    return flag

def sup_sub_checker(firing_table):
    #find the len order
    flag = 1
    for i in range (4):
        for j in range (4):
            if(len(firing_table[i])<=len(firing_table[j]) and i !=j ):
                flag = flag & check_uniq(firing_table[i],firing_table[j])
    if(flag == 1):
        print "all clear"
sup_sub_checker(sec_layer_firing_table)
def plot_spike_histogram(spikes, bin_num,title):
    hist,bins = np.histogram([i[0] for i in spikes],bins=bin_num)
    center = (bins[:-1]+bins[1:])/2
    width = (bins[1]-bins[0])*0.7
    ax = pylab.subplot(111,xlabel='Neuron ID',ylabel = 'spiking times',title =title
    plt.bar(center,hist,align='center',width =width)
    for item in ([ax.title, ax.xaxis.label, ax.yaxis.label] +
                 ax.get_xticklabels() + ax.get_yticklabels()):
        item.set_fontsize(15)
    plt.show()
plot_spike_histogram(post_spikes,postpop_size,"Post-Synaptic Neuron Spiking Rate ")
plot_spikes(post_spikes, "Spike Pattern of Post-Synaptic Population")
```