Engineering

School of Physics, Engineering and Technology

BEng Project Report

2022/23

Student Name: Benjamin McCann

Project Title: Neuromorphic Computing: Modelling the behaviour of the Primary visual cortex with Spiking Neural Networks

Supervisors: Dr David Halliday & Professor Gavin Kearney

Department of Electronic Engineering
University of York
Heslington
York
YO10 5DD

Acknowledgements

I want to thank my Mum and Great Uncle for always listening to any stresses I had and for the constant kindness and support.

I would like to thank David Halliday for helping me to understand difficult content throughout the project and for the advice that helped me to keep my project on track. Including help with specifics like the cross-correlation function.

I would also like to thank Gavin Kearney for advice and feedback from weekly emails and the initial report.

Table of contents

***Abstract*— This paper experimentally reveals the behaviour of a software model, seen in the paper, of the Primary visual cortex. This was made using Spiking Neural Networks to create a bio-inspired design that benefits from behaviours seen in the brain. The software's complexity and size is scaled down from other existing models without losing biological plausibility. Therefore, it can be used in modelling and engineering purposes. It fills a gap in the literature for the need for more power efficient, lower computationally complex and applicable bio-inspired software.**

The behaviours replicated theory of neural coding seeing transmission in the spike rate of neurons. Other behaviours seen were cortical stabilising, Hebbian learning and inhibitory and excitatory behaviour of nodes within the Cortical microcircuit. It also shows an application of the model with a cheaper representation of event-based vision, without using expensive neuromorphic cameras. Increasing accessibility and showcasing the applicability of bio-inspired software.

1. Introduction

This paper focuses on the use of Spiking neural networks (SNNs). This third generation of network builds on the previous Artificial neural networks (ANN's), by modelling nodes closely to neurons in the brain. This achieves better modelling capability for fields such as computational neuroscience and bio-inspired systems. For example, the brain functions with 20 Watts [1] of power by operating asynchronously. This asynchronous behaviour is beneficial as systems only respond when there is a stimulus, which saves energy from not having the network constantly processing. This is attractive for engineering power efficient devices such as IBM's True North chip [2].

It is power efficient, computationally powerful (as processing is done in parallel at singular neurons) and has the attribute of adaptation, the ability to learn to solve issues. This isn't possible with classic Von-Neumann architecture.

In systems like True North, adaptation and emergence are realized due to the unsupervised learning capabilities of SNN's [3]. These SNN algorithms solve problems that classical algorithms can't like NP-hard problems as shown by Jonke, Habenschuss, & Maass, 2016 [4]. The networks are also inherently noise resilient [3].

The project aims to create a bioinspired model that mimics behaviour of the primary visual cortex using SNN's. It addresses a gap in the literature for realistic, lower complexity and efficient models for multiple use cases. It will use characteristics of SNN's like adaptation to show how they benefit software models by allowing

them to adapt and learn to events to achieve desirable results.

It will model at least five nodes of the primary cortex using the Izhikevich model [5] with stochastic noise to mimic neuronal behaviour. This will include inhibitory and excitatory behaviour to see how this could affect the systems performance.

The software's behaviour will be illustrated using various inputs to showcase how information is processed in the brain. This enhances the literature around the Primary cortex and enables exploration of uses of bio-inspired vision systems. A specific application will be explored in safety control systems.

This demonstrates the advantages of event-based vision. The concept of event-based vision is a technology that relies on event cameras. These are sensors that take images asynchronously at a fixed rate and register events that encode the time, position, and change in brightness of the environment.

Event cameras have benefits over conventional cameras such as a higher pixel bandwidth causing lower motion blur, low power consumption, and lower latency [6]. This project sets the groundwork for creating stable systems that could interface with such technologies.

2. Literature Review

2.1 Background biology

This section outlines the structure and properties of the brain. The brain is an interconnected mass of approximately 100 billion neuron's [7] which all have their individual roles. These neurons are fundamental units of the brain that are less complex than processing units seen in computers. The brain employs Parallel Distributed Processing (PDP), where information is distributed across the network and processing by these single units is done in parallel to others [8].

This is what allows the brain to have processing abilities not present by singular neurons. It is an example of an emergent property. The term emergence is defined as a characteristic of a system that cannot be explained by the environment in which it is in. It has no prior programming or information to the system about its environment [9]. This attribute can be positive as it helps systems adapt in ways impossible to see beforehand and were not intended by its designers, Johnson [10]. This can also be detrimental to systems [10], as it could cause unwanted behaviours that make it unfit for its original purpose.

Finally, the brain is an adaptive system. This is the property of the brain that allows humans to adapt their behaviour from experience to

situations [11]. It is important and allows for bioinspired technologies like neural networks to be applied to engineering problems that classical algorithms can't. The next section will cover the structure of neurons that allows these behaviours to be possible.

2.1.1 Basic structure of neurons

The structure of neurons varies depending on their function. They're separated into three classes shown in Figure 1. Despite classification all neurons share characteristics. They have a cell body called the soma where the central processing unit of the neuron is. The long body sections leading away from here are called axons. Their ends are called axon terminals. These terminals (neurites) have small branching structures called dendrites [12].

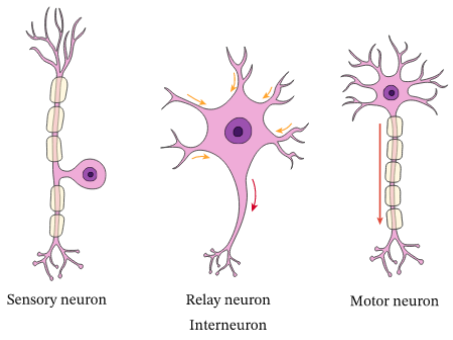


Figure 1 Classifications of neurons [13]. Three classes, sensory, interneuron and motor neuron.

These neurons connect to others via synapses, which are junctions between their ends [12]. The neuron before the gap is called presynaptic and the other postsynaptic. Communication between neurons is a chemical process. A neurotransmitter (chemical signal) is released once there is an action potential, depicted in Figure 2. Once a threshold of neurotransmitter is met a spike generates in the postsynaptic neuron. These neurotransmitters can be excitatory, enhancing the creation of action potentials or inhibitory, degrading the generation of potentials [14][15].

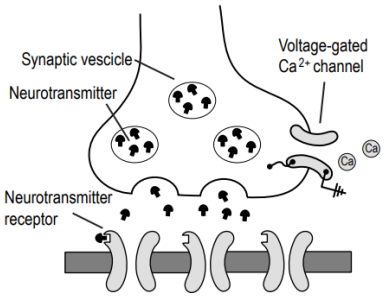


Figure 2 Chemical signalling between neurons [12].

2.1.2. Action Potentials

The membrane of the neuron encompasses channels to facilitate ion transfer in and out of the cell. There are around 10 types of ion channel [16] but voltage gated Sodium and Potassium channels are critical to mention. The opening of these channels depends on the membrane potential [17]. These channels take part in the active transport of Sodium (Na+) and Potassium (K+) ions, which generate action potentials [16]. An action potential is an electrical impulse caused by a fast change in the neuron's membrane potential [16]. The membrane potential is the potential difference between the inside and outside of the neuron [16].

When there is no action potential, this potential is -70mV. This is called the resting potential and is maintained by a Sodium/Potassium pump, until

an action potential occurs [17][18]. An Ion pump is an ion channel that transports ions against the concentration gradient [17] from low to high concentration.

When an action potential occurs, depolarisation happens (change in potential). This is facilitated through Sodium channels allowing positive ions into the neuron. Initially it is slow but as potential grows more channels open [18] strengthening the response. At -55mV the threshold voltage is reached [18] and the steep peak in Figure 3 is realised.

Approximately at +30mV, Sodium channels close and repolarisation begins. The positive potential opens Potassium channels causing the potential to become negative [18].

These channels are slow to close and cause hyperpolarisation [18], a dip of the neurons potential below its resting value to approximately -80mV [19]. The cell is restored to its resting potential by leakage and ion pump channels [16]. These leakage channels are always open and maintain the resting potential [16].

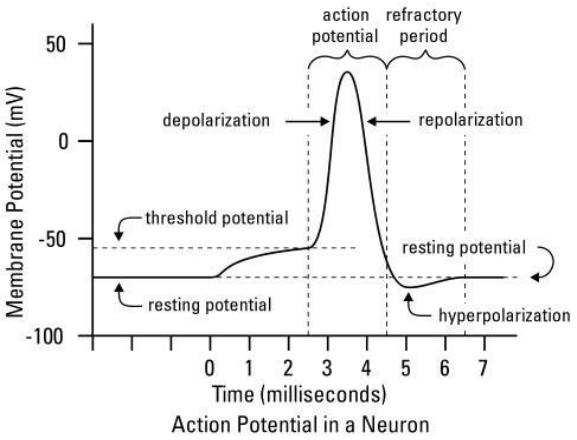


Figure 3 Shape of an Action potential [20]. Initially at resting of -70 mV and after threshold is met, depolarisation occurs and the potential builds to +30mV. The potential repolarises and overshoots to -80mV. It then returns to its resting potential.

2.1.3. Neural code

This section discusses how information is encoded into action potentials. It is not known exclusively but there are two main theories [19]. The first is the encoding of information in the rate at which spikes are fired. The second is that information is encoded in the exact timing at which spikes occur [12].

The idea of mean firing rate was described by E.D Adrian [21] in 1926 and showed the relationship between the magnitude of the input and the firing rate of the response in experiments on frog muscles. The experiment was ground-breaking but there is a question about the ethics of the study. As cells were dissected from living animals.

However, the information from the exact timing of the spikes isn't shown here. The concept is criticized further by researchers such as Bialek [22] who point out that organisms rarely compute

averages, and that this information is useless unless it could estimate the firing rate of its neurons.

The consensus is that the brain uses a form of the two coding schemes, but more understanding is needed [19]. In this study, the definition of the exact timing of when spikes occur will be used. As it links to how current neuron model's function.

2.2. Leaky-Integrate & Fire model.

The Leaky-Integrate and fire model is a less realistic model of a neuron. It is used in engineering and modelling applications as its computational complexity is reduced when compared to models such as Hodgkin Huxley.

The model gets its name as the biological membrane is permeable to chemicals as described in 2.1.2. This makes it an imperfect insulator and is integrated by a "leaky" integrator. Here the Euler integration method is used.

The model is defined by one standard differential equation, discussed further in [12].

$$\tau_m \frac{\mathrm{dV}}{\mathrm{dt}} = (E_l - \mathrm{v}) + \mathrm{R}I_{ext} \qquad (1)$$

Here a spike is characterised by its firing time and is generated at time t when a threshold value is met. This fixed threshold of -55mV is biologically implausible and is partly why the model is regarded as unrealistic. Once the neuron spikes its potential is reset to the resting potential, described in a limit as $\lim_{\delta \to \infty} v(t+\delta) = v_{resting}$ [12]. This behaviour can be seen in Figure 4 where the node spikes and is then reset instantly to -70mV. This is impossible in real neurons. Furthermore, LIF only shows pre threshold behaviour and doesn't show the actual spike.

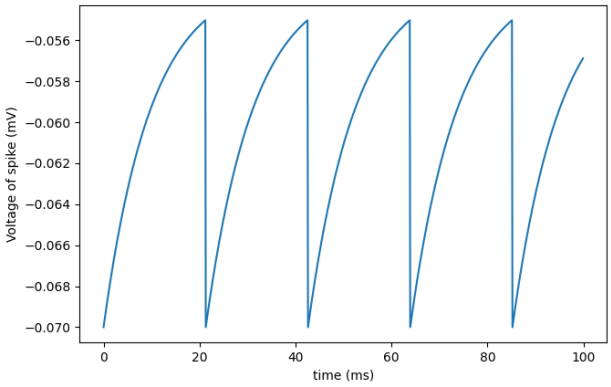


Figure 4 Shape of LIF Action potential. Shows node starts at resting and sharply increases to the threshold value of -56mV. It then instantly resets to -70mV. This instant reset doesn't match realistic potentials seen in Figure 3.

2.3. Izhikevich model

The Izhikevich model is more complex than LIF as it uses two differential equations, improving upon LIF's lack of detail in measuring the membrane potential, increasing biological plausibility.

The first equation denotes how the neuron responds to input current and the second equation describes a variable u called the recovery variable. The recovery variable relates to the activation of K+ channels and deactivation of Na channels and provides negative feedback to the variable v [5][23].

$$\frac{dv(t)}{dt} = 0.04v^2(t) + 5v(t) + 140 - u + I(t) \quad (2)$$

$$\frac{du(t)}{dt} = a(bv - u) \quad (3)$$

With reset conditions v (v > 30) =c and

u(v > 30)= u – d

The V is voltage in mV, and I is the external current in mA driving the neuron. There are four parameters a, b, c, and d which are set to simulate several types of neurons. The a sets the time scale of the recovery variable, b is the sensitivity of the recovery variable to fluctuations in the membrane potential, c is the after spike reset of membrane potential and d is the reset of the recovery variable to the rest value [12][23].

The equations model threshold behaviour, the small incline at the base of the spike, and the actual behaviour of spikes after the threshold value is reached. This is shown by the peaks of the spikes reaching values above the threshold in Figure 5.

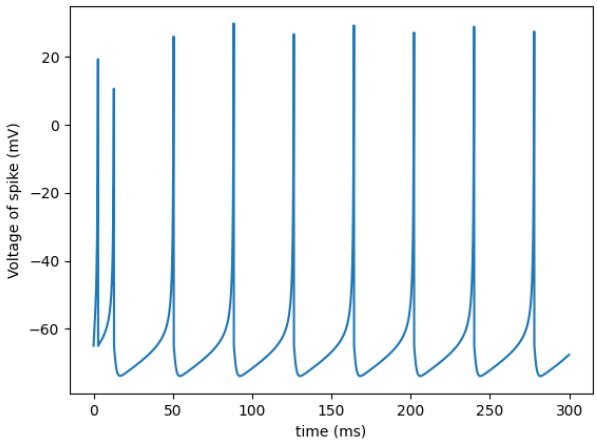


Figure 5 Shape of a Izhikevich model Action potential. Shape better matches Figure 3 and is therefore a more realistic representation of an action potential.

2.4. Hodgkin Huxley model

The Hodgkin Huxley model is a highly realistic model of a neuron derived from their pioneering work with the giant squid axon. They formed a prototype of an action potential using four standard equations based on the potentials measured from this axon [12]. It models its action potentials using the mechanisms of ion channels, represented mathematically in equations (4)-(7). It is the best for modelling systems as realistically as possible, but as seen below, the worst for computational complexity.

Hodgkin and Huxley included three dynamic variables n, m and h which describe the dynamics of the ion channels and their voltage dependence. The variable n is the activation of the Potassium channels and then m and h relate to the activation and inactivation of Sodium channels, respectively. The voltage and time dependency of the models are shown in the equations below with n_0, m_0 and h_0 [12].

The model is simplified into four standard differential equations shown below [12]:

$$C\frac{dV}{dt} = -g_{Na}m^3h(V - E_{Na}) - g_K n^4(V - E_K) - g_l(V - E_l) + I(t) \quad (4)$$

$$\tau_n(V)\frac{dn}{dt} = -[n - n_0(V)] \quad (5)$$

$$\tau_m(V)\frac{dm}{dt} = -[m - m_0(V)] \quad (6)$$

$$\tau_h(V)\frac{dh}{dt} = -[h - h_0(V)] \quad (7)$$

Where V is voltage, E_K, E_l and E_{Na} are reversal potentials for their respective channels. Then g_{Na}, g_K and g_l are the conductances for the sodium, potassium and third unspecified channel respectively [24], Figure 7. Finally, I(t) is the current input to the neuron.

The variables n_0, m_0 and h_0 are dynamic and depend on the solutions to two equations alpha and beta, discussed further by Xiaoyan Fang et al [24]. The alpha equation represents the rate of shut ion channels transitioning to open state, and beta the rate of open ion channels transitioning to the closed state. These functions [24][25] give the dynamics of the system.

The output of these alpha & beta functions, found experimentally by Hodgkin & Huxley, when plotted against voltage lead to the curves [12][24][25] seen in Figure 8A. These curves can be explained from theory from 2.1.2 as they relate to specific ion channels and their behaviour. For example, m_0 describes the dynamics of the Sodium channel.

The realisation of all the alpha and beta functions alongside the other four standard differential equations shows the computational complexity is higher than other models. Which is why it isn't used in engineering applications where there are energy and processing constraints. However, it is useful in computational neuroscience due to its accuracy.

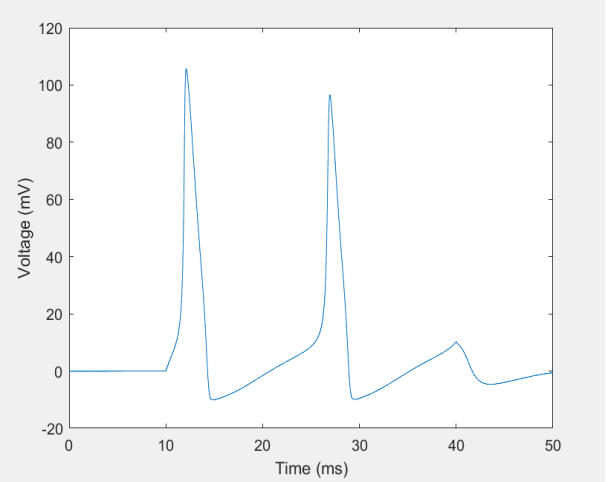


Figure 6 Shape of a Hodgkin Huxley neurons action potential [12]. This model matches Figure 3 the best.

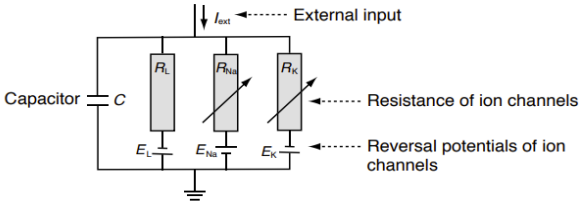


Figure 7 A visualisation of the HH neuron in circuit form [12].

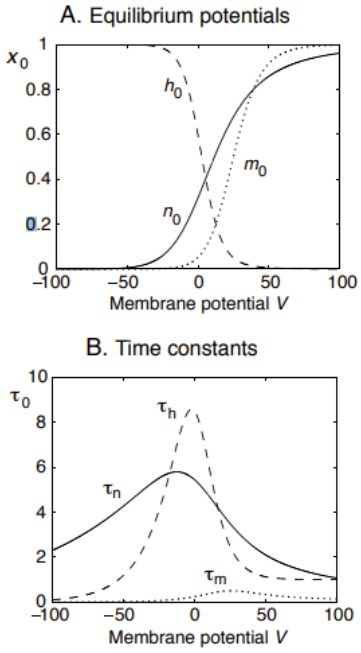


Figure 8A shows h0,m0 and n0 against the potential of the membrane. Figure 8B shows membrane time constants against membrane potential [12].

2.5. Intrinsic noise in the brain

Neurons recorded in situ have irregular and noisy activity. The issue is whether this ‘noise’ is encoding of information, a part of the neural code, or whether it is just random interference.

The noise seen by individual neurons is caused by the biochemical processes described in 2.1.2 like ion channels opening and closing and vesicles fusing with the cell membrane. It causes variations in potential which affects how the neuron operates. In large populations these fluctuations can be averaged out but on a local scale the effect of noise is larger [26]. The dominant noise is from the ion channels and experiments have shown this to influence the timing of action potentials [27].

2.6. Hebbian Learning

The idea that the brain can change and build associations is one of the main reasons why neural networks are used in engineering problems. The brain's ability to adapt and change behaviour to stimuli is incredibly useful.

In 1949 Hebb outlined that “When an axon of cell *A* is near enough to excite cell *B* or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that *A*'s efficiency, as one of the cells firing *B*, is increased.” [28]. This description of Hebbian learning explains how the alteration of synaptic efficacy changes the ability of the first cell to make the second cell react, this translates to how changing weights in a neural network makes them more or less likely to generate a spike.

This weight is altered via learning and dictates what the input's effect is on the output of the neuron. This effect can be inhibitory or excitatory as mentioned in 2.1.1. This change in synaptic plasticity, also known as synaptic efficacy [29], results in adaptations by changing connections in a system. Which changes how it responds to events. It can also result in new abilities such as pattern recognition [29].

This change in plasticity by exposure to events depends on pre and postsynaptic activity [30] and is called Hebbian plasticity/learning.

An implementation of Hebb’s rules is Spike timing dependent plasticity (STDP). STDP is an unsupervised learning rule that changes the weights of synaptic connections based on the timing between the pre and postsynaptic spikes [29][30]. The efficacy of a synapse is increased if a presynaptic spike comes before a postsynaptic spike and if a presynaptic spike comes after a

postsynaptic spike, then the efficacy of the synapse is decreased [30][31].

This rule is realistic unlike ANN algorithms. For example, backward propagation relies on loads of data and a gradient which doesn't translate to how the process of learning works in the brain [32]. The brain relies on unsupervised learning where the outcome isn't known. STDP is plausible as it learns by having no answers to compare the input to.

2.7. Vision

This section describes how visual stimuli is processed in the brain and the architecture of the primary visual cortex in the occipital lobe [33].

The brain dissects visual stimuli into distinct parts e.g., background and foreground and then uses experience to interpolate what it is [33]. Segmentation relies on cognitive processes that analyse at three levels low: focusing on contrast orientation and colour, intermediate: foreground, background and contours and then high, focusing on object recognition.

The visual cortex is divided into five subsections [34] denoted as V1,V2 etc up to V5, seen in Figure 9. In this study we focus on the V1 layer also called the primary visual cortex. It focuses on low level processing like V2 & V3 [34].

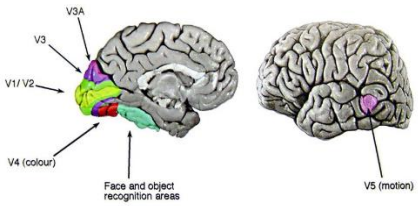


Figure 9 The occipital lobe and where the primary visual cortex is located. [35]

The V1 layer gets inputs from photoreceptors in the eye. These receptors transmit light to ganglion cells inside the optic nerve that passes to the lateral geniculate in the Thalamus, which then connects to V1 [33].

These ganglion cells form the receptive field of the retina. In 1953 Kuffler [36][37] found this field was made of cells with inhibitory and excitatory effects [33][37]. They have a centre-surround organisation categorised as on or off-centre [33][37]. This organisation encodes contrast in the visual field. The centre, as seen in Figure 10, has an opposite sign to the perimeter and are mutually inhibitory.

It is this light to dark boundary that elicits a response. On-centre, opposite of off-centre, has light in the middle and darkness on the perimeter. This allows selectivity for orientation. Hubel and Wiesel 1959 [38] discovered this emergent property when they found cells responded to the edges of slides rather than the images on them.

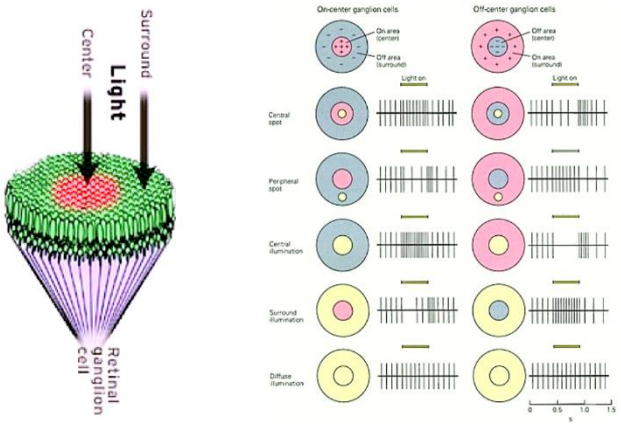


Figure 10 Receptive field on/off centre structure & responses. [39]

These cells are organised into columns based on function within the cortex. In Figures 11 & 12 circuit representations of these columns are shown, which feature inhibitory and excitatory neurons seen as smooth stellate cells, spiny and star pyramidal neurons, respectively. The inhibitory cells stabilise cortical processing [12].

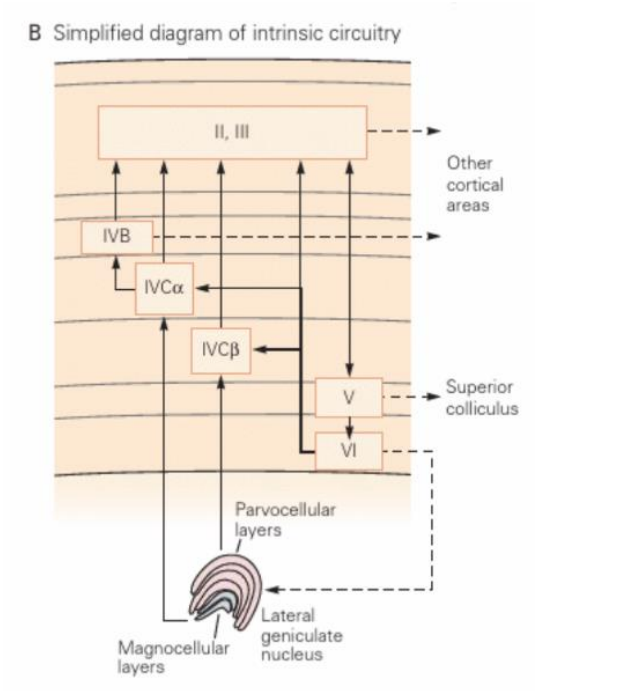


Figure 11 Diagram of simplified cortical microcircuit in the primary visual cortex. [33]

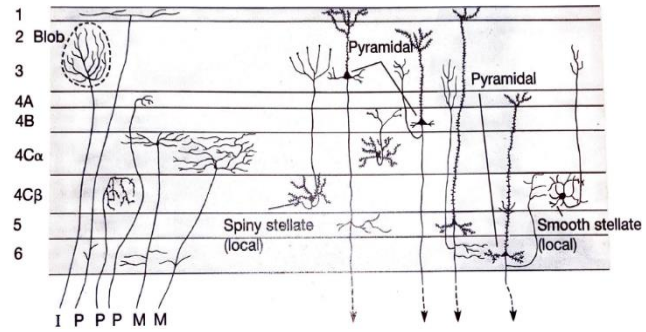


Figure 12 Diagram of neurons within the cortical microcircuit in the primary visual cortex. [12]

3. Model Architecture

The architecture of the model is formed from one cortical microcircuit seen in Figure 13. This is a circuit modelling visual columns of the V1 layer, seen in theory [33]. The outside column connections are ignored.

The connections included have been colour coded in Figure 13. The nodes are indexed according to how early they appear from the input, illustrated in Figures 13 ,14 & 16. They are numbered right to left, with the lowest index in each layer on the right. The key in Figure 18 describes how nodes are addressed in diagrams.

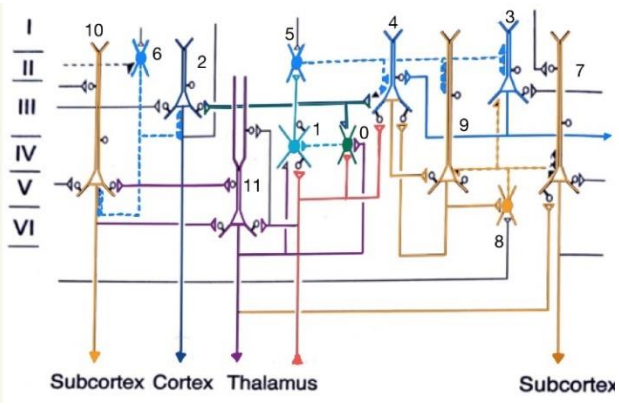


Figure 13 Modified advanced diagram of a visual column in the primary cortex. [37]

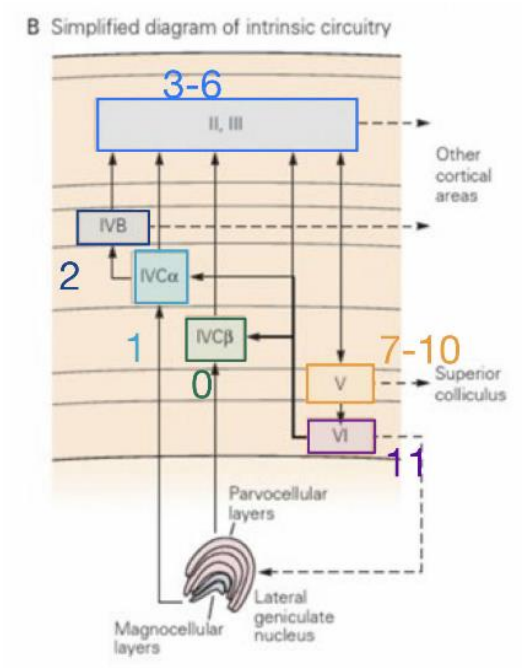


Figure 14 Colour coded Figure 11 [33].

The model will have a singular node for the input. This node abstracts the lateral geniculate nucleus which is the input to the primary cortex. It extends to IVCβ and IVCα through two different paths, seen in Figure 15 [40]. Since it passes the same input through each path it was simplified to one to optimize the design. This input is also instantiated like this to allow a feedback loop to be made between neuron VI and the input seen in Figures 14/16.

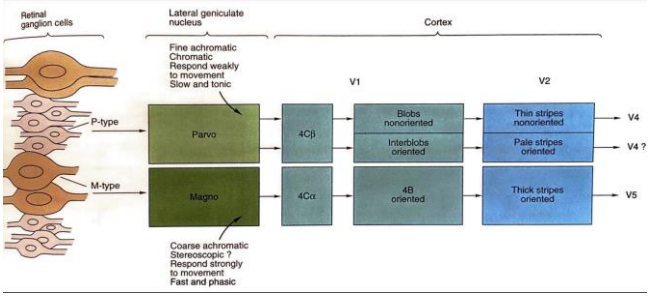


Figure 15 Diagram of lateral geniculate nucleus pathways [40].

In Figure 13, all long neurons are excitatory and in software, using the values from [5][23], have parameters of the Izhikevich model set to imitate excitatory neurons. All nodes that are represented as a ball with sticks e.g., PVC[6] are inhibitory.

Inhibitory nodes will be configured to be Thalamocortical inhibitory neurons using E. Izhikevich [23] values. This matches theory in Figure 12 where smooth stellate neurons are inhibitory and long pyramidal neurons are excitatory.

In [23] E.Izhikevich explains that the modelling of cells using these values accurately models thalamocortical neurons in the cortex which aligns with the project aims. The values are seen below.

```
#Setting Excitatory Regular Firing Patterns for Neurons  [5][23]
#-------------------------------------------------------------
input_neuron.a = 0.02
input_neuron.b = 0.2
input_neuron.c = -65
input_neuron.d = 8
input_neuron.u =0.2*-65
input_neuron.v = v_r

#Setting Inhibitatory Thalamo-Cortical Firing Patterns for Neurons [23]
#-------------------------------------------------------------------
PVC[1].a = 0.02
PVC[1].b = 0.25
PVC[1].c = -65
PVC[1].d = 0.05
PVC[1].u =0.25*-65
PVC[1].v = v_r
#-----------------
```

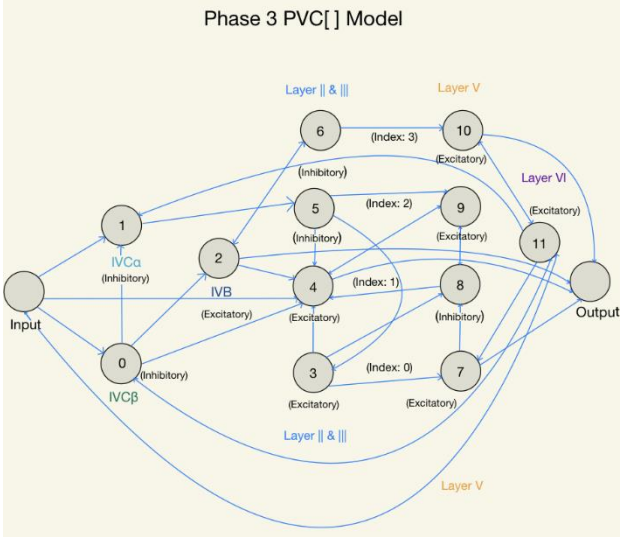


Figure 16 Original envisioned design of the model (before testing).

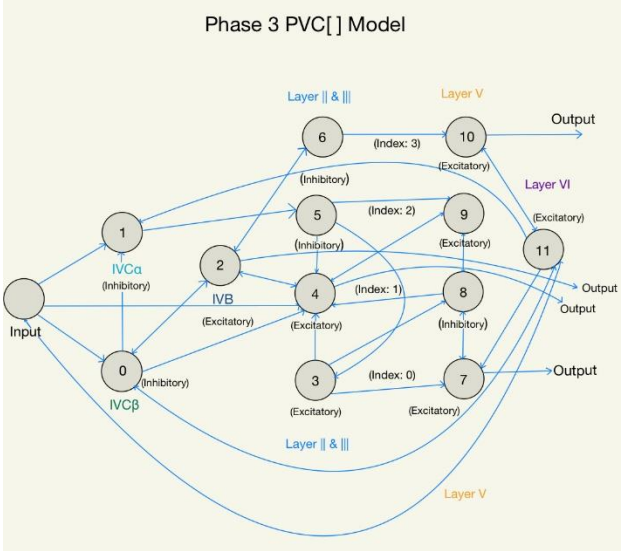


Figure 17 Final Phase 3 design (including improvements after testing).

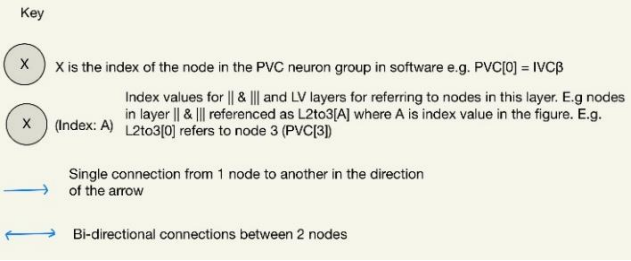


Figure 18 Key for node diagrams.

Here we discuss major aspects of the architecture. The || & ||| layer forms feedback loops between Layer V and itself. Then Layer VI implements feedback to IVCα, IVCβ and the input with further connections to layer V as shown by Figures 13 and 14. The loops are included to see how internal feedback effects the model. The final outputs are set by the four arrows outwards in Figure 13.

4. Implementation

The model is coded in Python as the language is simpler than alternatives like MATLAB and Java. There was prior experience of python which reduced setbacks early in the project and there are packages that aid in the creation of SNN's. Whereas languages like Java would require the creation of functions to construct nodes which isn't a core objective for this project. The package the software uses is Brian2 [41]. It was chosen as there is heavy documentation of functions and tutorials on how to use key concepts. Due to the simplicity of the language and good documentation, it was easier to learn than competitors.

Alternative packages were SnnTorch, Nest or Tensor flow. Tensorflow didn't have functions for instantiating SNN's, and SnnTorch & Nest only supported backpropagation as a learning algorithm. Which didn't align with the need for biological plausibility. Brian2 supports biologically plausible concepts like STDP that

wasn't possible with other packages and has functions built in to instantiate nodes which aids in the architecture's creation.

The Izhikevich model is used to closely mimic real neurons without the complexity or lack of detail in the Hodgkin Huxley or LIF model, respectively as discussed by E. Izhikevich [23]. The complexity from Izhikevich is acceptable as it doesn't reduce the model's performance due to its scale. This allows it to model the primary cortex realistically without reducing its applicability elsewhere. The nodes are simulated with the Ornstein-Ulhnbeck method [42] that represents thermal noise from movement of chemicals (2.1.2) to replicate noise in neurons.

The STDP algorithm is used as it is unsupervised and mimics processes that the brain uses to learn. It enhances the biological plausibility of the model. Here all weights will be instantiated to one to stop unexpected behaviour affecting the algorithm. Random weights were trialled, but affects are seen in Appendix Q.

The Microcircuit in Figure 13 is used to replicate the structure in the brain to gain benefits from its bio-inspired design. The design is split into three phases, each adding four nodes until reaching completion, Figures 16/17. This project contrasts to sizeable implementations like Guozhang, Franz and Wolfgang's [43] model as it includes biological detail for modelling without restricting its usability in engineering applications.

Finally, the software code is split between three files. The "Compact Phase 2" is for the testing of sinusoidal, Poisson and individual events, "Vision Phase 2" is for inputting visual stimuli and "Phase 2 Demo" demonstrates a model application. This allows for a clearer structure in the files.

5. Experiment

To expose the behaviour of the model, both continuous and non-continuous signals are used. Poisson spike trains [44] simulate the input of noise, revealing its performance over a variety of scenarios. The use of sinewaves simulates the response of the microcircuit to "light." Then non-continuous signals are used to see behaviour under isolated events. Finally, event-based images are input to see if the model achieves low level processing like the brain. Configurations of experiments below.

- Poisson Inputs from 10 to 100 Hz in 10 Hz intervals. Testing with weights 1 to 5 from 25Hz to 100Hz
- Sinusoidal inputs from 10 to 100 Hz in 10 Hz intervals. At each frequency trialling amplitudes between 5-20 in 5-unit increases.
- Use Timed arrays to implement non continuous inputs.
- Trial event-based number representations 0-9

6. Testing

The testing of Phases 1 & 2 is described here, mentioning in order of completion.

6.1 Phase 1 Architecture: LIF

The first phase of the architecture, Figure 19, featured five nodes to meet the minimum goals of the project. It uses the STDP algorithm with constants, from Song & Abbotts [45], on the connections between nodes. This model used LIF [12] as there was more experience with it at the start of the project.

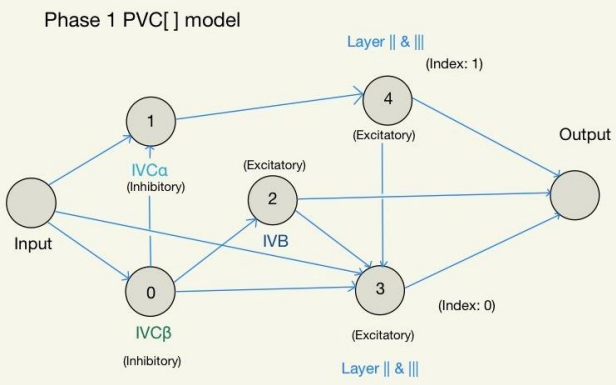


Figure 19 Original Phase 1 design of the primary cortex model.

The full development is in Appendix O. Initial testing with low weighted Poisson inputs showed unstable behaviour, Figure 20.

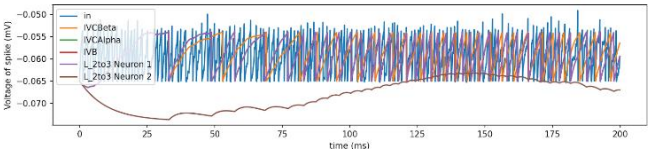


Figure 20 Unstable behaviour with low weighted Poisson input (1mV) to phase 1 LIF model.

When increasing this weight to one this behaviour was removed. However, across multiple frequencies an error with STDP was observed, Figure 21. At 60ms nodes stop firing due to weights dropping to zero. This unknown issue meant the implementation was abandoned to continue project development.

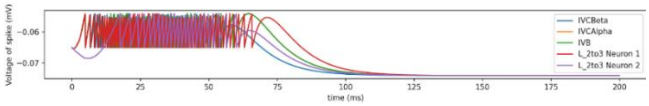


Figure 21 Failed simulation of Phase 1 LIF model. STDP drops weights to zero and nodes fail at approximately 60ms.

6.2 Phase 1 Architecture: Izhikevich

The architecture was reconstructed using the Izhikevich model. The changing of models meant the STDP equation had to be changed. It was difficult to get any output without the code below. However, running the model exhibited instability as the nodes hyperpolarized to -90mV, Figure 22.

```
pre_eqn ='''
v_post += 100*w
Apre += dApre
w = clip(w + Apost, 0, gmax)'''
```

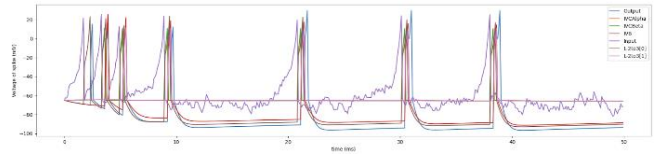


Figure 22 Model reacting to trialled version of STDP. Scaled version of learning mentioned in 2.6. It was scaled as an issue in the design caused functionality issues. Nodes synchronously hyperpolarise to -90mV which isn't biologically plausible.

The previous incorrect code was changed in a separate way that aligns with multiplicative Hebbian learning [12][45]. This meant altering the voltage of the spikes by multiplying with the

weight with the code below. This produced Figure 23.

```
pre_eqn ='''
v_post *= w
Apre += dApre
w = clip(w + Apost, 0, gmax)'''
```

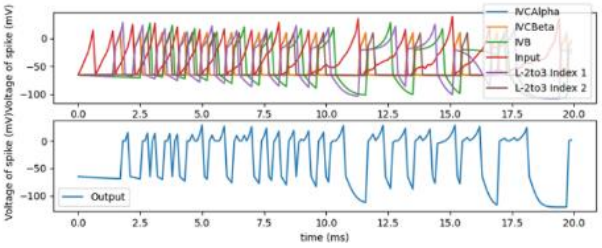


Figure 23 Incorrect implementation of STDP using trialled multiplicative Hebbian learning (Oja's rule [12]). Nodes exhibit uncharacteristic action potentials for the Izhikevich model.

The previous graphs were incorrect implementations of STDP. However, with reference to research by E. Izhikevich 2006 [46] the code was edited to match theory, seen below. The weight modulates current, not voltage directly [46].

```
pre_eqn ='''
I += w
Apre += dApre
w = clip(w + Apost, 0, gmax)'''
```

Figure 22 highlighted an error with L-2to3[0]. Therefore, the links from Figure 19 were referenced against Figure 13 to locate the issue. The original design misses bidirectional links, corrected in Figure 27. Another iteration was then completed, checking every link after instantiation like in Figure 24, Appendix P. The new architecture is shown in Figure 25.

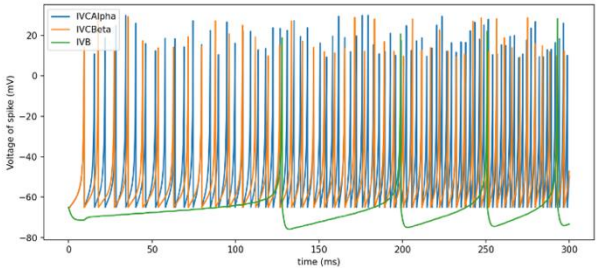


Figure 24 Phase 1 with IVCα/β and IVB.

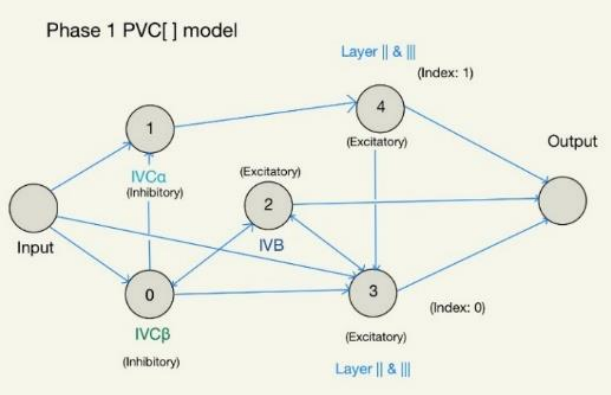


Figure 25 Corrected Phase 1 design. Bidirectional links introduced between PVC[0] and PVC[2]. Then between PVC[2] to PVC[3]. Internal architecture matches Figure 13.

This ensured phase 1 was completed correctly. However, it found having one output node caused unstable behaviour, seen in Figure 26.

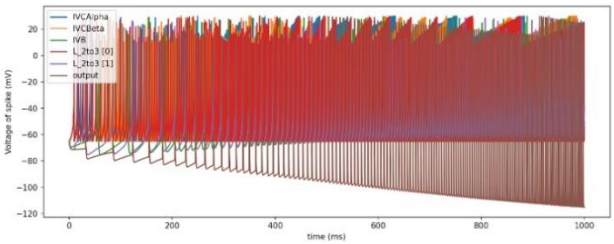


Figure 26 Unstable output behaviour with original design. Output nodes hyperpolarisation keeps increasing across simulation.

The architecture for this and subsequent models was changed to output from specific nodes dictated by Figure 13. Shown in Figure 27.

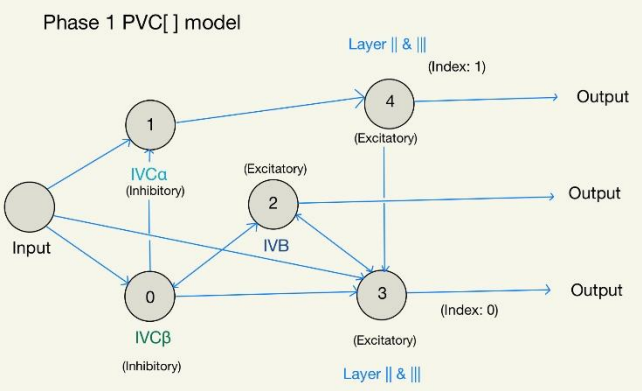


Figure 27 Final phase 1 model. All links correct and outputs with reference to Figure 13.

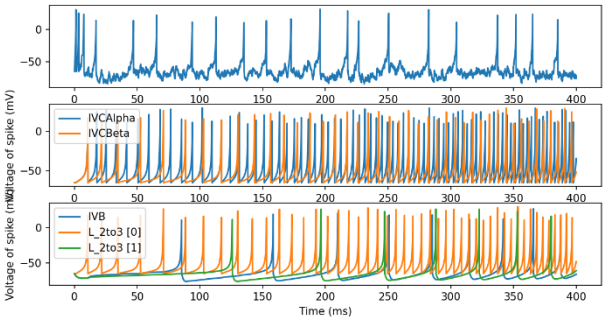


Figure 28 Testing of new architecture showing it to function correctly under Poisson inputs.

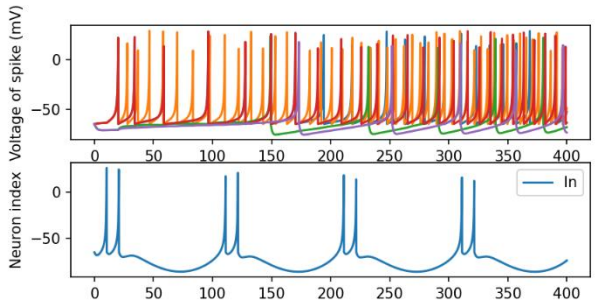


Figure 29 Test of final Phase 1 Izhikevich architecture with 15 Amplitude Sine wave at 10Hz showing it to function. Major testing of this shown in Appendix F.

6.3 STDP verification

Here the learning algorithm is shown to function correctly. The STDP variables, Apre and Apost, have positive and negative dependencies on the update of the weight respectively [47], Figure 30. The different polarities mean if one is increased the other is decreased [47]. As they are exponential terms if there are no inputs, they will decay to zero [30].

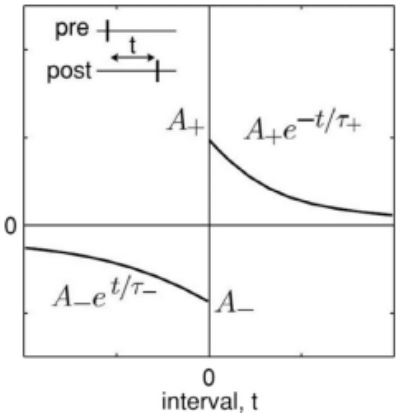


Figure 30 plotted STDP variables, displaying their polarities [30] and exponential function shapes.

In Figure 31 at 160ms an input spike comes before an output spike at 175ms, this causes Apre to increase and Apost to decrease [30]. The opposite of this can be seen at 250ms where an output spike comes before an input, decreasing the value of Apre and increasing Apost as expected from theory. The described behaviour shows the algorithm works.

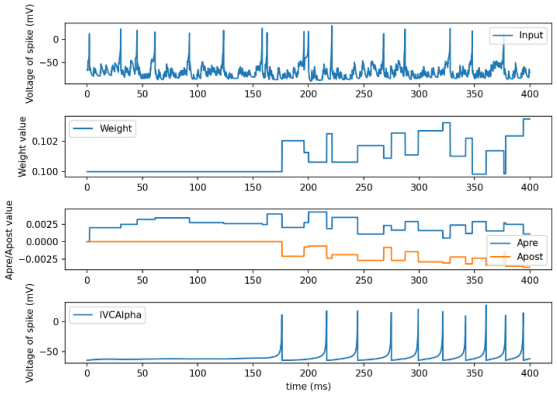


Figure 31 STDP variable outputs on a connection between the input & IVCα to Poisson input. The behaviour matches theory and shows the algorithm to function correctly. The weights are instantiated at 0.1 to make values clearer in this plot.

6.4 Phase 2 Architecture: Izhikevich

The second phase (Figure 32) builds on phase 1 by including the full || &||| layer and two nodes from the LV layer. This surpasses the minimum goal of the project, as the network stands at nine nodes and includes the feedback loop between || & ||| and LV. The third phase was reached but the experiment was run on phase 2 due to issues in its construction.

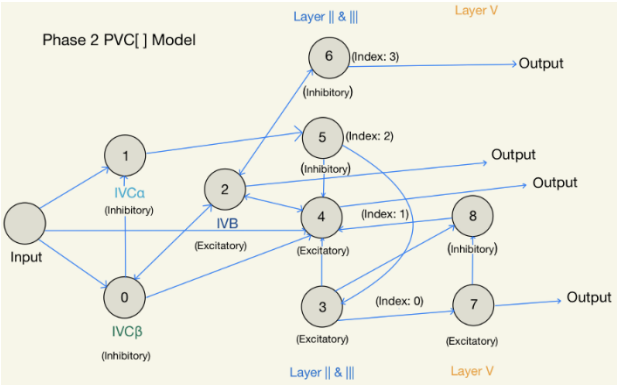


Figure 32 Original Phase 2 design.

6.5 Poisson Inputs

The boundary of operation was found for Poisson inputs. At frequencies below 25 Hz and weighted inputs below 0.3 the phase 2 model did not respond, Figure 33. The testing range now starts at 30 Hz as there wasn't enough data to be used in the results for lower frequencies.

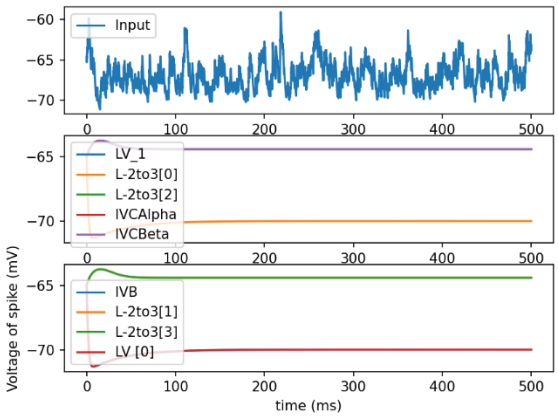


Figure 33 Model unresponsive to small inputs. All nodes in the architecture stay at their resting potentials. Differing action potentials

6.6 Sinusoidal Inputs

In Figure 34 an issue with LV[0] was found. There were missing links between PVC[7]-PVC[8] in the original design and were corrected in Figure 35

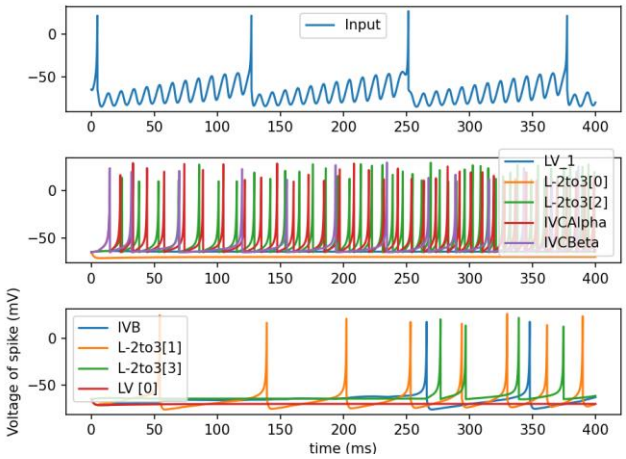


Figure 34 Issue found with connections within the model. Incorrect connection on LV[0] left node unresponsive.

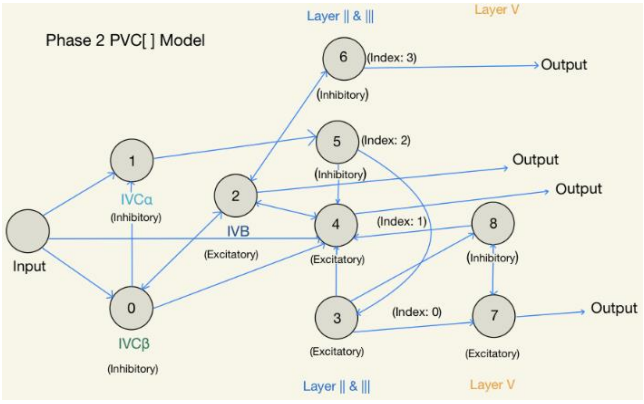


Figure 35 Final Phase 2 design. Connections updated on LV[0] with reference to Figure 13.

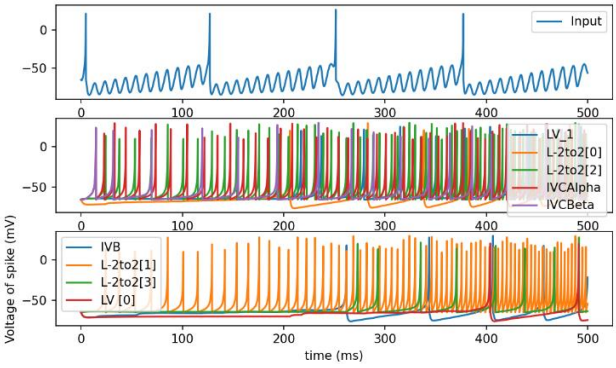


Figure 36 Response from corrected model, showing all nodes firing as expected.

The frequencies of operation are between 25 to 100Hz. This is because the model doesn't respond below 25 Hz and above 100Hz is unphysical for neurons. The minimum amplitude for operation changes across the frequency range shown in Figure 37.

	Amplitude			
Frequency (Hz)	5 to 20	6 to 20	7 to 20	8 to 20
10				
20				
30				
40				
50				
60				
70				
80				
90				
100				

Figure 37 Experimentally found range of operation for model under Sinewave input.

6.7 Individual events

The "dt" part of the syntax allowed for the spacing of individual events every 100ms. The first begins at zero, showing current values of 6 & 7 were enough to cause a response.

```
times = [2,1,6,0,7]
I_recorded = TimedArray(times,dt = 100*ms)
```

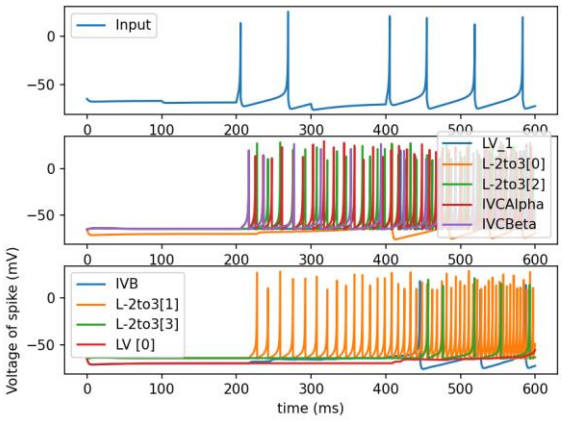


Figure 38 Output of Phase 2 when presented with individual events of different current values.

As shown in Figures 38 & 39 the addition of other events at separate times causes the response of the model to strengthen. The tests are successful in showing there is no response unless there is an event.

```
times = [6,3,0,0,6,0,0,0,0]
I_recorded = TimedArray(times,dt = 100*ms)
```

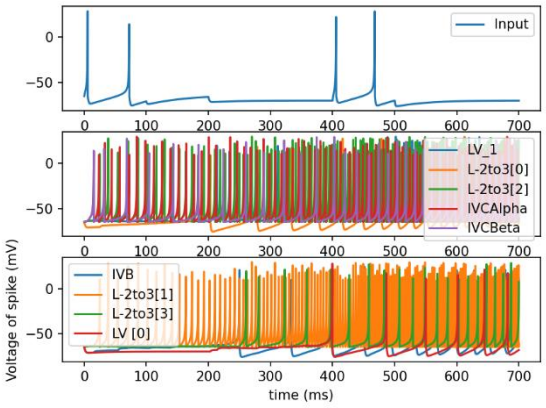


Figure 39 Output from different configuration of non-continuous input. Increased response after second event shows adaptation of the firing rate to inputs.

6.8 Vision

The visual stimuli will be representations of event-based images made from the function from Brian2 [48]. Below is an output of the code [48], without alteration, when exposed to an image of the number 7 [49].

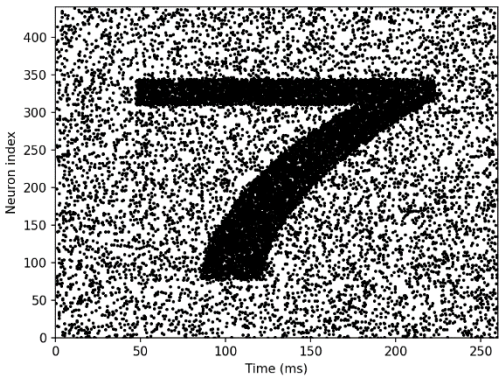


Figure 40 Event based image of a 7 [49] represented by individual neurons [48].

The resolution of the event image was changed by trailing different multipliers on the array and noise term, here 1.95 and 0.02 are used respectively, Appendix R.

```
eqs = '''
dv/dt = (1.95*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1
'''
```

The code below attaches the nodes exposed to the image to an Izhikevich neuron producing Figure 41. The stimulus is unable to cause the Izhikevich node to respond, Figure 41.

```
# Neurons exposed to image
#-------------------------------------------------
im_neuron = NeuronGroup(N, eqs, threshold='v >1', reset='v = 0', method='euler')
# Usual input neuron group instantiation

input_neuron = NeuronGroup(1,izhikevich_eqns,threshold = 'v >=30',reset ='v = c;
# Connecting the image neurons to the main model
image_input =  Synapses(im_neuron,input_neuron,model='w:1',method = 'euler')
image_input.connect()
image_input.w =1
#-------------------------------------------------
```

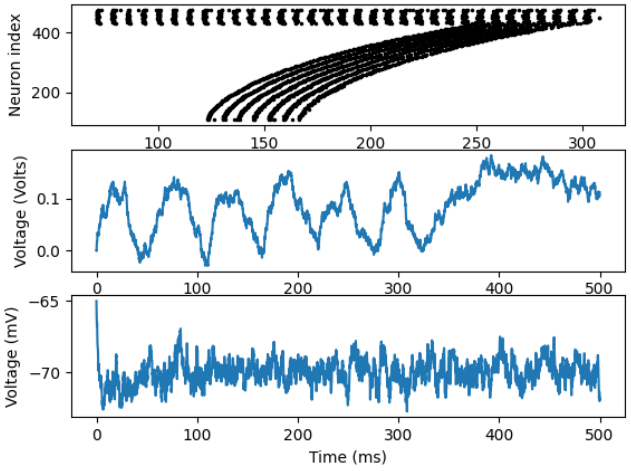


Figure 41 Unresponsive node to event image. Top pane is the number 7 represented by individual neurons. Middle pane is response of nodes exposed to image. Bottom pane is the response of an Izhikevich node connected to the image exposed nodes.

This issue of not meeting threshold was fixed by adding one to the output voltage of the synapse. This ensures the second neuron responds by aiding the transition between two groups with different threshold conditions.

```
# Connecting the image neurons to the main model
image_input =  Synapses(im_neuron,input_neuron,model='w:1',on_pre ="v_post +=1",method
image_input.connect()
image_input.w =1
```

The output of the Izhikevich node is seen in Figure 42. The response is irregular, but this is because the function relies on noise.

When changing the stimulus, Figure 43, the nodes response changes depending on the image presented.

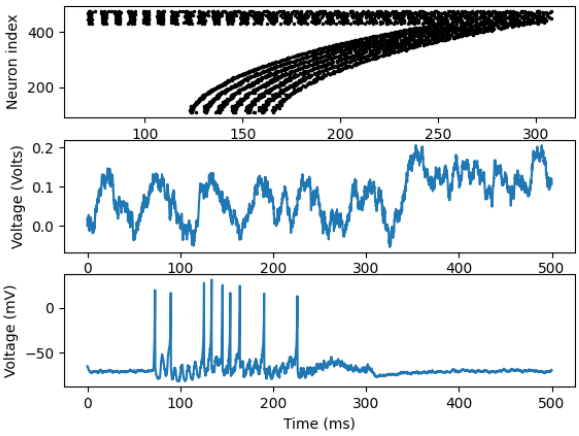


Figure 42 Response from Izhikevich node to LIF image neurons. Middle pane is response of nodes exposed to image. Bottom pane is the response of an Izhikevich model connected to the image exposed nodes.

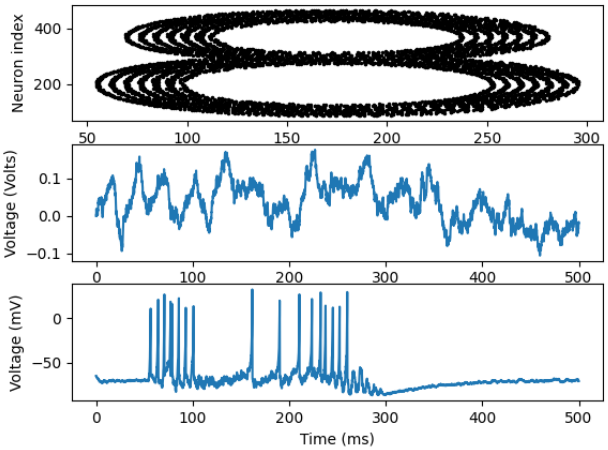


Figure 43 Different response in node to a different image (8 from [50]). Middle pane is response of nodes exposed to image. Bottom pane is the response of an Izhikevich model connected to the image exposed nodes.

The group of image exposed neurons were input into the phase 2 architecture. The connection has STDP applied as the image exposed neurons now function as the input layer. It successfully elicited activity from the model. However, it is a strong response, Figure 44.

```
# Connecting the image neurons to the main model
image_input =  Synapses(im_neuron,input_neuron,model =stdp_eqn,on_pre=pre_eqn,
          on_post='''Apost += dApost
                w = clip(w + Apre, 0, gmax)''',method = 'euler')
image_input.connect()
image_input.w =1
#-----------------------------------------------------
```

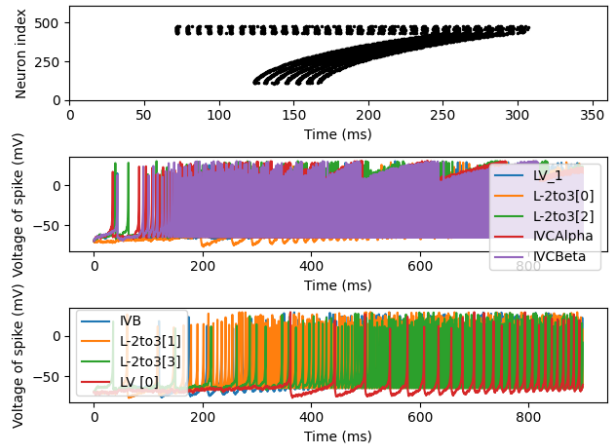


Figure 44 STDP on connection between image exposed neurons and model input causes unstable responses. Shown in the nodes firing patterns being very regular.

Removing STDP on the input improves stability of the model's response. When changing the method back to what was used in Figure 42, it is seen the "+=1" was too large. This was tuned to "+=0.2" as it resulted in stable behaviour, Figure 45.

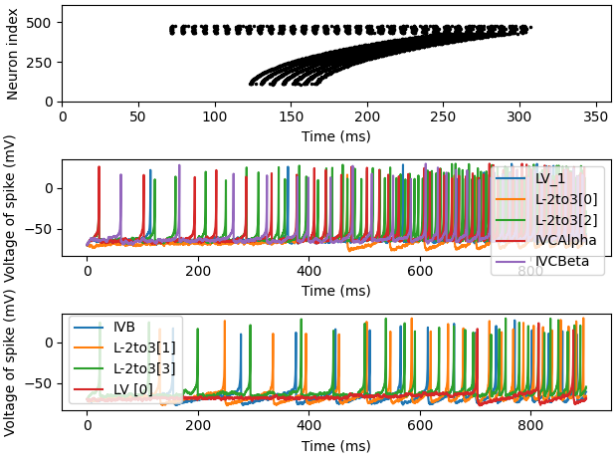


Figure 45 Reverting to previous methods showing more sparse firing rate in line with other tests.

6.9 Noise

It was found that using an amplitude of two on the noise term in the Ornstein-Ulhnbeck equation [42] granted sufficient noise to simulate real neurons without degrading the model's performance, Figure 46. Full testing of strong noise and Poisson inputs is seen in Appendix H, which shows the model's noise resilience.

```
izhikevich_eqns = '''
dv/dt = (0.04*v*v + 5*v + 140 -u + I )/ms + 2*sqrt(2/tau_M)*xi : 1
du/dt = a*(b*v - u)/ms : 1
a : 1
```

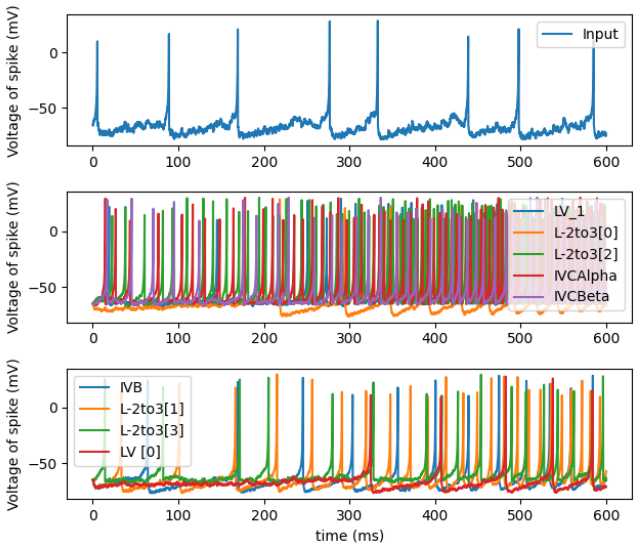


Figure 46 Nodes with noise with an amplitude of 2 in the Ornstein-Ulhnbeck equation (2*sqrt(2/tau_m)*xi).

7. Results

This section discusses observed behaviour of the model. There are spike times recorded from each experiment that are averaged over 5 epochs due to noise. Statistical analysis is done via a cross-correlation function, which is a python version of David Halliday's code [51]. The behaviour is input dependent operating within the ranges below and Figure 37. Different weighted Poisson inputs allow the model to respond in ranges where it previously didn't e.g., between 25-100Hz instead of 30-100Hz, Figure 47.

Poisson	Frequency operating range
Weight 1	30-100Hz
Weight 2 to 5	25-100Hz
Sinusoidal	25-100Hz

The spiking rate of individual nodes increases exponentially for Poisson and Sinusoidal, Figures 48 & 49, Appendices B/C/D. Sinusoidal responses are similar across frequency ranges showing stable activity, Figure 50, Appendix D.

The model struggles at higher frequency sine waves with middle band frequencies featuring more activity.

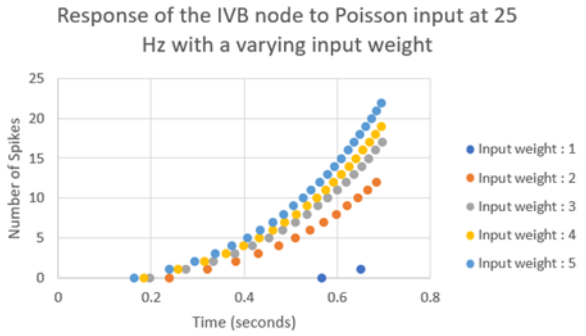


Figure 47 Node IVB's firing response is stronger under stronger inputs, allowing it to function at lower frequencies.

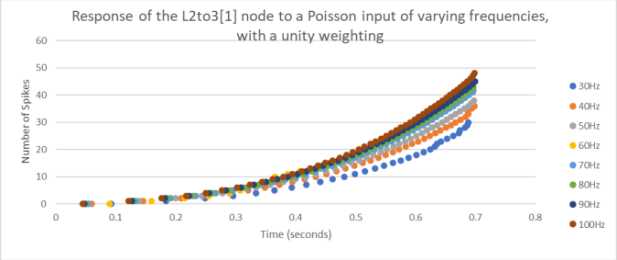


Figure 48 L2to3[1] response under Poisson input. Shows behaviour is similar under higher frequencies. Higher activity is shown at larger frequencies as shown by the spike count.

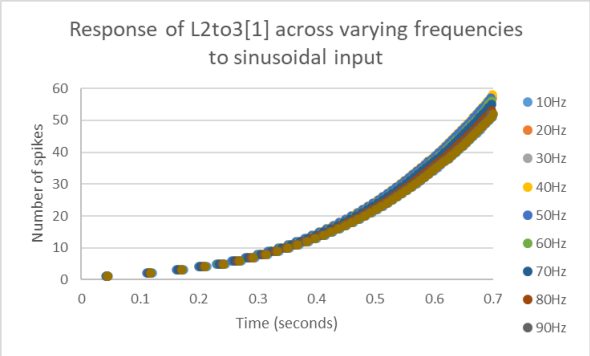


Figure 49 L2to3[1] the most active node in the network has a similar firing pattern across the testing range. Lower frequencies of 40-60Hz elicited more spikes suggesting model works best in this region.

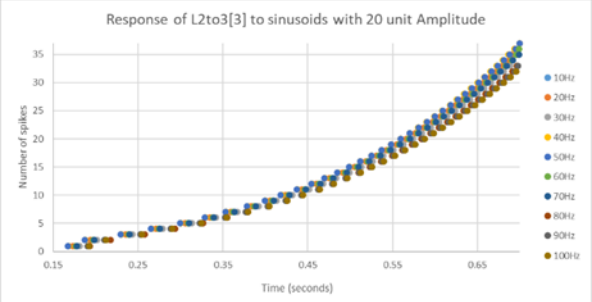


Figure 50 L2to3[3] node has similar spike count across the frequency range. 40/50Hz region seems to be nodes most active frequency. Nodes struggle at 100Hz which for real neurons is also unphysical.

In the results from Poisson inputs the nodes responses differ from each other except under high frequencies 70-100Hz. For event based its very nonlinear for each node, Figure 51/52, Appendix E/N. They scale their response randomly after 800milli seconds. Despite this, each plot is different for each number, suggesting it can differentiate between images. Showing low- and high-level processing of the brain as discussed in section 2.7.

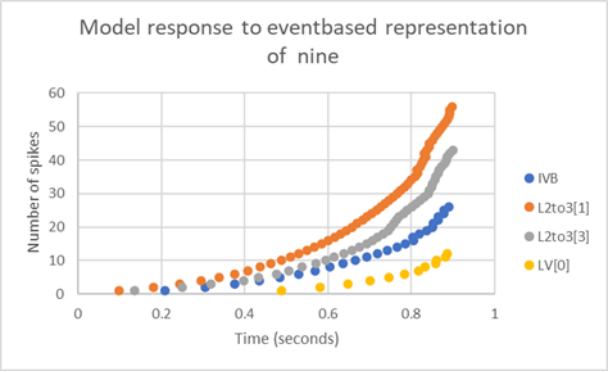


Figure 51 Averaged responses of nodes under the event-based representation of the number 9.

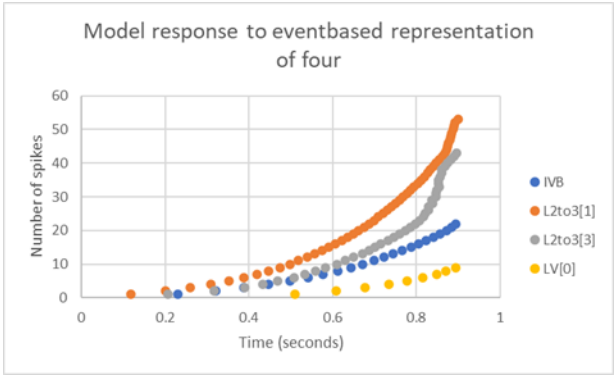


Figure 52 Averaged responses of nodes under the event-based representation of the number 4. Nodes react differently to other images suggesting ability to distinguish between visual stimuli.

Other biological behaviours observed are cortical stabilising , excitatory nodes hyperpolarisation decreasing over time from stabilising from inhibitory nodes and adaption of firing rates in output plots, Appendix B/D and Figure 39.

Statistical analysis of the Poisson and sinusoidal tests found the model has a sparse firing rate. Although the count is high for correlated spike pairs, Figure 53, this is due to chance as highlighted by Figure 54. This because with a high spiking rate the chance of spike times naturally overlapping is higher as in the Rasta plot showing spike times heavily overlapping.

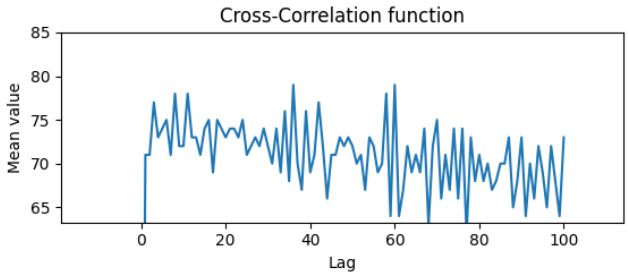


Figure 53 Cross correlation of input to L2to3[1] under Poisson input at 50Hz.

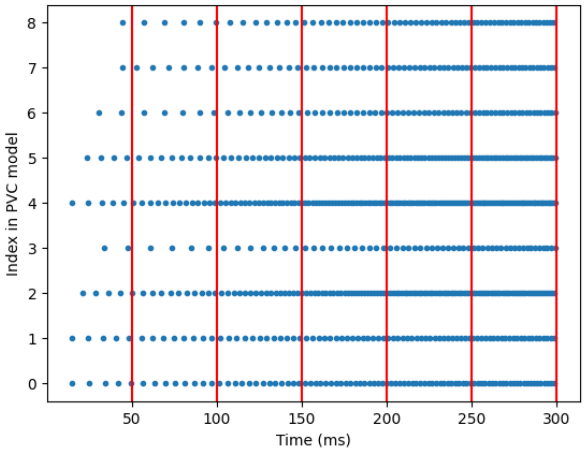


Figure 54 Rasta plot showing synchrony of nodes under 50Hz Poisson input. High number of spikes shows spike times correlating quite frequently. This is due to chance as the more spikes occurring the higher the probability of spike times correlating.

For sinusoidal plots, this point is easily realised, Figures 55 & 56. The counts are low with the highest count of five being from the L2to3[1] node. The low counts are not statistically significant and show no correlation. This is because these counts are due to chance.

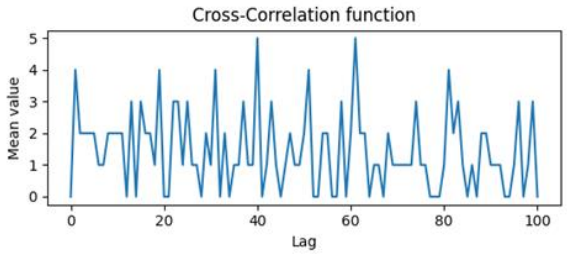


Figure 55 Low number of correlated spike times showing nodes have a sparse firing rate. All correlation is due to chance and no behaviour can be determined as results aren't statistically significant.

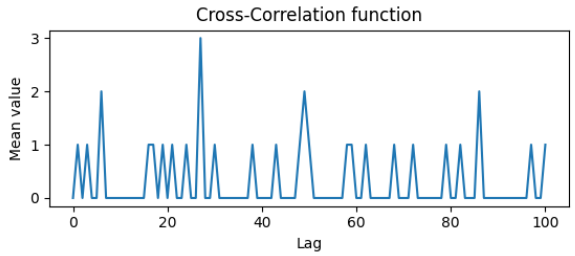


Figure 56 Low number of correlated spike times showing nodes have a sparse firing rate. All correlation is due to chance, no significant peaks.

The cross-correlation function and previous tests did highlight architectural behaviour. It's seen that L2to3[1] is most active closely followed by L2to3[3]. The LV[0] is then shown as the least active node. These results are physical as L2to3[1] is shown to be close to the input and to have the most connections. Furthermore, LV[0] is far from the input and relies on a feedback loop for input, Figures 57 & 58.

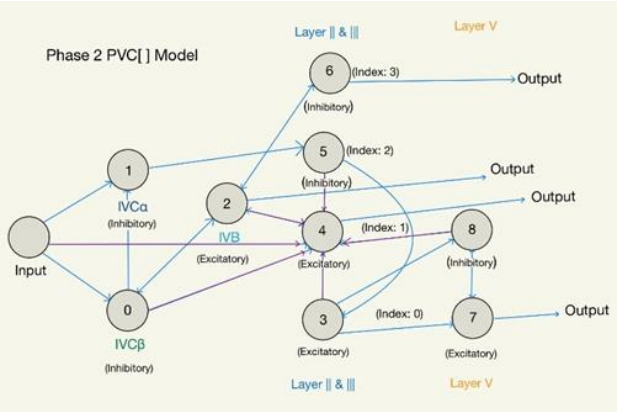


Figure 57 Purple annotation shows L2to3[1] (PVC[4]) to take the most inputs than other nodes in thenetwork. Showing higher spike counts than others.

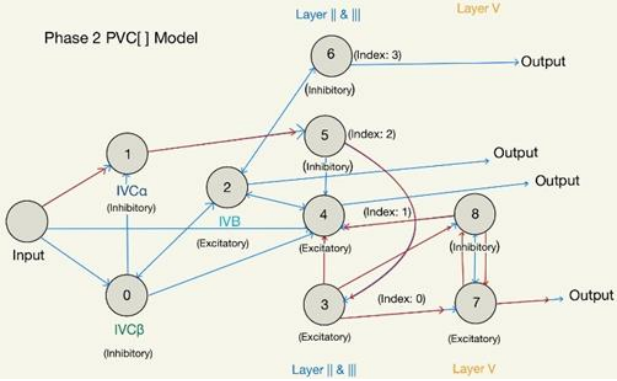


Figure 58 Red annotations show LV[0] (PVC[7]) to be distant from the input relying on a feedback loop that gets input from PVC[5]. Lack of input gives it its low response.

Despite the low firing rate revealing little intrinsic detail, it is a positive result for the study. It is documented that cortical microcircuits have slow dynamics [52] which means the study successfully modelled the primary visual cortex correctly, with a minimum of mimicking other studies.

8. Application of the Model

An example application of SNN's is in event-based vision, seen here in the context of safety control systems. Event cameras are asynchronous and their benefits [6] come from the fact they only register changes in their vision, Figure 59. A specialist in neuromorphic vision, PROPHESEE [54], outlines uses of these products in machinery monitoring, Figures 60 & 61.

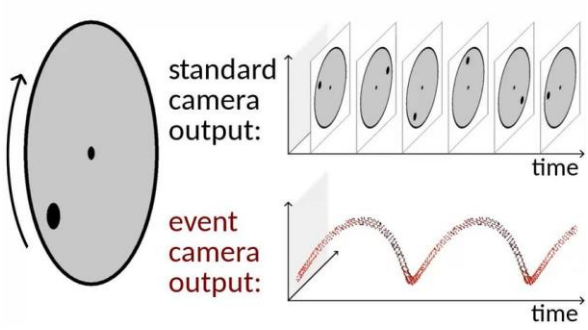


Figure 59 Shows asynchronous nature of Event based vision, only detecting changes in the field. Doesn't detect the majority of the background as there are no changes. Unlike classical which oversamples the whole image which isn't as efficient [53]. This type of vision matches how the brain achieves visual processing by detecting edges or events in its visual field.

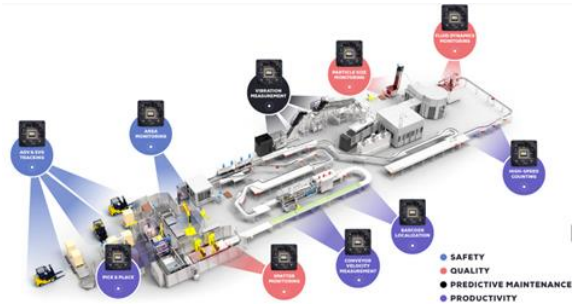


Figure 60 Assembly line in a factory showing where event vision can be used for safety and quality control [54].

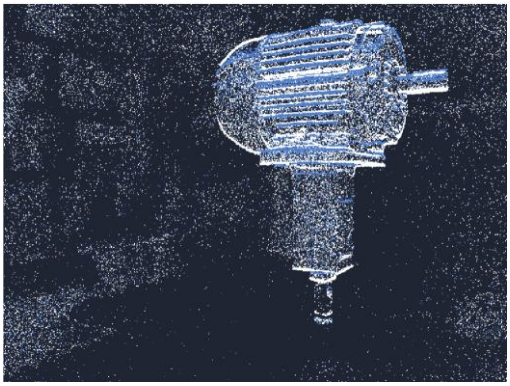


Figure 61 PROPHESEE's neuromorphic camera detecting oscillations of a motor during its operation [54]. Shows movement in the image better than classical cameras, indicating its usefulness in monitoring mechanical systems.

Their example shows clear monitoring of a motor's movement, which if it were to fail would be captured by the camera's sensitivity to abrupt changes.

An example of the model completing a similar task is seen in Figure 62. This version is cheaper and accessible, as it requires a basic webcam and doesn't cost thousands. Whilst having the same performance benefits [6] e.g., power efficiency.

Figure 62 An image of a generator [55] simulating a scenario where the model is applied to monitoring industrial machinery

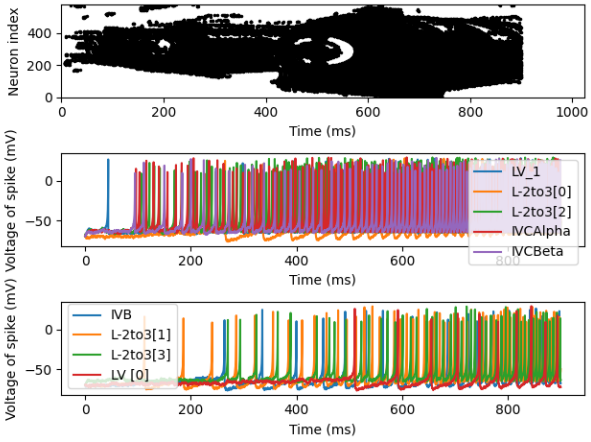


Figure 62 B Example of the Phase 2 model reacting to the generator in Figure 62 A similar to PROPHESEE's example. Showcasing the models applicability outside of biological modelling.

A demonstration of our architecture videoing environments is seen using OpenCV [56][57]. Where the phase 2 SNN architecture responds to events in its vision in real time.

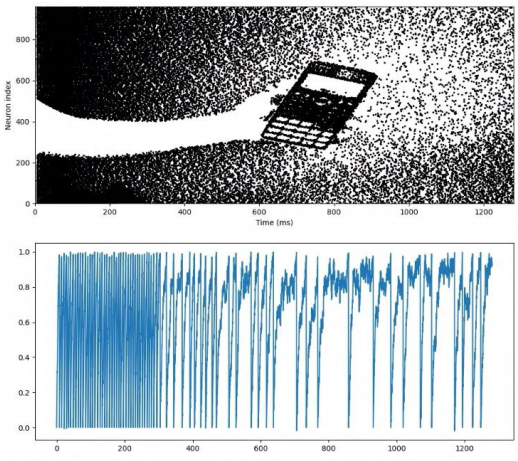


Figure 63 A Event-based representation of Figure 63 B showing how a tested node responds to the image.

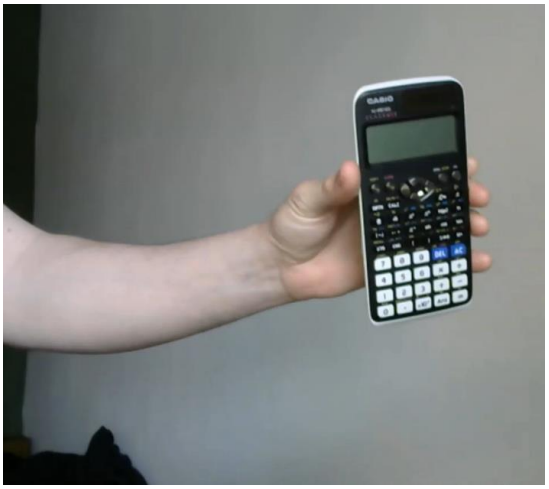


Figure 63 B Image of video capture from webcam showcasing the projects alternative cheaper implementation of event-based vision. Bottom pane is response of neuron in time to recent camera image.

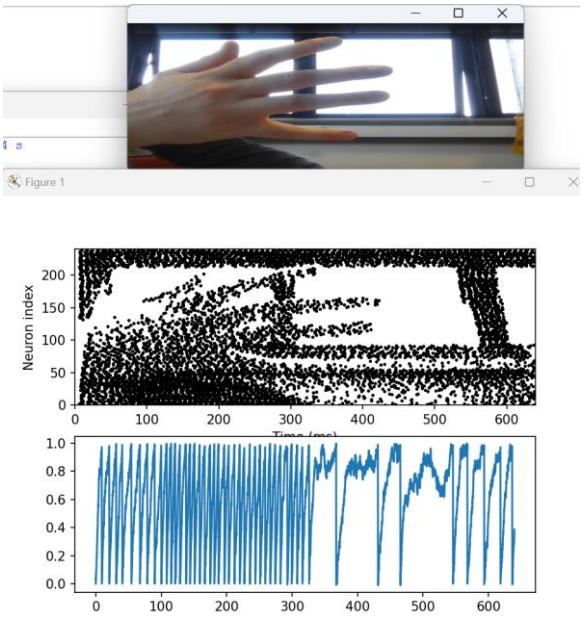


Figure 64 Live demo Test on singular neuron with live camera feed. Bottom pane is response of neuron in time to recent camera image. Video demo within files.

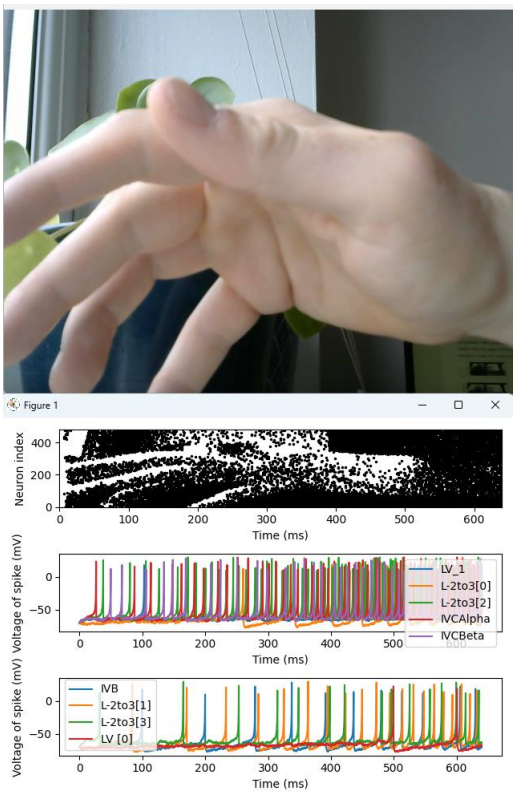


Figure 65 Live camera feed from webcam into phase 2 architecture. Phase 2 response to event-based image feed, showing edge detection of the stimulus.

9. Project Management

Initially the aims of the project were outlined to address a gap in the literature. This determined the scope and size of the project and allowed for objectives to be set. These were outlined in the beginning of the report by using the S.M.A.R.T methodology [58]. By using this it made a project plan, in the form of a Gantt chart, easy to create as the aims were specific and attainable. The Gantt chart outlined a timeline and divided the work into three main phases in parallel to tasks around report writing and research.

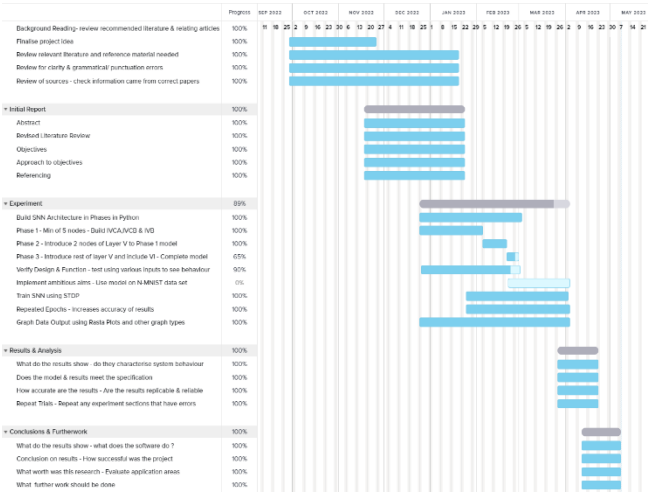


Figure 66 Project Gantt chart made with [59]. All project aims were achieved except running the model with the neuromorphic data set N-MINST and finishing phase 3. This shows the Gantt structure to be effective in keeping the project on track and accomplishing most goals in the project.

The progress was monitored through checkpoints, weekly updates to both supervisors and an increasing number of meetings with the primary supervisor as time elapsed. Personal monitoring

was achieved through the Gantt chart and by sticking to weekly aims. Through this monitoring, risks of scope creep were mitigated, and progress maintained.

In the end of the project, the development of phase 2 and the vision section took longer to complete than expected. Through control of project issues, this was solved by changing the plan by disregarding phase 3 and focussing on the verification of phase 2.

This management allowed for the project to finish according to plan only missing one section, which is regarded as successful as project objectives were still met.

10. Conclusion

The project has been successful in modelling the primary cortex. It mirrors concepts in theory such as cortical stabilising, Hebbian learning and noise resilience. Its bio-inspired design achieves biological plausibility strived for in its design.

The software's complexity means its applicable to both modelling and engineering applications which was fundamental, Appendix N. The model can perform low level processing like the cortex and has demonstrated behaviours of the microcircuit not documented in its description.

The analysis revealed low activity in the model which matches existing studies, showing it functions correctly. Its activity also shows neural coding in the form of the timings of spikes, resonating theory. Aside from results, the minimum of modelling five nodes with noise was achieved. Whilst also using excitatory & inhibitory nodes. This research provides groundwork into bio-inspired software in conjunction with neuromorphic hardware and is useful in expanding the literature in this developing field. The applicability of this work and the insightful results means the project was a success.

11. Further work

The third phase of the model wasn't completed. An implementation is given in the files with this document. It is faulty as some nodes don't fire. In future work I would reconfigure its code to ensure that it worked correctly.

I would then implement the model on the neuromorphic data set N-MNIST. This would allow performance benchmarks against other systems and define the system's ability for edge detection and character recognition. In future work I would reuse weight values and perform more runs of the simulation to ensure the model

would become selective to event-based images.

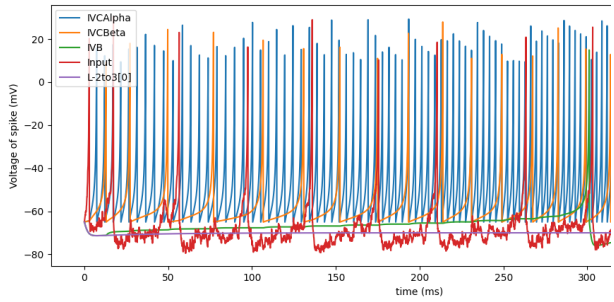


Figure 67 Faulty output of Phase 3 model. An internal node is seen to be unresponsive (L2to3[0]).

Figure 68 N-MNIST handwritten dataset of numerical digits. That have been converted into event-based representations using an event camera. It is widely used to benchmark performance of SNN models [60].

12. Statement of Ethics

There are no ethical considerations for this project as no others are involved. The only ethics consideration is the review of papers that have questionable treatment of animals.

References

[1] L. Kováč, "The 20 W sleep-walkers," *EMBO reports*, vol. 11, no. 1, pp. 2–2, Dec. 2009, doi: https://doi.org/10.1038/embor.2009.266.

[2] IBM, "The brain's architecture, efficiency… on a chip," *IBM Research Blog*, Dec. 19, 2016. https://www.ibm.com/blogs/research/2016/12/the-brains-architecture-efficiency-on-a-chip/ (accessed Apr. 23, 2023).

[3] L. Deng *et al.*, "Rethinking the performance comparison between SNNS and ANNS," *Neural Networks*, vol. 121, pp. 294–307, Jan. 2020, doi: 10.1016/j.neunet.2019.09.005

[4] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997, doi: 10.1016/s0893-6080(97)00011-7.

[5] E. M. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?," *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004, doi: 10.1109/tnn.2004.832719.

[6] G. Gallego *et al.*, "Event-based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020, doi: 10.1109/tpami.2020.3008413.

[7] S. Herculano-Houzel, "The Human Brain in numbers: a Linearly scaled-up Primate Brain," *Frontiers in Human Neuroscience*, vol. 3, no. 31, 2009, doi: 10.3389/neuro.09.031.2009.

[8] "APA Dictionary of Psychology," *dictionary.apa.org*.https://dictionary.apa.org/parallel-distributed-processing

[9] S. R. Brown, "Emergence in the central nervous system," *Cognitive Neurodynamics*, vol. 7, no. 3, pp. 173–195, Nov. 2012, doi: 10.1007/s11571-012-9229-6.

[10] C. W. Johnson, "What are emergent properties and how do they affect the engineering of complex systems?," *Reliability Engineering & System Safety*, vol. 91, no. 12, pp. 1475–1481, Dec. 2006, doi: 10.1016/j.ress.2006.01.008.

[11] P. G. Smelik, "Chapter 1 Adaptation and brain function," *ScienceDirect*, Jan. 01, 1987. https://www.sciencedirect.com/science/article/abs/pii/S0079612308601910#:~:text=If%20adaptation%20includes%20behavioral%20and (accessed Jan. 22, 2023).

[12] T. Trappenberg, *Fundamentals of Computational Neuroscience*. OUP Oxford, 2010. Accessed: Jan. 22, 2023. [Online]. Available: https://books.google.co.uk/books/about/Fundamentals_of_Computational_Neuroscien.html?id=4PDsA1EVCx0C&redir_esc=y

[13] "Lesson Explainer: Nerve Cells | Nagwa," *www.nagwa.com*. https://www.nagwa.com/en/explainers/492141307986/

[14] D. Purves *et al.*, "Summation of Synaptic Potentials," *Neuroscience. 2nd edition*, 2001, [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK11104/

[15] D. Purves *et al.*, "Excitatory and Inhibitory Postsynaptic Potentials," *Nih.gov*, 2016. https://www.ncbi.nlm.nih.gov/books/NBK11117/

[16] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, "Ion Channels and the Electrical Properties of Membranes," *Molecular Biology of the Cell. 4th edition*, 2002, Accessed: Jan. 22, 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK26910/#:~:text=A%20single%20neuron%20might%20typically

[17] M. de Lera Ruiz and R. L. Kraus, "Voltage-Gated Sodium Channels: Structure, Function, Pharmacology, and Clinical Indications," *Journal of Medicinal Chemistry*, vol. 58, no. 18, pp. 7093–7118, May 2015, doi: 10.1021/jm501981g.

[18] "Action potential - The School of Biomedical Sciences Wiki," *Ncl.ac.uk*, 2012. https://teaching.ncl.ac.uk/bms/wiki/index.php/Action_potential

[19] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. Accessed: Jan. 22, 2023. [Online]. Available: https://www.google.co.uk/books/edition/Spiking_Neuron_Models/Rs4oc7HfxIUC?hl=en&gbpv=0

[20] "Action Potential: Depolarization and repolarization ...," *GrepMed*, Oct. 08, 2018. https://www.grepmed.com/images/3822/pathophysiology-neuron-actionpotential-neurology-depolarization (accessed Jan. 22, 2023).

[21] E. Adrian, "THE IMPULSES PRODUCED BY SENSORY NERVE ENDINGS. Part 3. Impulses set up by Touch and Pressure." Accessed: Jan. 22, 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1514868/pdf/jphysiol02525-0003.pdf

[22] W. Bialek, F. Rieke, R. de Ruyter van Steveninck, and D. Warland, "Reading a neural code," *Science*, vol. 252, no. 5014, pp. 1854–1857, Jun. 1991, doi: 10.1126/science.2063199.

[23] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003, doi: 10.1109/tnn.2003.820440.

[24] X. Fang, S. Duan, and L. Wang, "Memristive Hodgkin-Huxley Spiking Neuron Model for Reproducing Neuron Behaviors," *Frontiers in Neuroscience*, vol. 15, no. Front Neurosci. 2021 Sep 23;15:730566. doi: 10.3389/fnins.2021.730566. PMID: 34630019; PMCID: PMC8496503., Sep. 2021, doi: https://doi.org/10.3389/fnins.2021.730566.

[25] "The Hodgkin-Huxley Model," *St-andrews.ac.uk*, 2020. https://www.st-andrews.ac.uk/~wjh/hh_model_intro/ (accessed Sep. 10, 2022).

[26] A. A. Faisal, L. P. J. Selen, and D. M. Wolpert, "Noise in the nervous system," *Nature Reviews Neuroscience*, vol. 9, no. 4, pp. 292–303, Apr. 2008, doi: https://doi.org/10.1038/nrn2258.

[27] K. Diba, H. A. Lester, and C. Koch, "Intrinsic Noise in Cultured Hippocampal Neurons: Experiment and Modeling," *Journal of Neuroscience*, vol. 24, no. 43, pp. 9723–9733, Oct. 2004, doi: https://doi.org/10.1523/JNEUROSCI.1721-04.2004.

[28] Y. Choe, "Hebbian Learning," *Encyclopedia of Computational Neuroscience*, pp. 1–5, 2014, doi: https://doi.org/10.1007/978-1-4614-7320-6_672-1.

[29] F. Ponulak and A. Kasinski, "Introduction to spiking neural networks: Information processing, learning and applications," *Acta Neurobiologiae Experimentalis*, vol. 71, no. 4, pp. 409–433, 2011, Available: https://pubmed.ncbi.nlm.nih.gov/22237491/

[30] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. M. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, vol. 122, pp. 253–272, Feb. 2020, doi: https://doi.org/10.1016/j.neunet.2019.09.036.

[31] P. Mateos-Aparicio and A. Rodríguez-Moreno, "The Impact of Studying Brain Plasticity," *Frontiers in Cellular Neuroscience*, vol. 13, no. 66, Feb. 2019, doi: https://doi.org/10.3389/fncel.2019.00066.

[32] Daniel E. Feldman, "The Spike-Timing Dependence of Plasticity," *Neuron*, vol. 75, no. 4, pp. 556–571, Aug. 2012, doi: https://doi.org/10.1016/j.neuron.2012.08.001.

[33] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, *Principles of Neural Science, Fifth Edition*. McGraw Hill Professional, 2012. Accessed: Jan.22, 2023. [Online]. Available: https://books.google.co.uk/books/about/Principles_of_Neural_Science_Fifth_Editi.html?id=Z2yVUTnlIQsC&redir_esc=y

[34] T. Huff and Prasanna Tadi, "Neuroanatomy, Visual Cortex," *Nih.gov*, Mar. 15, 2019. https://www.ncbi.nlm.nih.gov/books/NBK482504/

[35] Figure 9 from G. Peters, "Aesthetic Primitives of Images for Visualization," *IEEE Xplore*, Jul. 01, 2007. https://ieeexplore.ieee.org/document/4272000 (accessed Apr. 23, 2023).

[36] G. M. Shepherd, *Neurobiology*. Oxford University Press, 1988. Accessed: Jan. 22, 2023. [Online].Available: https://books.google.co.uk/books/about/Neurobiology.html?id=aO9qAAAAMAAJ&redir_esc=y

[37] S. W. Kuffler, "DISCHARGE PATTERNS AND FUNCTIONAL ORGANIZATION OF MAMMALIAN RETINA," *Journal of Neurophysiology*, vol. 16, no. 1, pp. 37–68, Jan. 1953, doi: 10.1152/jn.1953.16.1.37.

[38] D. H. HUBEL and T. N. WIESEL, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–91, 1959, doi: https://doi.org/10.1113/jphysiol.1959.sp006308.

[39]"rcofre – KEOpS," *Project inria*. "Image of On/Off Centre Ganglion cells" https://project.inria.fr/keops/author/rcofre/ (accessed Apr. 23, 2023).

[40] E. R. Kandel, J. H. Schwartz, T. M. Jessell, and T. M. Jessell, *Essentials of Neural Science and Behavior*. McGraw-Hill, 1995.

[41] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural

simulator," *eLife*, vol. 8, Aug. 2019, doi: https://doi.org/10.7554/elife.47314.

[42] "Ornstein-Uhlenbeck Process: Definition." https://www.statisticshowto.com/what-is-the-ornstein-uhlenbeck-process/ (accessed Mar. 15, 2023).

[43] G. Chen, F. Scherr, and W. Maass, "A data-based large-scale model for primary visual cortex enables brain-like robust and versatile visual processing," *Science Advances*, vol. 8, no. 44, p. eabq7592, Nov. 2022, doi: https://doi.org/10.1126/sciadv.abq7592.

[44] D. Heeger, "Poisson Model of Spike Generation," 2000. Available: https://www.cns.nyu.edu/~david/handouts/poisson.pdf

[45] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, Sep. 2000, doi: https://doi.org/10.1038/78829.

[46] E. M. Izhikevich, "Polychronization: Computation with Spikes," *Neural Computation*, vol. 18, no. 2, pp. 245–282, Feb. 2006, doi: https://doi.org/10.1162/089976606775093882.

[47] S. A. Bamford, A. F. Murray, and D. J. Willshaw, "Spike-Timing-Dependent Plasticity With Weight Dependence Evoked From Physical Constraints," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 4, pp. 385–398, Aug. 2012, doi: https://doi.org/10.1109/tbcas.2012.2184285.

[48] "Introduction to Brian part 3: Simulations — Brian 2 2.5.1 documentation," *brian2.readthedocs.io*. https://brian2.readthedocs.io/en/stable/resources/tutorials/3-intro-to-brian-simulations.html

[49] "Black and white Brand Pattern, number 7, angle, white, text png | PNGWing," *www.pngwing.com*. https://www.pngwing.com/en/free-png-ysmxz (accessed Apr. 23, 2023).

[50] "Black and white Circle Pattern, number 8, text, black, number png | PNGWing," *www.pngwing.com*. https://www.pngwing.com/en/free-png-bpvlr (accessed Apr. 23, 2023).

[51] D. Halliday, "A framework for the analysis of mixed time series/point process data-theory and application to the study of physiological tremor, single motor unit discharges and electromyograms," *scholar.google.co.uk*, Oct. 01, 1995. https://scholar.google.co.uk/citations?view_op=view_citation&hl=en&user=XPQlSyAAAAAJ&citation_for_view=XPQlSyAAAAAJ:u5HHmVD_uO8C (accessed Apr. 23, 2023). Sourced MATLAB code and spike timing data

[52] S. Habenschuss, Z. Jonke, and W. Maass, "Stochastic Computations in Cortical Microcircuit Models," *PLoS Computational Biology*, vol. 9, no. 11, p. e1003311, Nov. 2013, doi: https://doi.org/10.1371/journal.pcbi.1003311.

[53] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, "Asynchronous, Photometric Feature Tracking using Events and Frames," *arXiv:1807.09713 [cs]*, Jul. 2018, doi: https://doi.org/10.1007/978-3-030-01258-8_46.

[54]"Recordings and Datasets — Metavision SDK Docs 4.0.1 documentation," docs.prophesee.ai. https://docs.prophesee.ai/stable/datasets.html#chapter-datasets (accessed May 05, 2023).

[55] "Industrial machinery (OEM) from design to installation," *NMH s.r.o.* https://www.nmh-sro.com/markets/industrial-machinery/ (accessed Apr. 23, 2023).

[56] “OpenCV: Bibliography,” *docs.opencv.org*. https://docs.opencv.org/4.x/d0/de3/citelist.html (accessed Apr. 24, 2023).

[57]“OpenCV: Getting Started with Videos,” *docs.opencv.org*. https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html (accessed May 05, 2023).

[58] NI Business Info, “Five stages of project management,” *nibusinessinfo.co.uk*, Mar. 12, 2019. https://www.nibusinessinfo.co.uk/content/five-stages-project-management

[59]“TeamGantt,” *app.teamgantt.com*. https://app.teamgantt.com/projects/gantt?ids=3308592 (accessed Oct. 06, 2022).

[60]“The n-MNIST handwritten digit dataset,” *csc.lsu.edu*. https://csc.lsu.edu/~saikat/n-mnist/ (accessed May 10, 2023).

[61] “Number Clipart Transparent - Number 3 With No Background - Free Transparent PNG Clipart Images Download. ClipartMax.com,” *ClipartMax.com*. https://www.clipartmax.com/middle/m2i8A0m2b1K9A0A0_number-clipart-transparent-number-3-with-no-background/ (accessed Apr. 23, 2023).

[62] IMGBIN.com, “Number Numerical Digit 1st Kerry Scouts Stencil PNG - Free Download,” *IMGBIN.com*. https://imgbin.com/png/TdLpGaxY/number-numerical-digit-1st-kerry-scouts-stencil-png (accessed Apr. 23, 2023).

[63] “Brand Logo Pattern, number 0, text, logo, number png | PNGWing,” *www.pngwing.com*. https://www.pngwing.com/en/free-png-bylgj (accessed Apr. 23, 2023).

[64] “Black Line Background - Unlimited Download. cleanpng.com.,” *cleanpng.com*. https://www.cleanpng.com/png-number-1-png-22597/ (accessed Apr. 23, 2023).

[65] “Black Line Background - Unlimited Download. cleanpng.com.,” *cleanpng.com*. https://www.cleanpng.com/png-number-2-png-41726/ (accessed Apr. 23, 2023).

[66] “Logo Line Angle Brand, Number 4, angle, white, text png | PNGWing,” *www.pngwing.com*. https://www.pngwing.com/en/free-png-bpqzo (accessed Apr. 23, 2023).

[67] “Download Icon Number 5 PNG Transparent Background, Free Download #24789 - FreeIconsPNG,” *freeiconspng.com*. https://www.freeiconspng.com/img/24789 (accessed Apr. 23, 2023).

[68] “Symbol Numerical digit Number, Number 6, text, number, black png | PNGWing,” *www.pngwing.com*. https://www.pngwing.com/en/free-png-byamk (accessed Apr. 23, 2023).

[69] “Brian2 documentation,” *Brian2*. https://readthedocs.org/projects/brian2/downloads/pdf/stable/

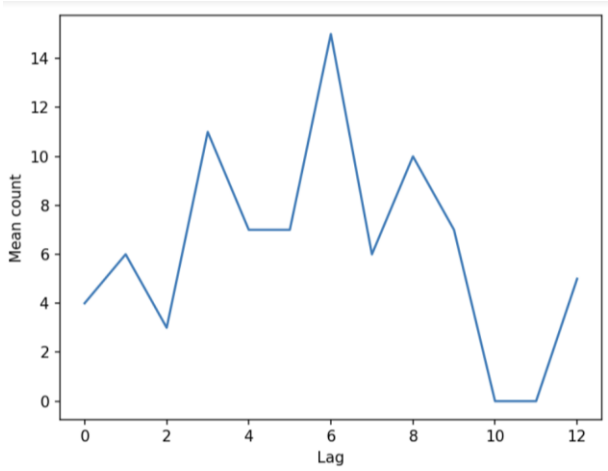
APPENDIX

A. Cross-correlation function verification

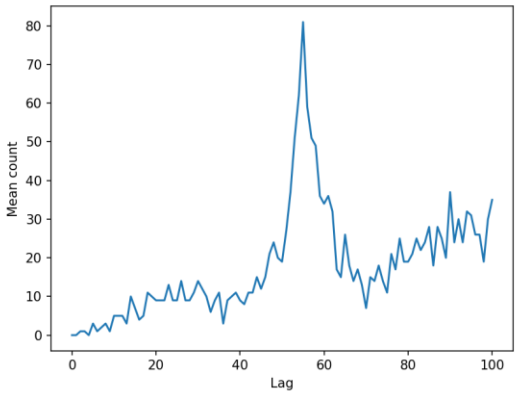
The first iteration of the cross-correlation function is shown below. The understanding of the cross-correlation function wasn’t adequate for the correct implementation of the function. The way of centring the graph and calculating the time difference was incorrect.

```
InputSpikes =[1,2,3,4,5,6,7,8,9]
OutputSpikes=[11,34,37,46,51,52,65,66,78]

def cross_correlation_func(InputSpikes,OutputSpikes):
    #Largest time difference gives the range of which the
    # cross correlation function will look at
    Max_time_dif =OutputSpikes[len(OutputSpikes)-1] - InputSpikes[0]
    # number of sampling intervals
    num_intervals =12
    # Return integer number of width of each interval
    bin_width = int(Max_time_dif/num_intervals)
    lag_array=[]
    # Assign lag array same number of indexs as the number of intervals
    for j in range(num_intervals + 1):
        lag_array.append(0)

    time_lag = 0
    print("Bin width is: ",bin_width)
    # go through the entire list and then compare each index in input spikes to every value in output spik
    for i in InputSpikes:

        for j in OutputSpikes:
            # minus the reference spike (i) and the comparison spike (j)
            # if whatever value this falls between , get the correct index in the
            # lag_array() and increase the count for this index
            # for an example of -50ms to +50 ms if the j-i gives a value of 40ms
            # increase the value stored in the index that represents 40ms in the lag array
            time_lag = i-j
            #
            lag_index = int(time_lag/bin_width)
            print("Lag Index is",lag_index)
            lag_array[lag_index] += 1
```

The output is shown below to some random inputs. The graph is incorrect as it doesn't calculate the correlation of spikes correctly.

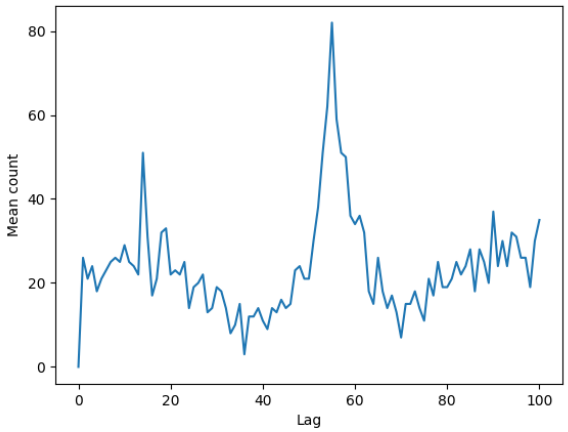


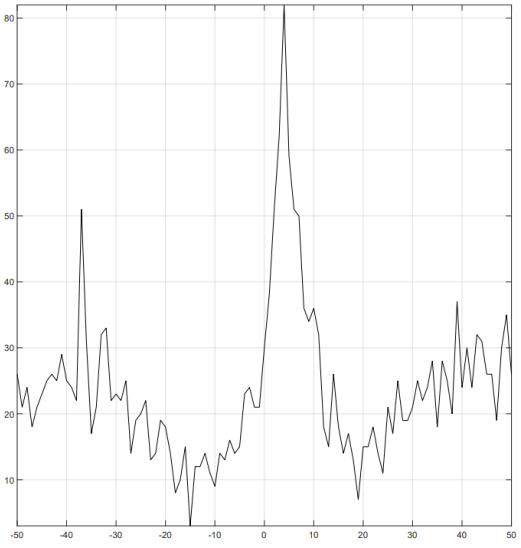
David Halliday's [50] MATLAB function and data from previous publications were used to create the python version of the function. The first iteration is shown below. The output is shown to be incorrect as only the right side of the graph is correct, when compared to David's plot.



When correcting the use of MATLAB's find function using the code below.

```
# Python version of Matlabs "find()" function
# Will find the index's in the output array
#that meets the conditions needed.
# Since the find function finds ALL indexs
# that meet these requirements in pyhton
#this is translated to appending an array after going through the array entirely

# If an index is within this rnage of -binsize +0.5 -----X----binsize +0.5
# add to array
for j in OutputSpikes:
    if(int(j) >lag_start):
        if(int(j) <=lag_stop):
            outputindex.append(out_ind)

    out_ind +=1
```





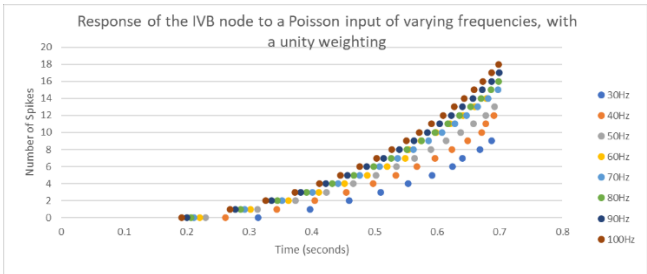
[50]

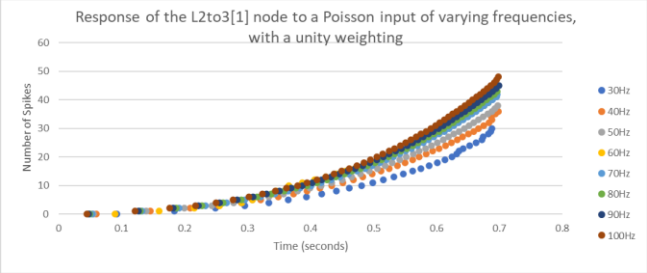
The plot above from David Halliday [50] matches to the final function output showing the function works correctly.

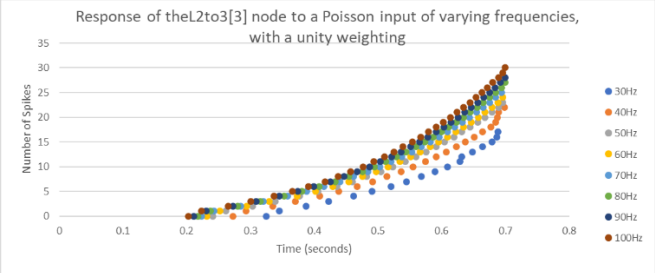
B. Graphs : Poisson inputs from 30-100Hz

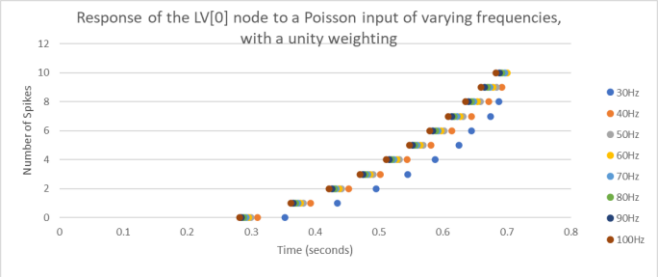
Poisson input graphs show that the model has higher activity under higher levels of noise. This contrasts to other tests like sinusoidal where the model struggles to achieve high activity under higher frequencies.

It should be noted that activity may be higher due to noise causing erroneous spiking activity, that wouldn't have been present without the noise.



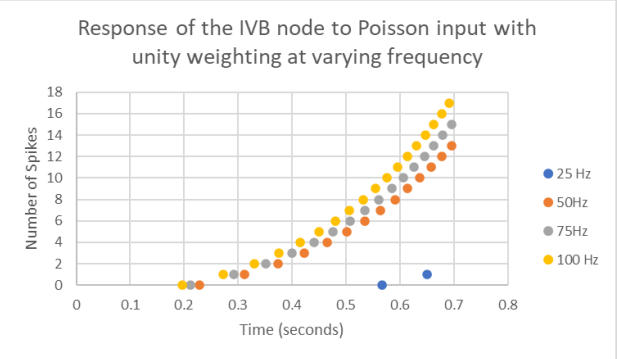


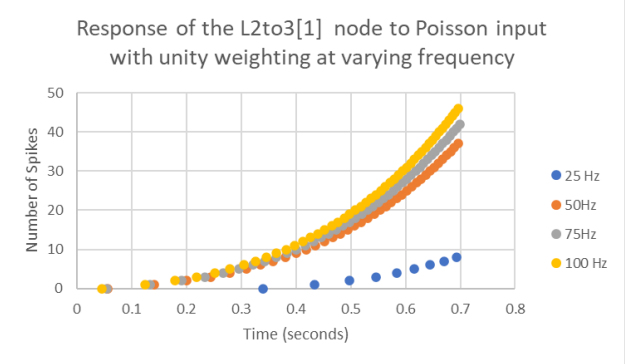


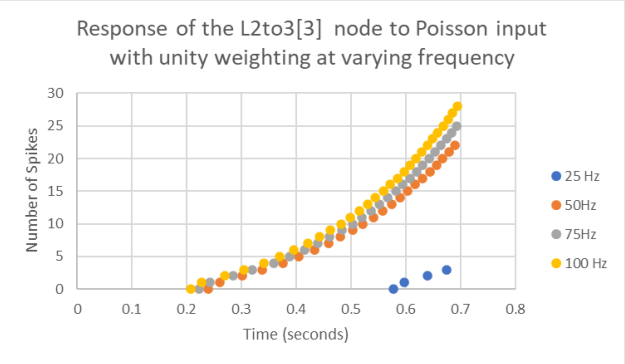


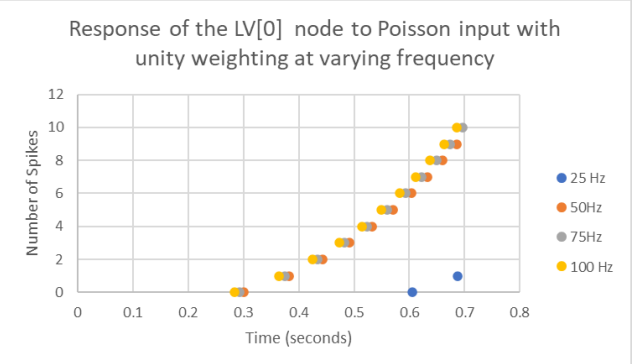
C. Graphs of Poisson plots with changing weights 1-5

The results shown by graphs in this section are that with a higher weighted noise input, the model's behaviour was able to better.

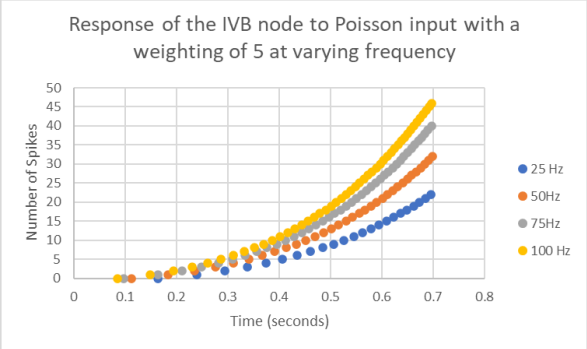


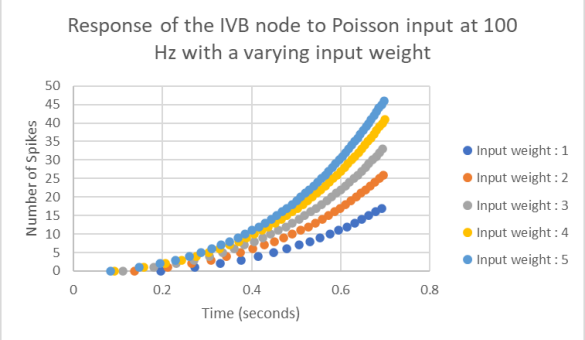


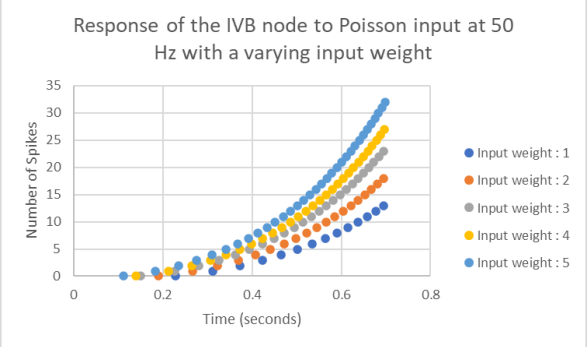


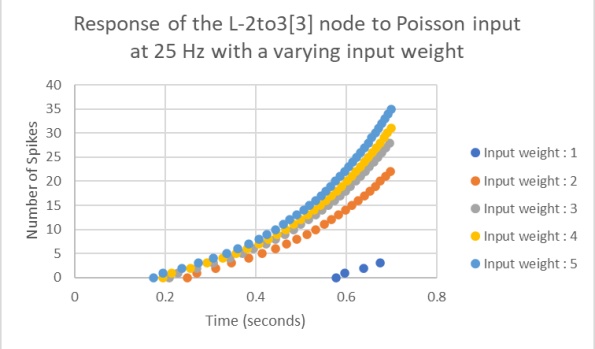


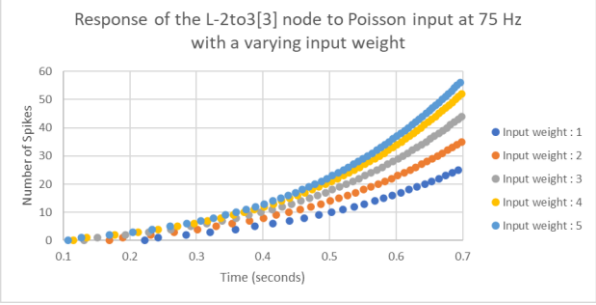
The subsequent graphs show this relationship better. As even at low frequencies of 25Hz the higher weighted inputs granted more active spiking activity.







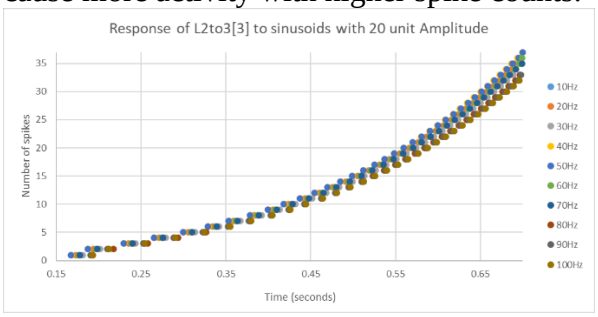


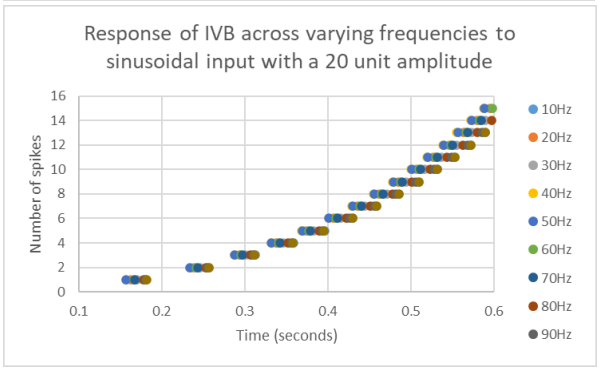


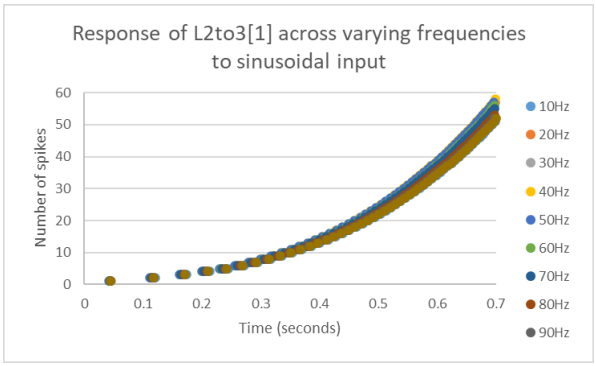
D. Graphs: Sinusoidal 25-100Hz

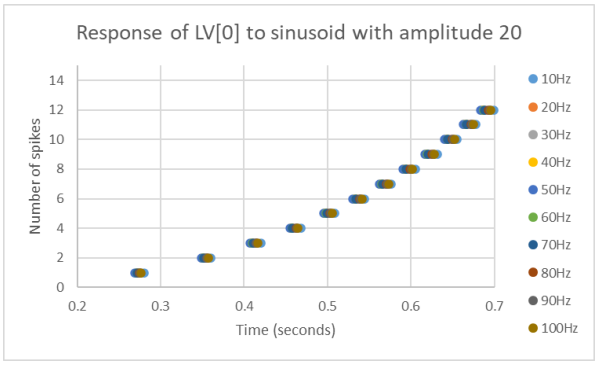
The graphs in this section show that across the operating ranges of the experiment that the activity in sinusoidal testing, is within a specific range.

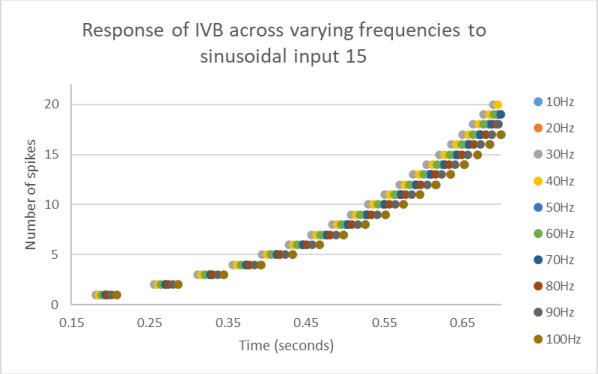
In this range it is seen that lower frequency inputs cause more activity with higher spike counts.

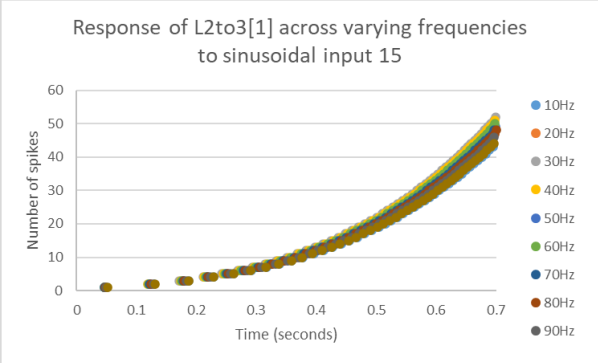


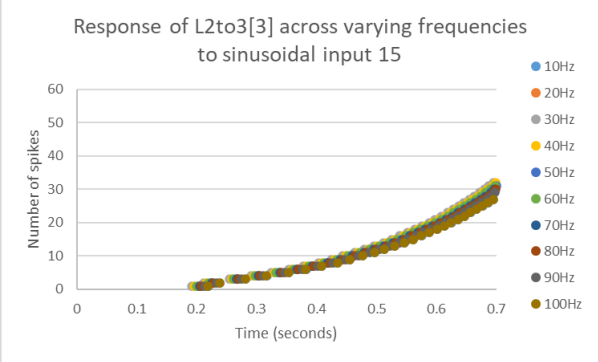


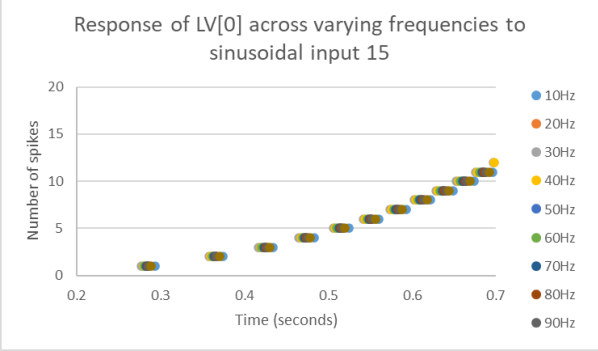


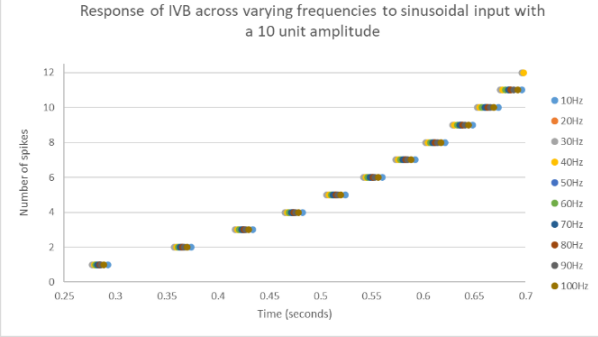


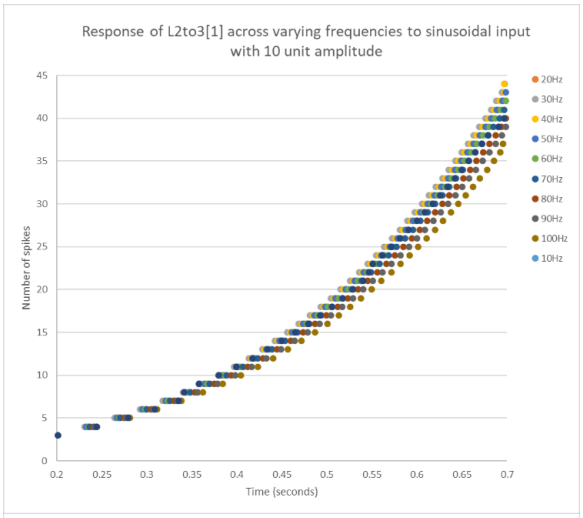

Response of IVB across varying frequencies to sinusoidal input 15
Number of spikes
Time (seconds)
10Hz
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz
100Hz

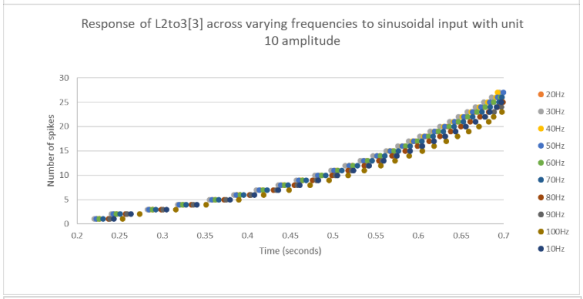

Response of L2to3[1] across varying frequencies to sinusoidal input 15
Number of spikes
Time (seconds)
10Hz
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz

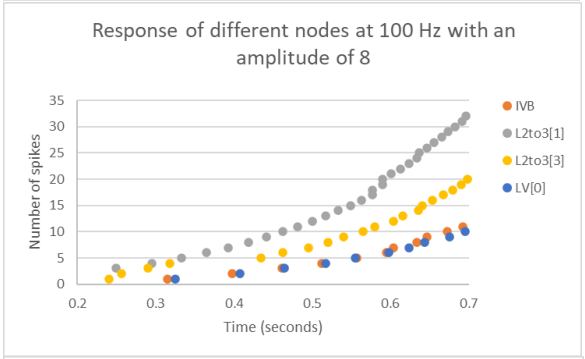

Response of L2to3[3] across varying frequencies to sinusoidal input 15
Number of spikes
Time (seconds)
10Hz
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz
100Hz

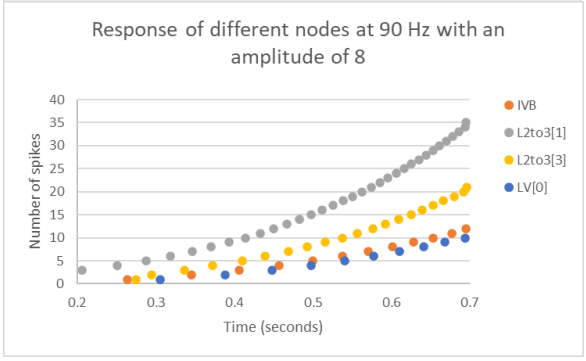

Response of LV[0] across varying frequencies to sinusoidal input 15
Number of spikes
Time (seconds)
10Hz
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz

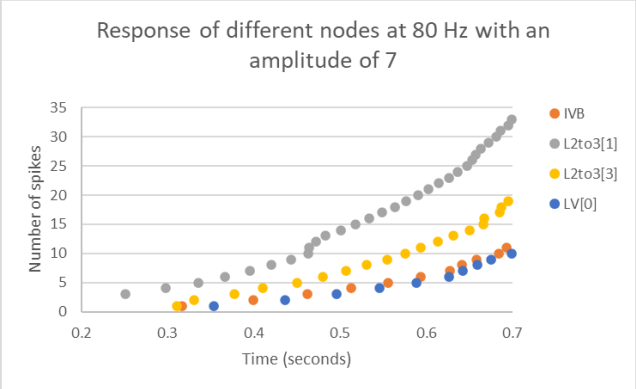

Response of IVB across varying frequencies to sinusoidal input with a 10 unit amplitude
Number of spikes
Time (seconds)
10Hz
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz
100Hz

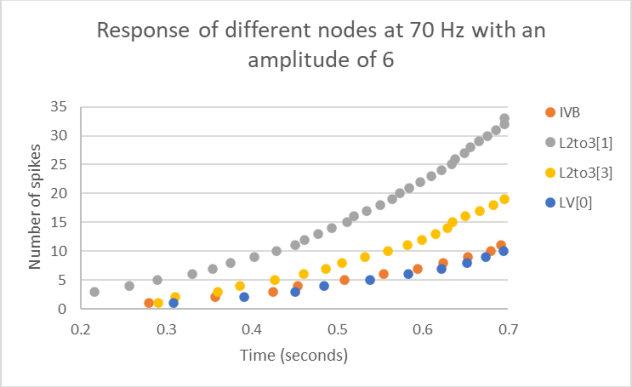

Response of L2to3[1] across varying frequencies to sinusoidal input with 10 unit amplitude
Number of spikes
Time (seconds)
20Hz
30Hz
40Hz
50Hz
60Hz
70Hz
80Hz
90Hz
100Hz
10Hz

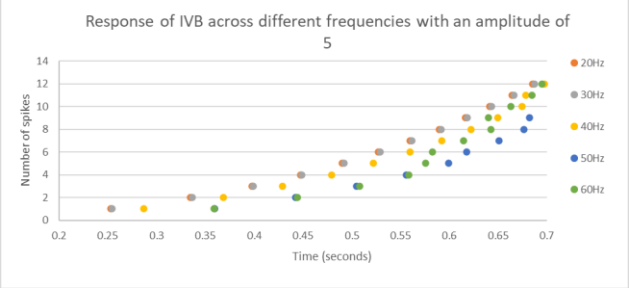

Response of L2to3[3] across varying frequencies to sinusoidal input with unit 10 amplitude
Number of spikes
Time (seconds)

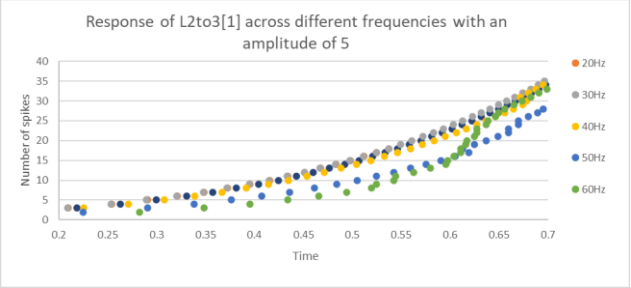

Response of different nodes at 100 Hz with an amplitude of 8
Number of spikes
Time (seconds)
IVB
L2to3[1]
L2to3[3]
LV[0]

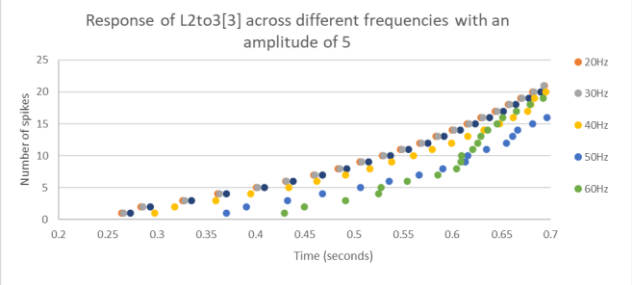

Response of different nodes at 90 Hz with an amplitude of 8
Number of spikes
Time (seconds)
IVB
L2to3[1]
L2to3[3]
LV[0]

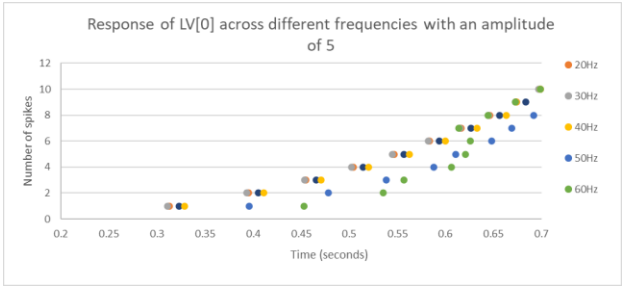








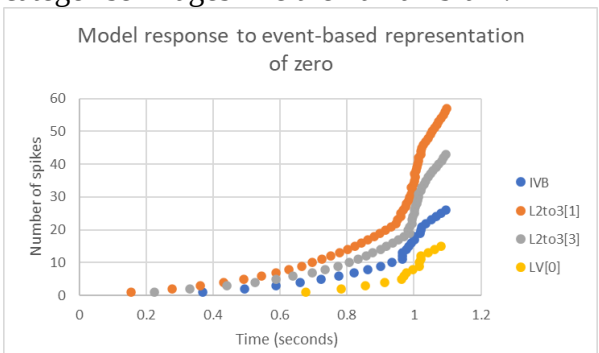


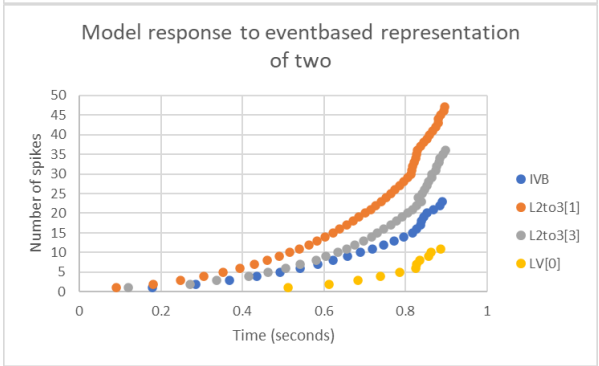


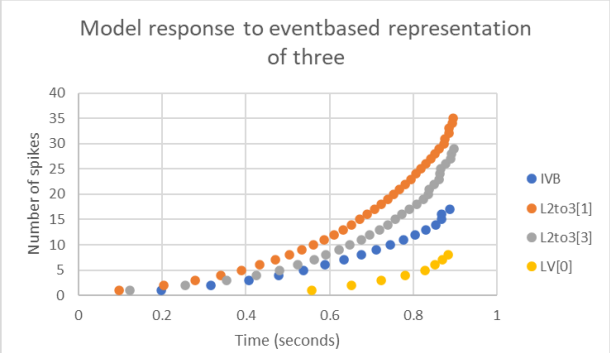
E. Graphs: Vision

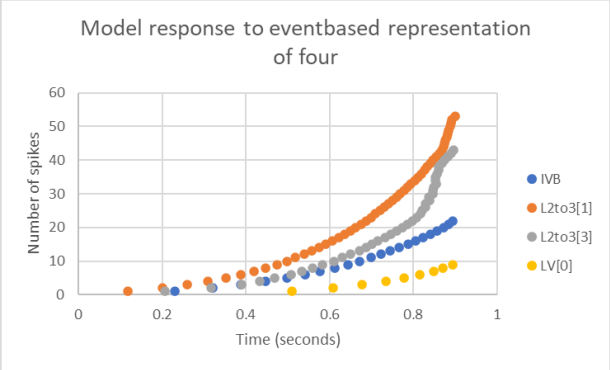
The results from viewing the responses from the model to event-based numbers achieves understanding of whether the model responds similarly to every image it is presented with. This is seen to be false looking at the graphs in this section.

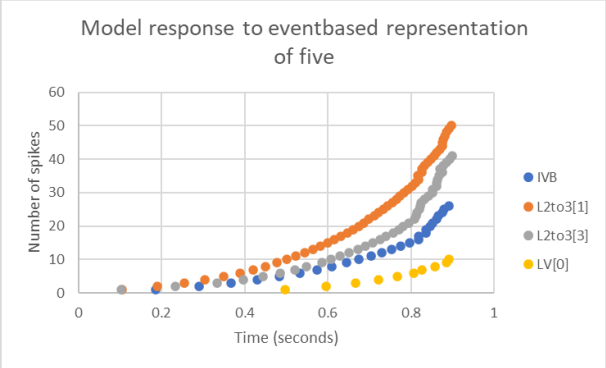
It is a positive for the project as it suggests the model is capable of achieving low level processing (edge detection) and possibly high-level processing, which would allow it to categorise images like the human brain.

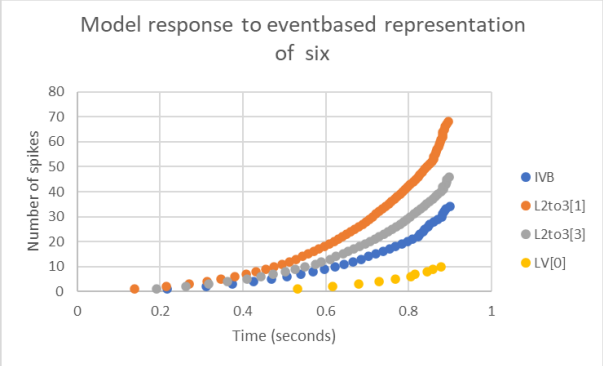


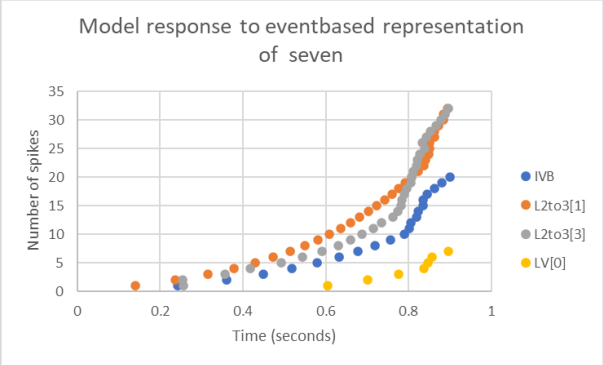


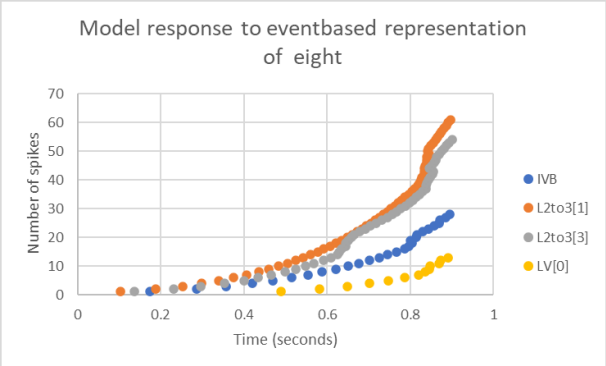


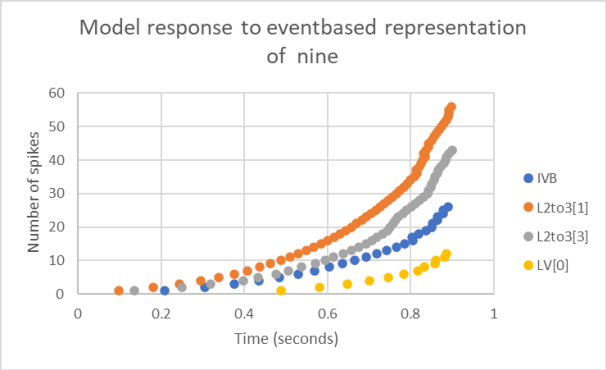








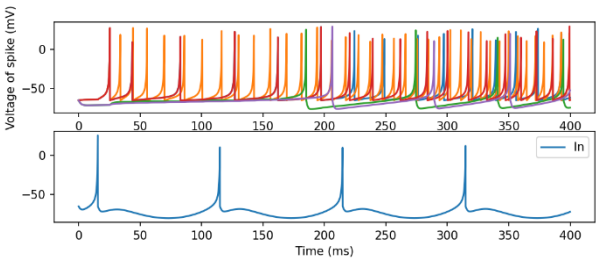


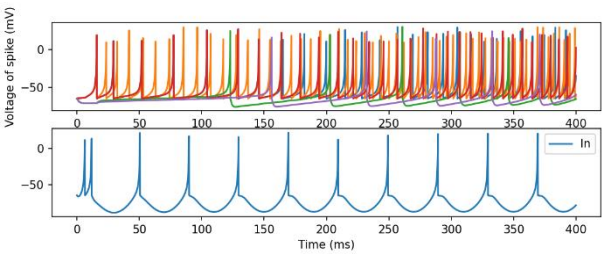


F. Testing Phase 1 Izhikevich

The sinusoidal input was used to verify its behaviour alongside Poisson testing seen in the report. This is included as sinusoidal inputs found the most errors in the projects design. The included plots showcase across the experimental range that the Phase 1 LIF functions correctly.

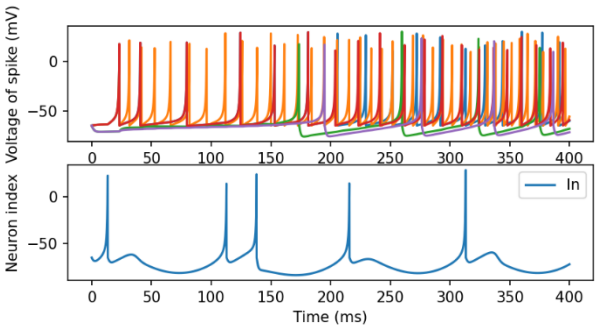
10hz 8 amplitude

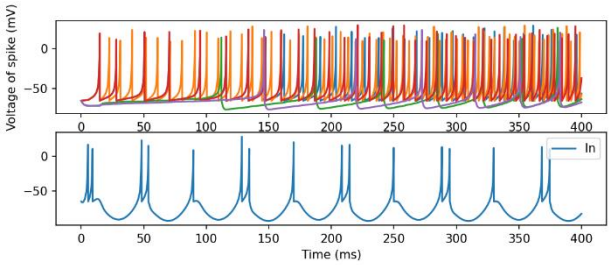




10 Hz 10 amplitude

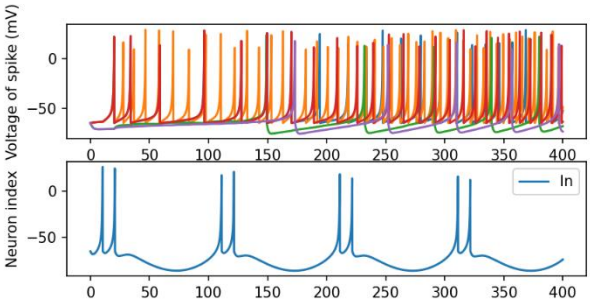
20 amplitude 25 Hz

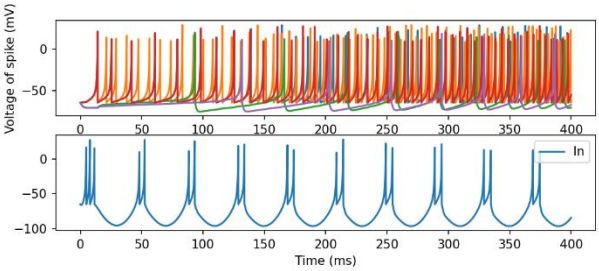




15 amplitude 10 Hz

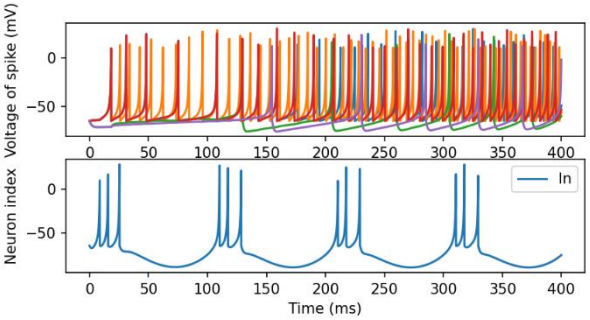
25 amplitude 25Hz

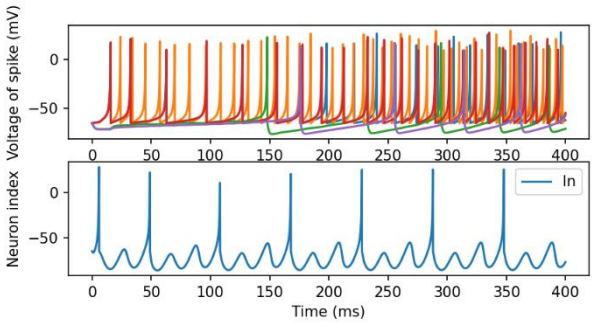




20 amplitudes to 10 Hz

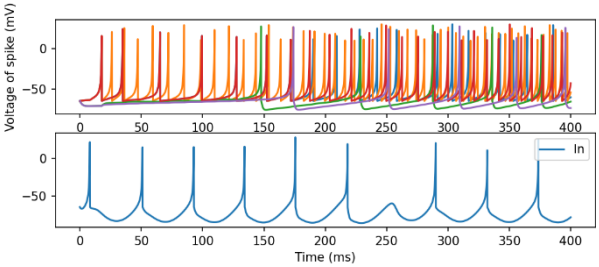
10 amplitude 50 Hz



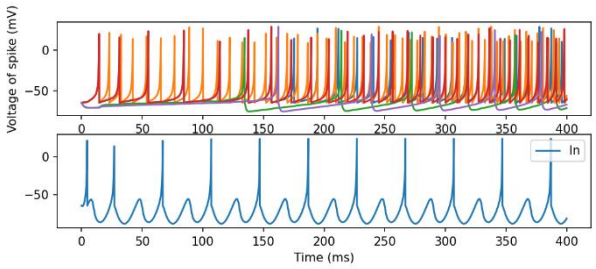


10 amplitude 25 Hz

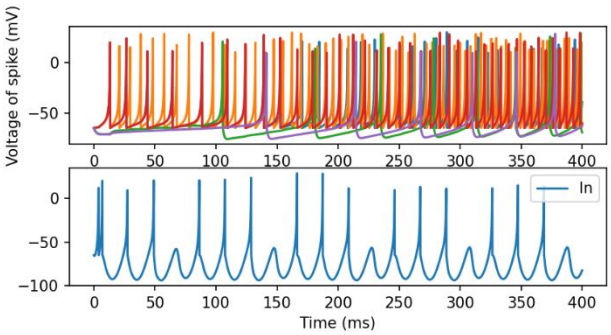
15 amplitude 50 Hz



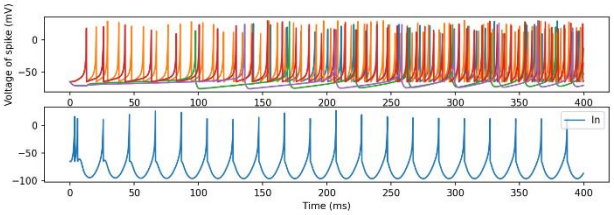
15 amplitude 25 Hz



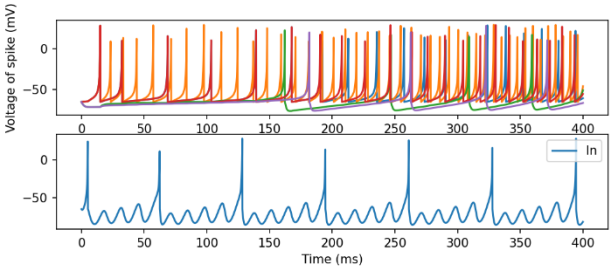
20 amplitude 50 Hz



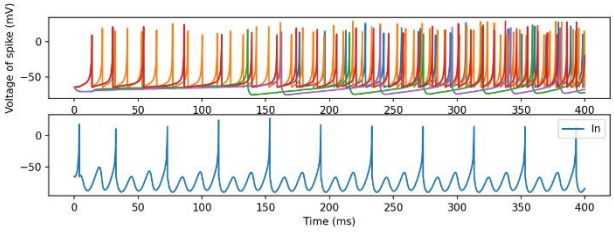
25 amplitude 50 Hz



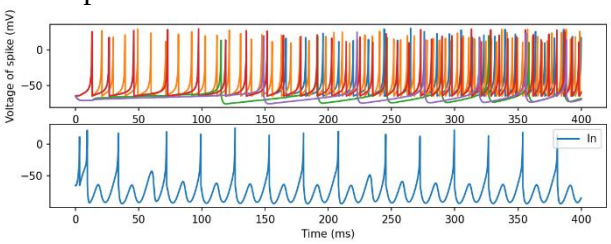
10 Amplitude 75Hz



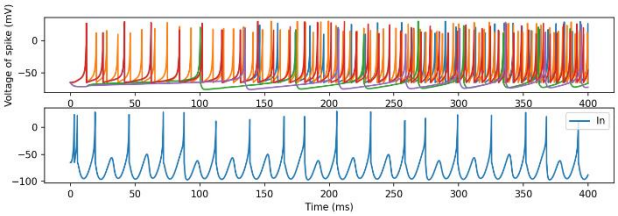
15 Amplitude 75Hz



20 Amplitude 75Hz



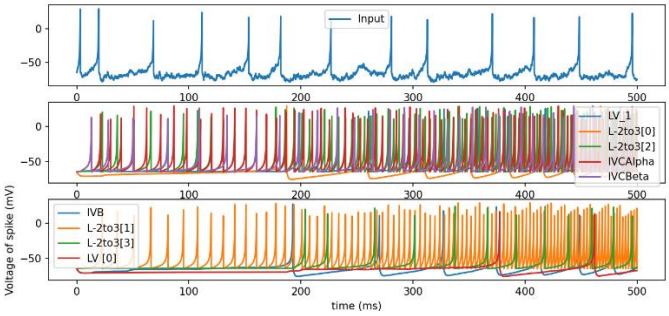
25 Amplitude 75Hz



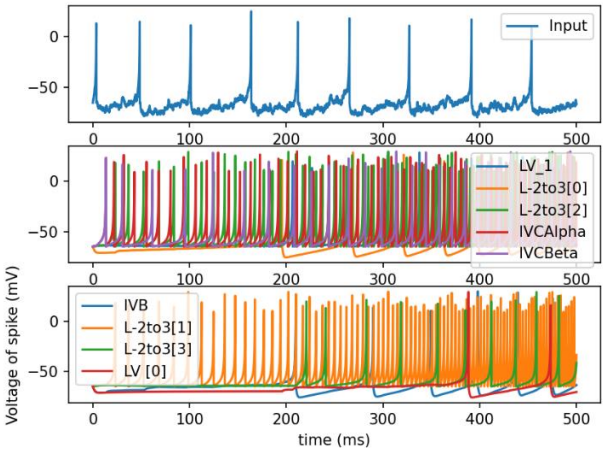
G. Phase 2 Software output Poisson plots

Showcases some plots from changing the weight with Poisson input. Shows how increasing the weight increases the response from the Phase 2 model.

100 Hz weight 1

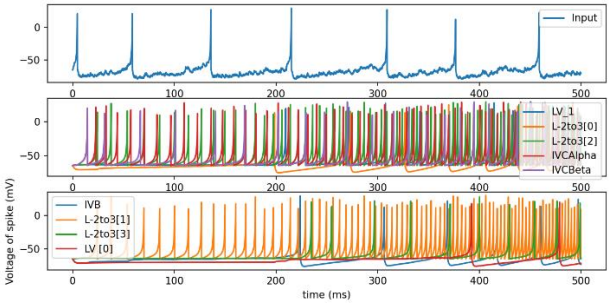


75Hz weight 1

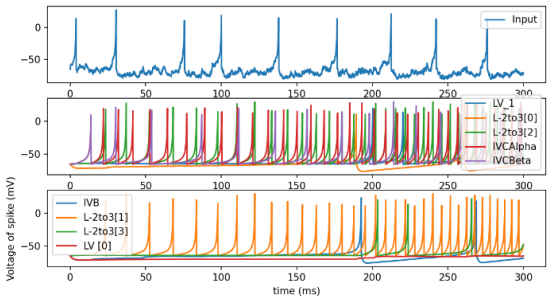


Benjamin McCann

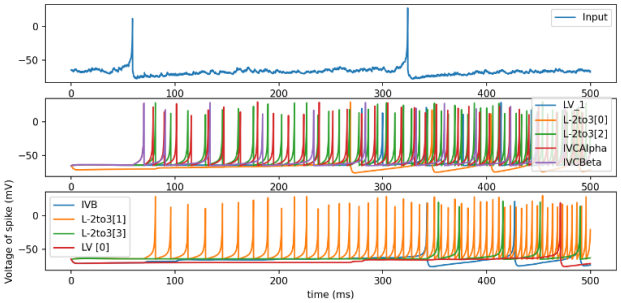
50 Hz weight 1



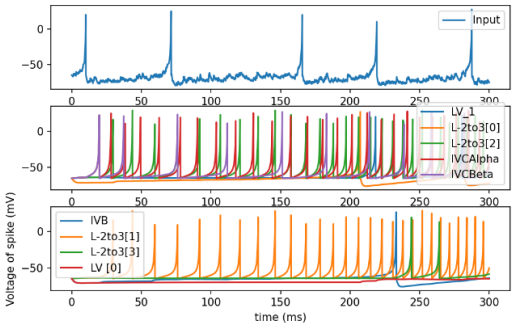
50 Hz weight 2



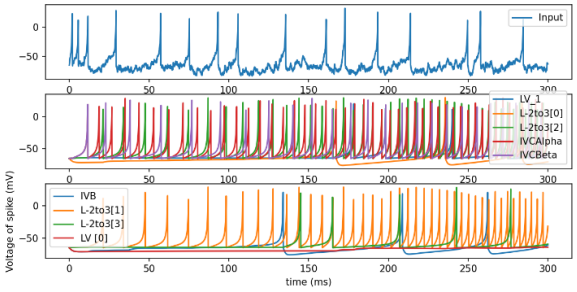
25Hz weight 1



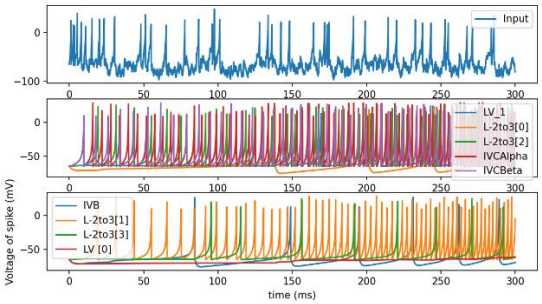
25Hz weight 2



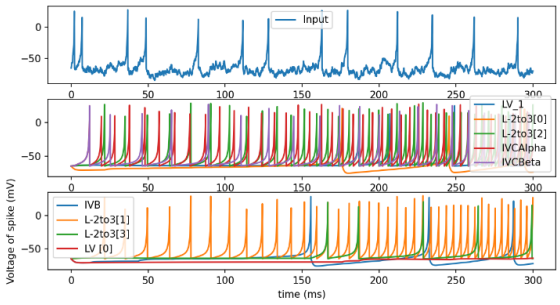
Poisson input at 100Hz weighted input of 2



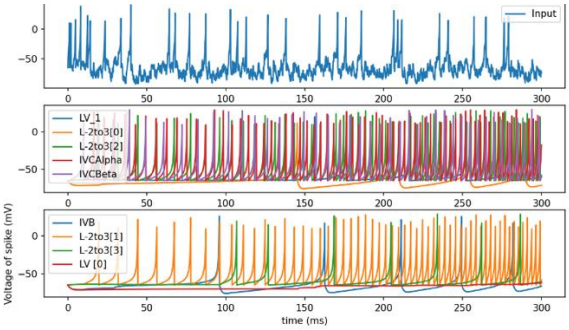
100 Hz weight 5



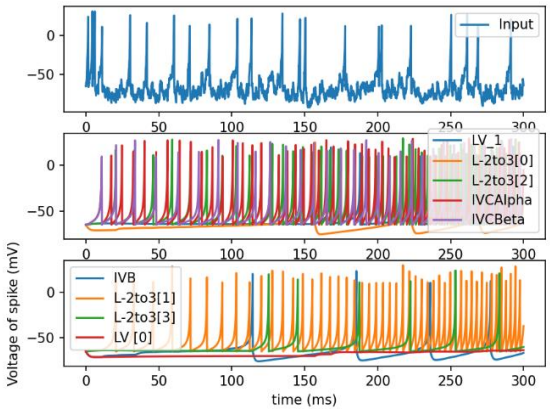
75Hz weight 2



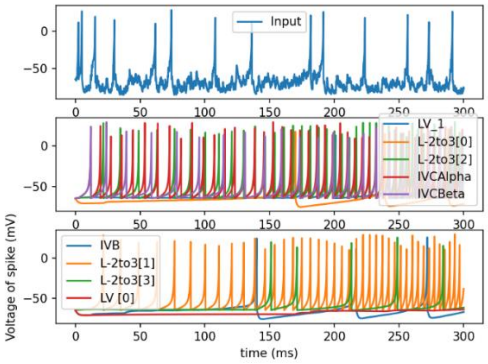
75 Hz weight 5



50 Hz weight 5



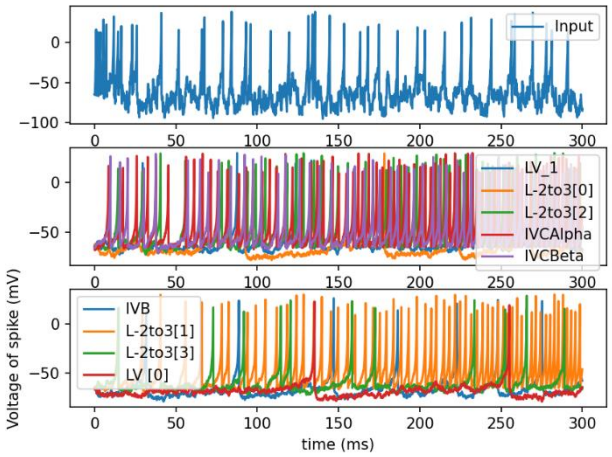
25 Hz weight 5



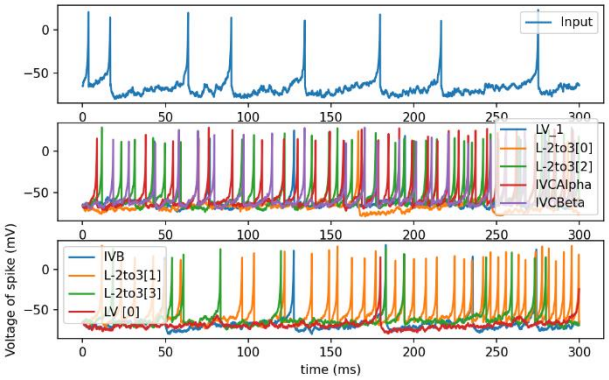
H. Phase 2 Intense weight and noise testing under Poisson input

These tests show the noise resilience of the model. As it still functions under elevated levels of noise and strong inputs.

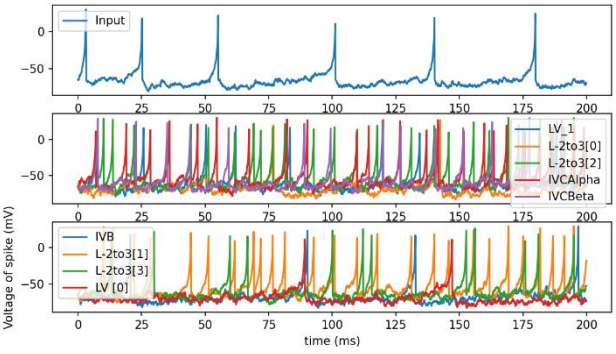
Standard deviation from the mean of 5 in Ornstein-Ulhnbeck equation with Poisson weighted input of 5 at 100Hz.



Weighted Poisson input of 1 at 100 Hz with Standard deviation value sigma of 5 in the noise equation.



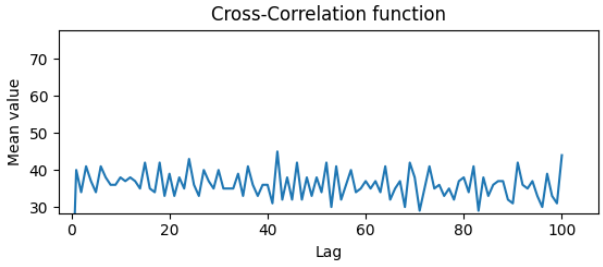
Standard deviation of 10 - large deviation of the mean value. Poisson input at 100Hz with a Weight 1



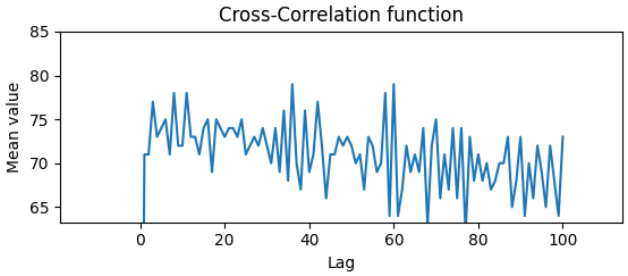
I. *Cross-correlation of Phase 2 model under Poisson input*

All high counts due to chance and show no statistically significant behaviour. They show the model has a sparse firing rate.

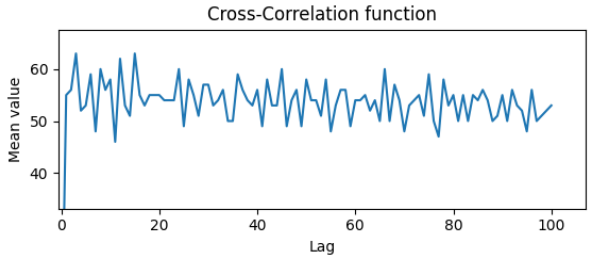
Input to IVB



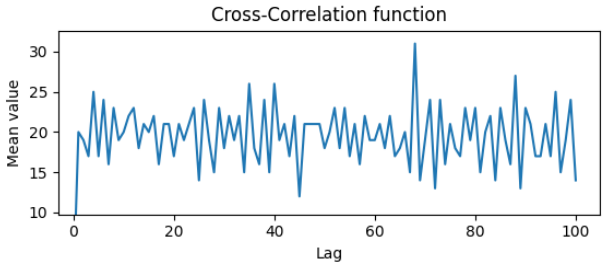
Input to L2to3[1]



Input to L2to3[3]



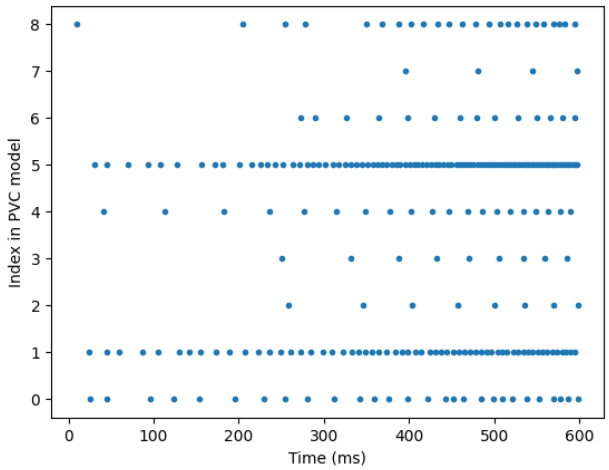
Input to LV[0]

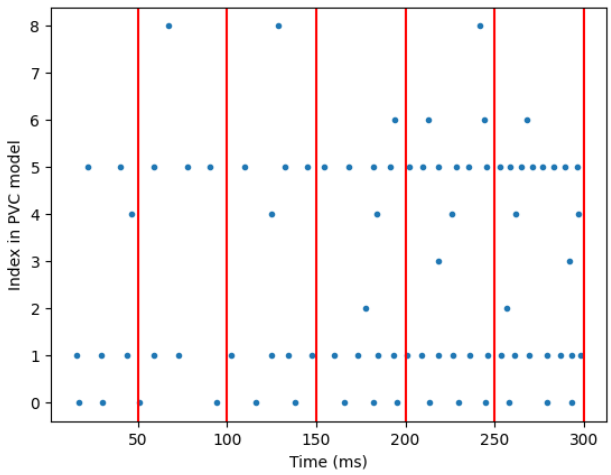


J. *Phase 2 Synchrony Plots under Sinusoidal input*

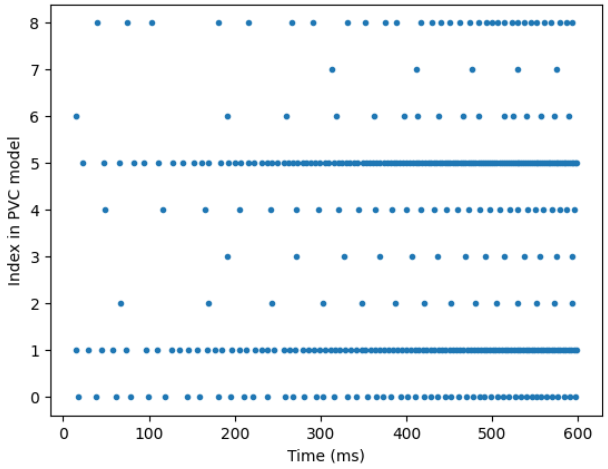
Poisson synchrony plots not shown as plots in report best represent their behaviour across whole range. This test shows synchronous behaviour is seen across testing ranges. However, there is no correlation or definitive trend. Therefore, a factual statement on the models synchronous behaviour cannot be made.

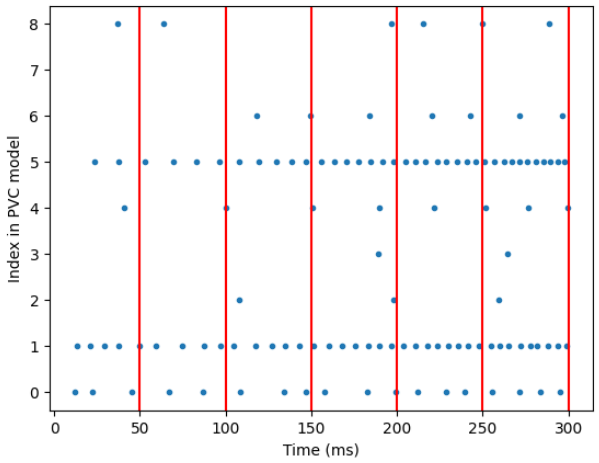
Amplitude : 5 at 25Hz

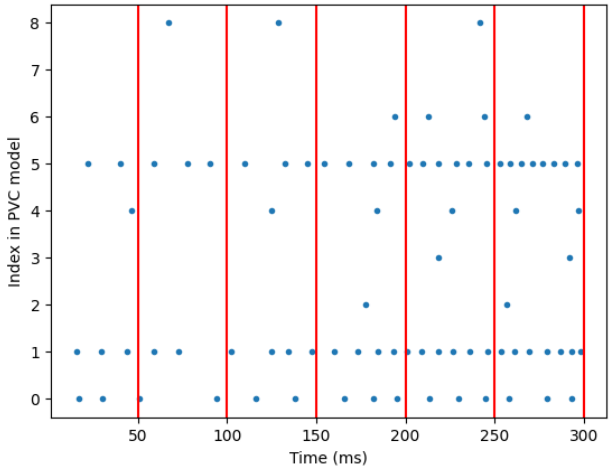




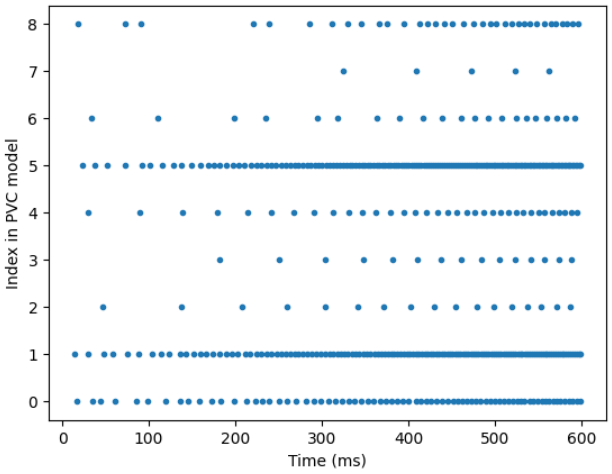
Amplitude : 10 at 25Hz



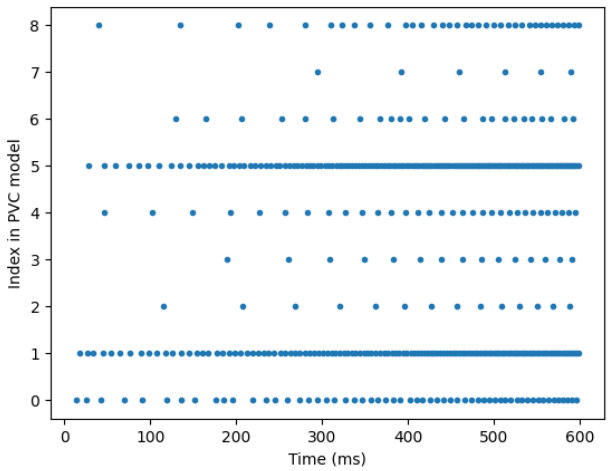


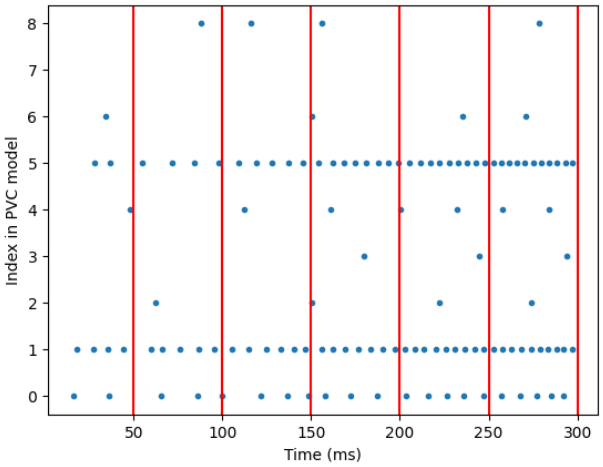


Amplitude : 20 at 25Hz

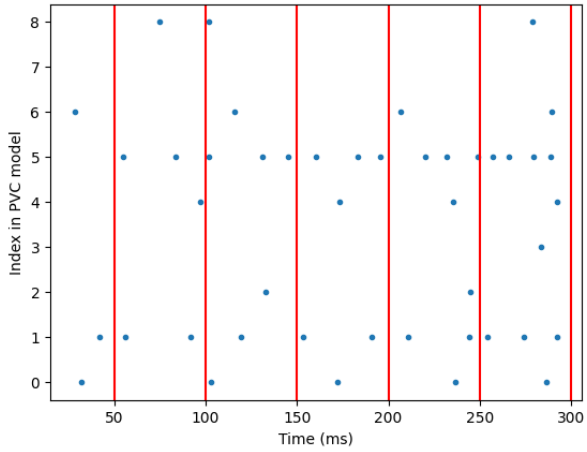


Amplitude : 15 at 25Hz

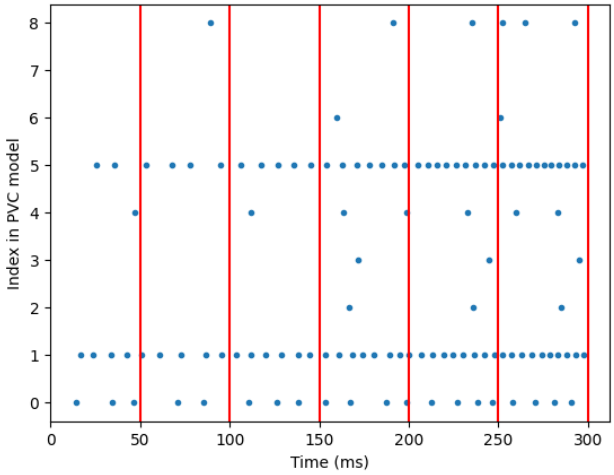




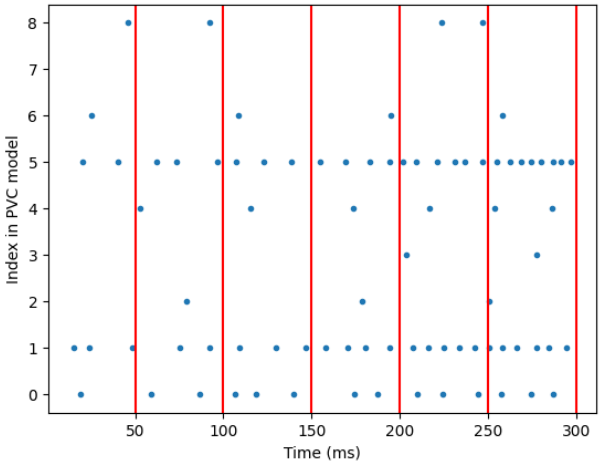
Amplitude : 5 at 50Hz



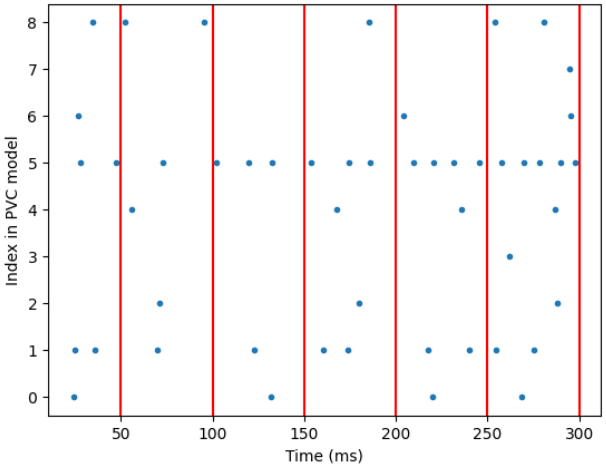
Amplitude: 20 at 50Hz



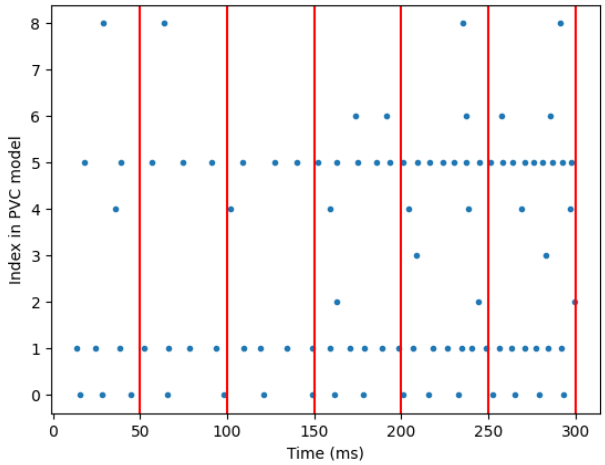
Amplitude: 10 at 50Hz



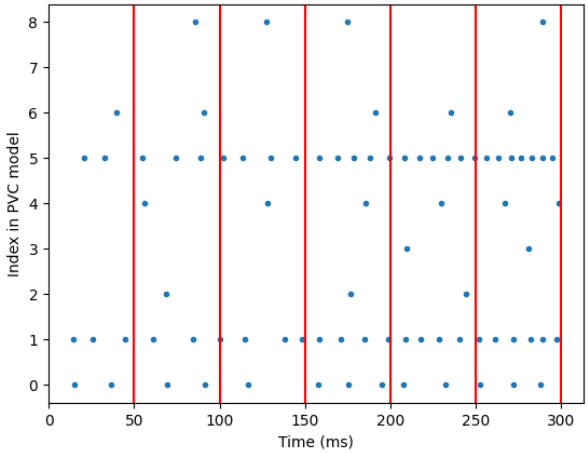
Amplitude: 6 at 75Hz



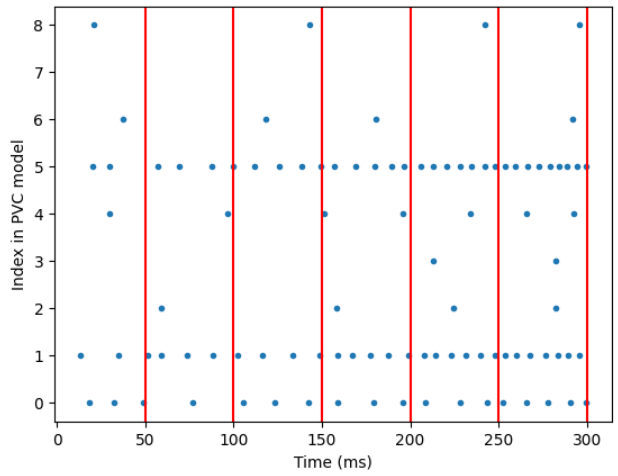
Amplitude: 15 at 50Hz



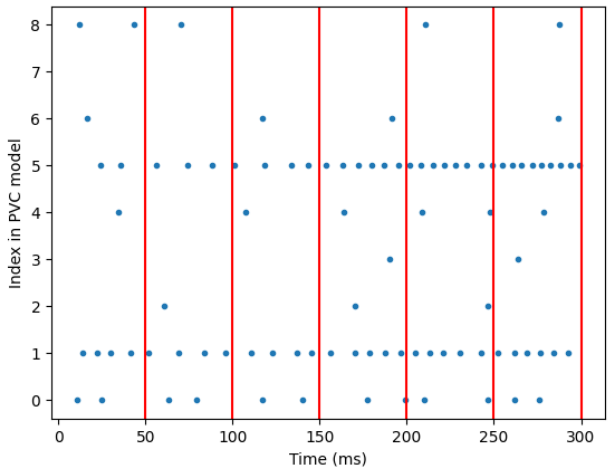
Amplitude: 10 at 75Hz



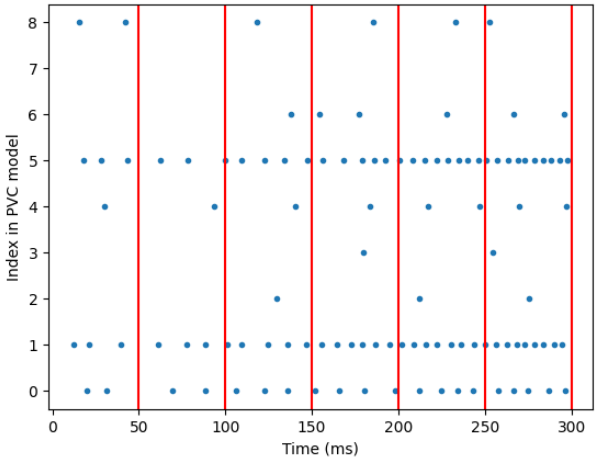
Amplitude: 15 at 75Hz



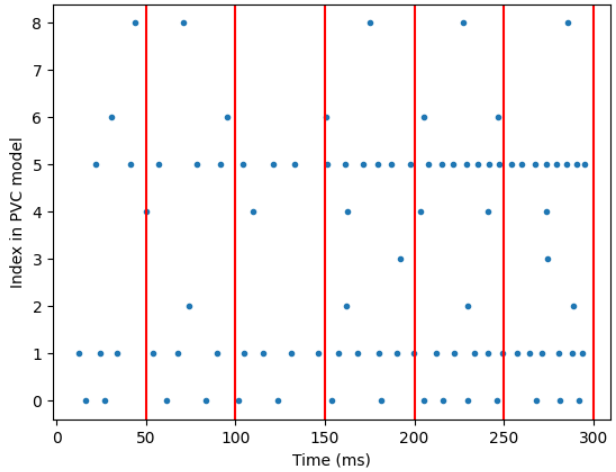
Amplitude: 10 at 100Hz



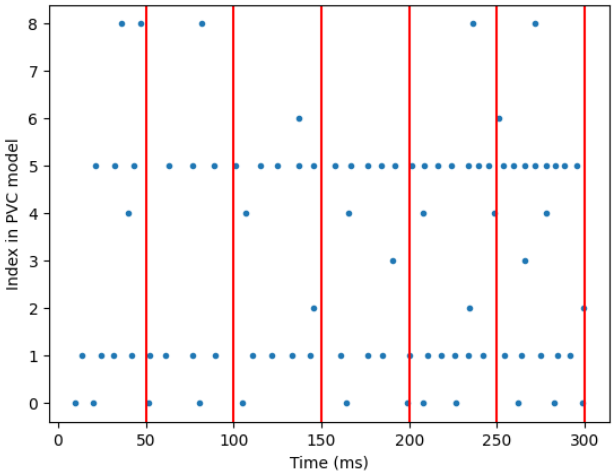
Amplitude: 20 at 75Hz



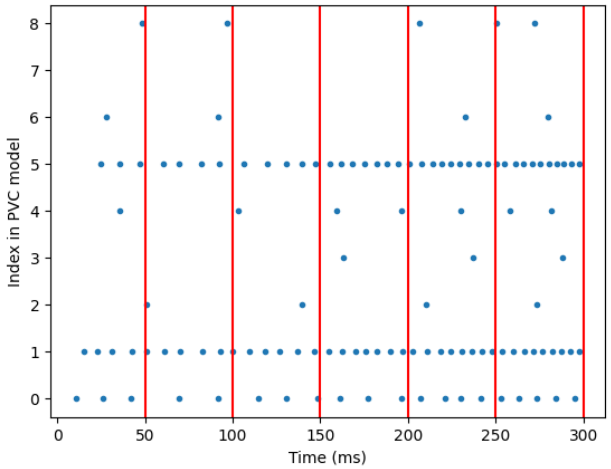
Amplitude: 15 at 100Hz



Amplitude: 9 at 100Hz



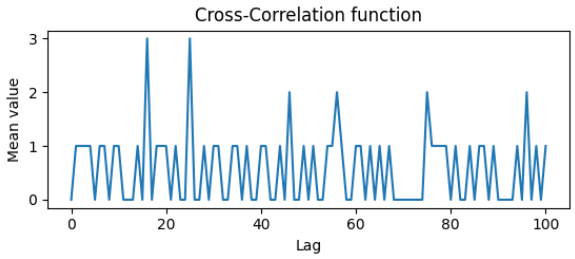
Amplitude: 20 at 100Hz



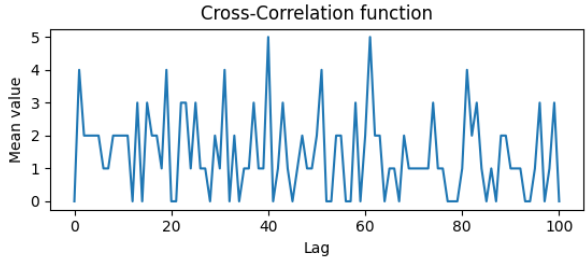
K. Phase 2 Cross-correlation of model under Sinusoidal input

All ran at 20 amp at 50 Hz with unity weight, due to this range producing the most activity in the model, Appendix D. All show model has sparse firing rate as there is no behaviour of statistical significance.

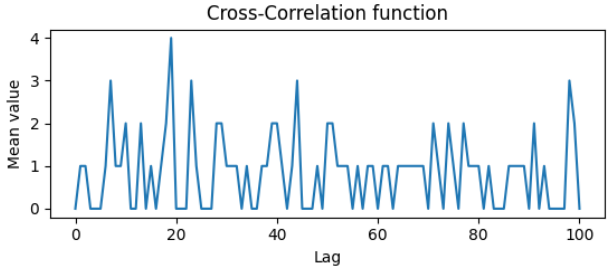
Input-IVB



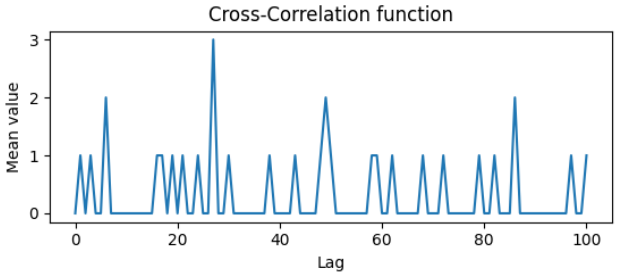
Input – L2to3[1]



Input-L2to3[3]



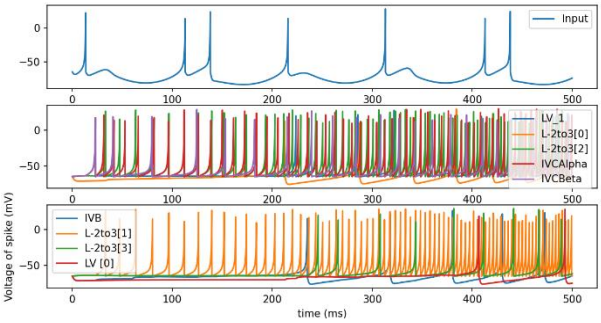
Input to LV[0]



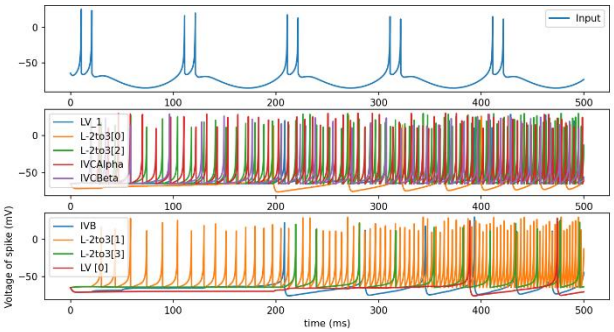
L. Phase 2 Software output Sinusoidal plots

These plots are the output of running the sinusoidal experiment on the phase 2 model. They are here to show firing rates of the model while showcasing the model to be functioning correctly.

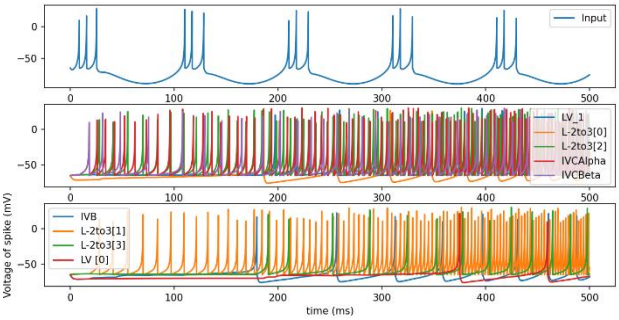
Amplitude: 10 at 10Hz



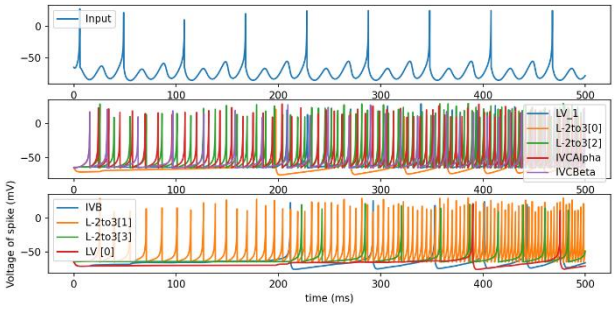
Amplitude: 15 at 10 Hz



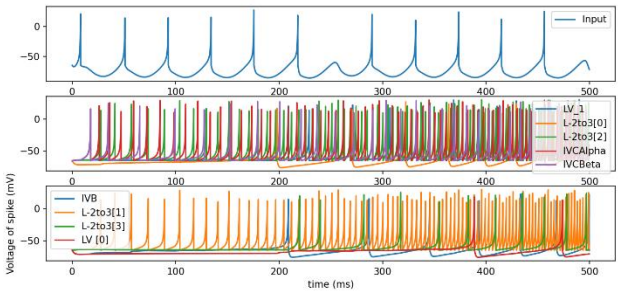
Amplitude: 20 at 10 Hz



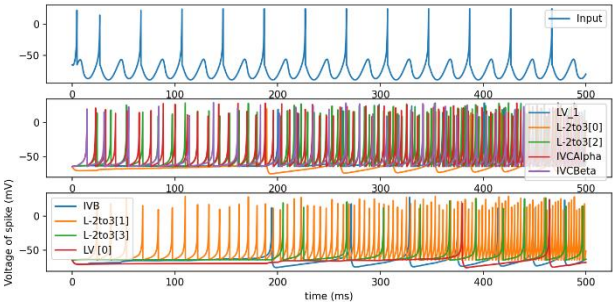
Amplitude: 10 at 50 Hz



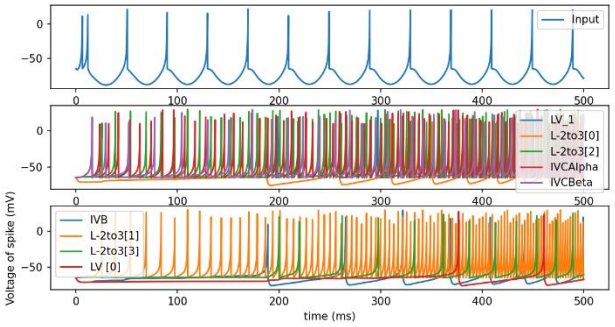
Amplitude: 10 at 25Hz



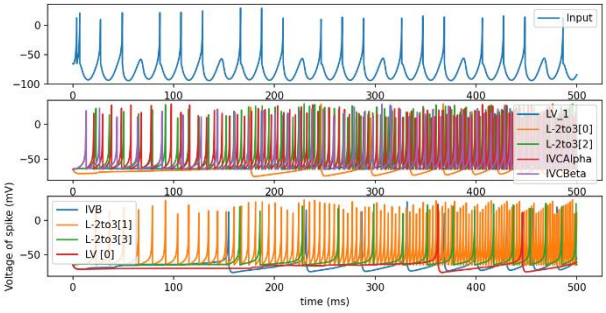
Amplitude: 15 at 50 Hz



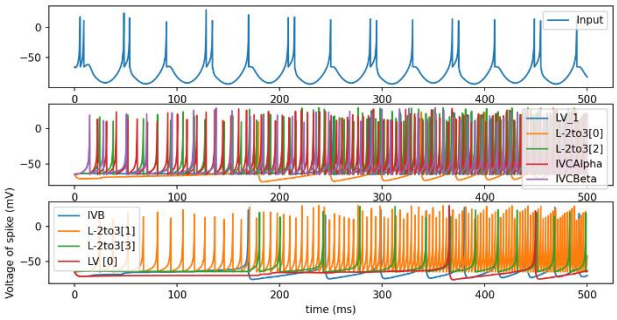
Amplitude: 15 at 25 Hz



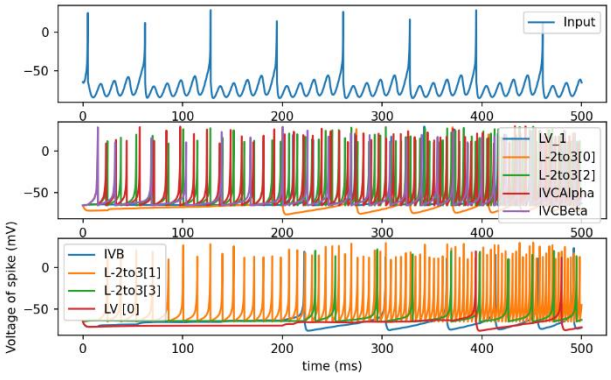
Amplitude: 20 at 50Hz



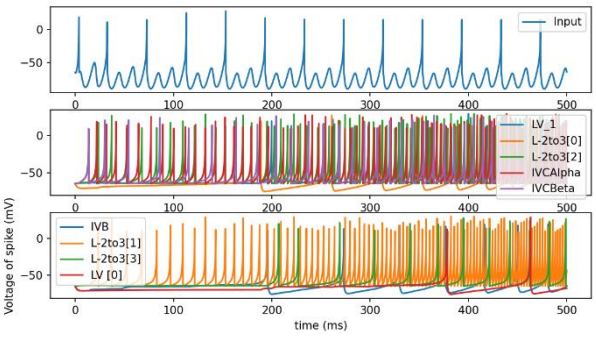
Amplitude: 20 at 25 Hz



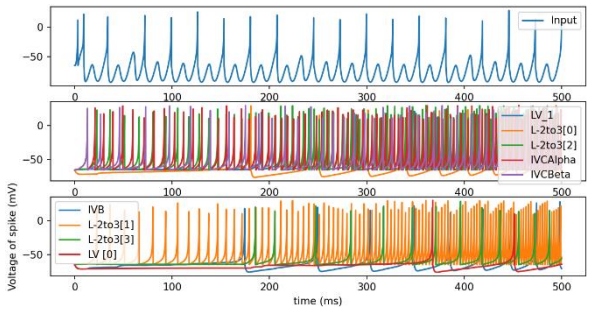
Amplitude: 10 at 75Hz



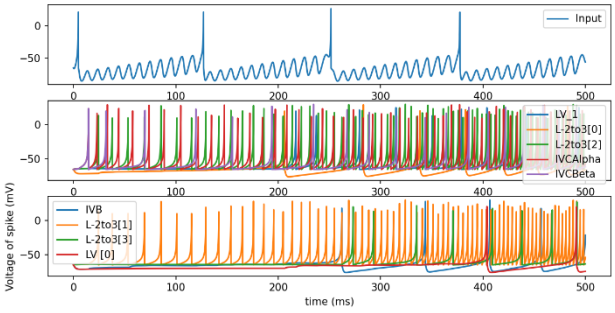
Amplitude: 15 at 75 Hz



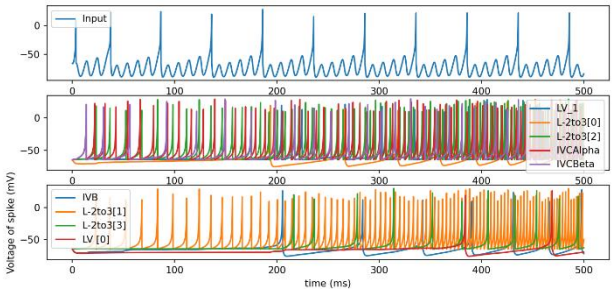
Amplitude: 20 at 75 Hz



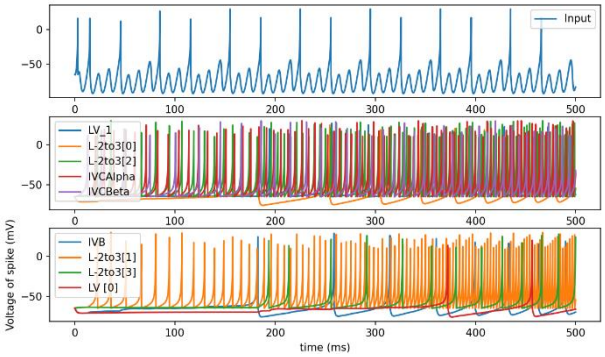
Amplitude: 10 at 100Hz



Amplitude: 15 at 100Hz

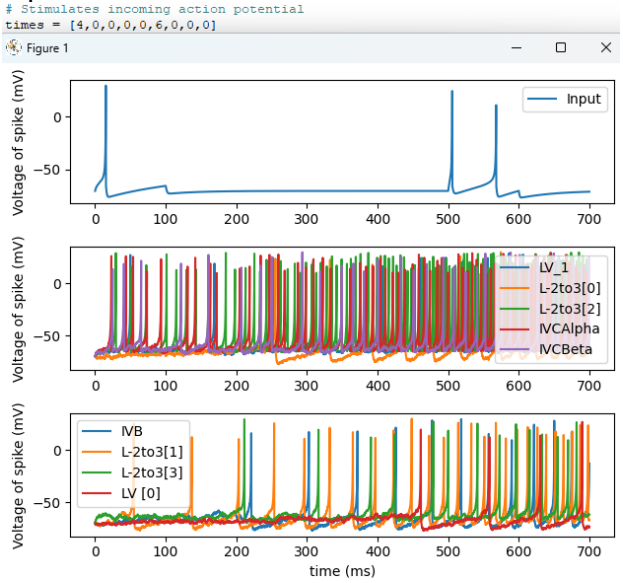


Amplitude: 20 at 100Hz

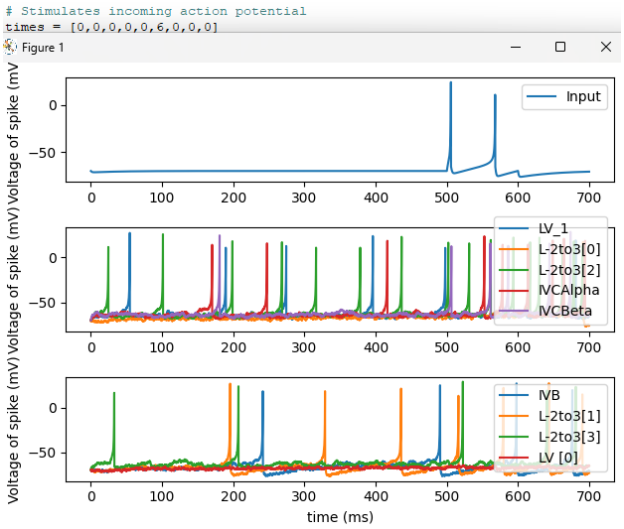


M. Phase 2 Individual event testing plots

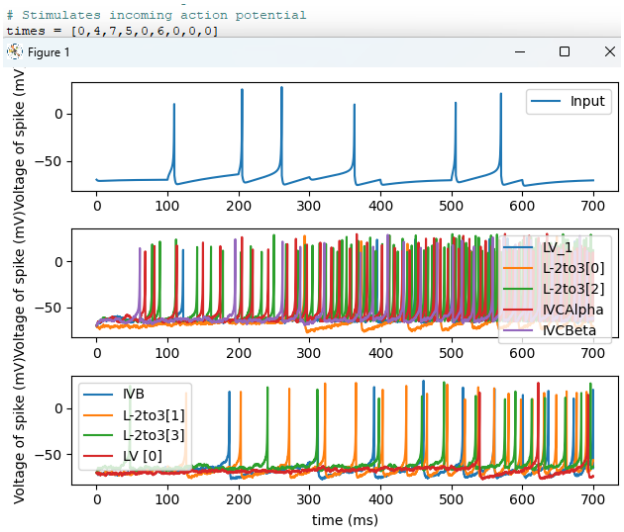
The models response starts correctly in this plot once an input is shown. Its response also strengthens as expected once there is a second input.

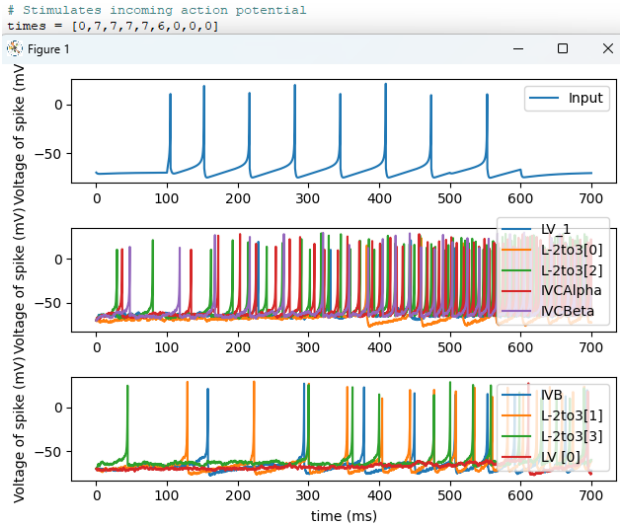


However, there is some random spiking before events occur. Random spiking before event comes from internal nodes having a standard deviation of 2 for their noise value.

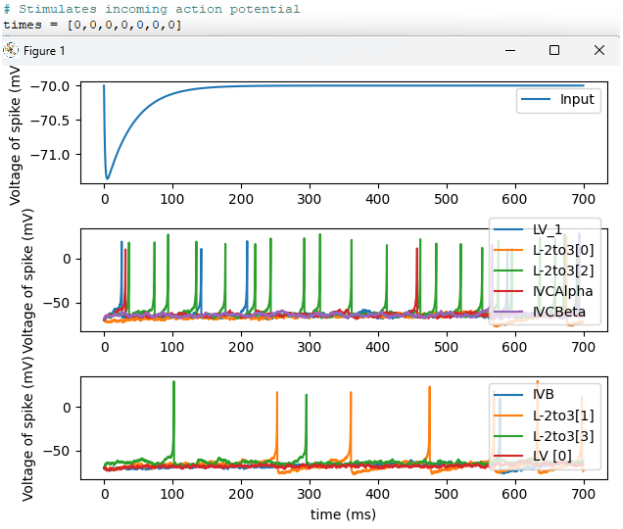


The plots below start close to the start of the simulation and it is still seen that random spiking behaiour is observed across multiple plots when there is high internal noise present.



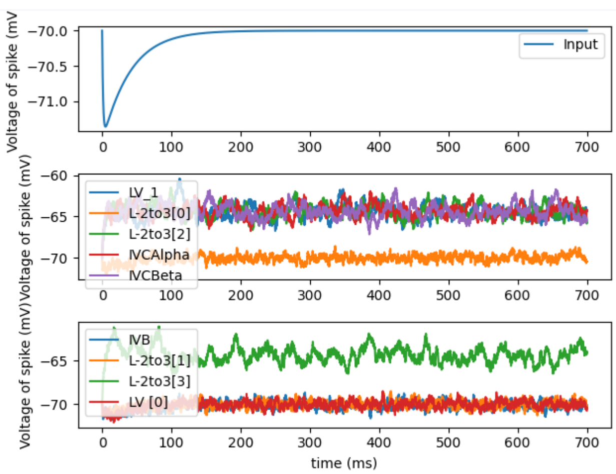


Trialling Standard deviation value of 2 in noise equation for neurons. Shown below. It shows spiking without input.



When turning down internal nodes noise to standard deviation of 1 it exhibited the correct behaviour. However, when using this value in other tests it removed the realistic neuron action potential shape and is why 2 is used in subsequent

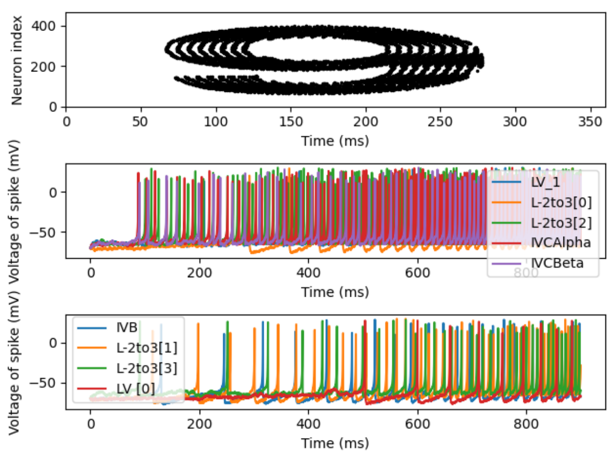
tests.

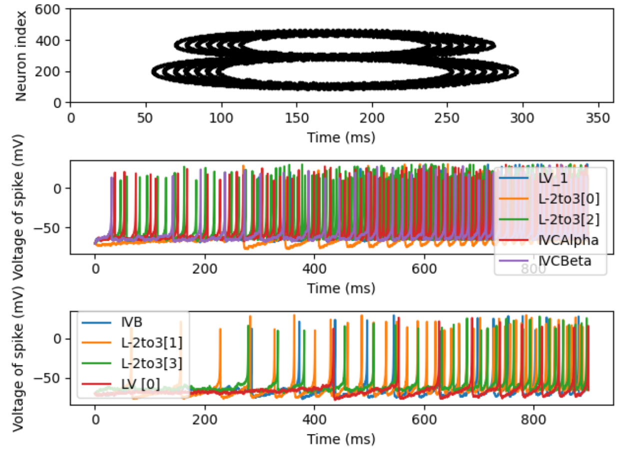


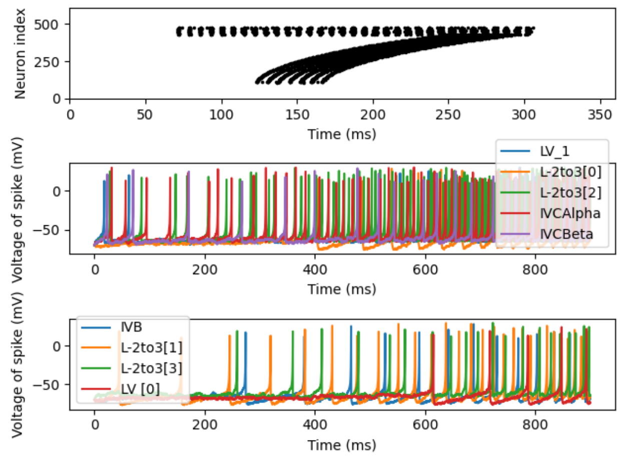
N. Phase 2 Event-based plots 0-9

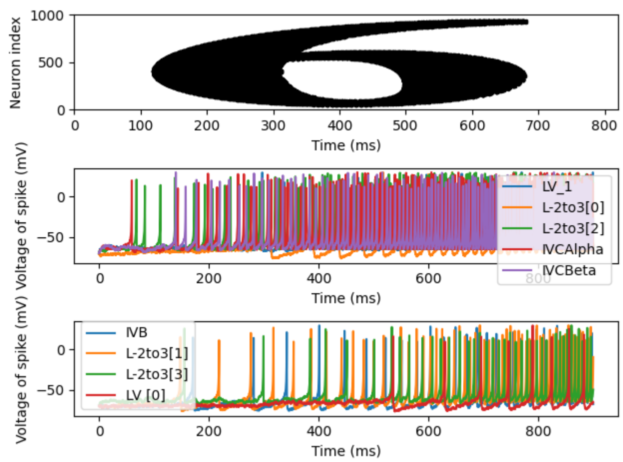
The numbers used to create these plots are referenced in the references section of the report. They showcase different spiking activity when presented with different images.

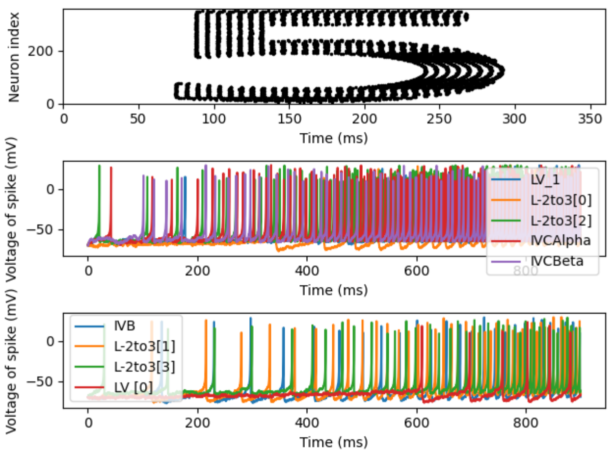
This Suggests the model has the ability to classify and process visual stimuli just like the primary visual cortex.

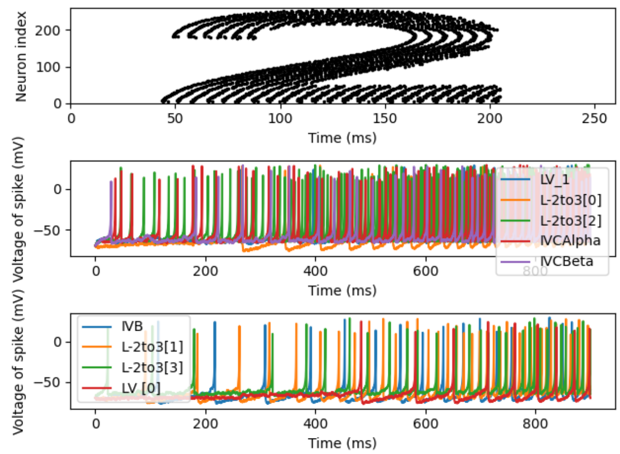


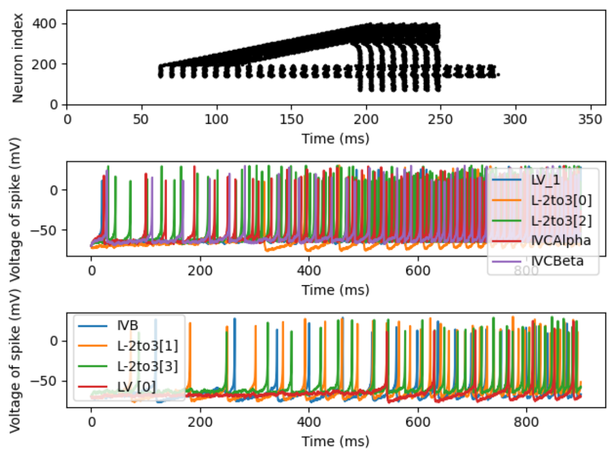


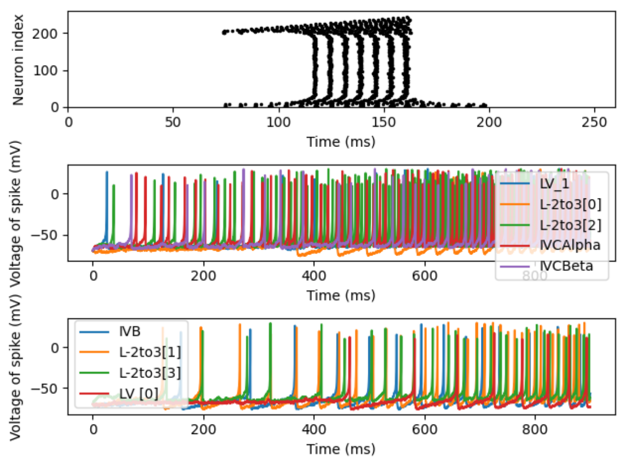


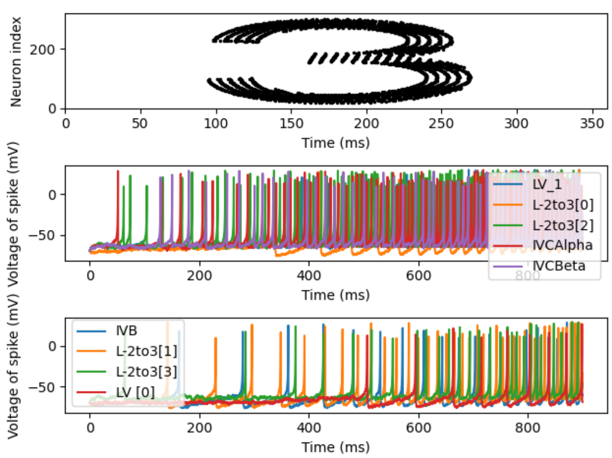


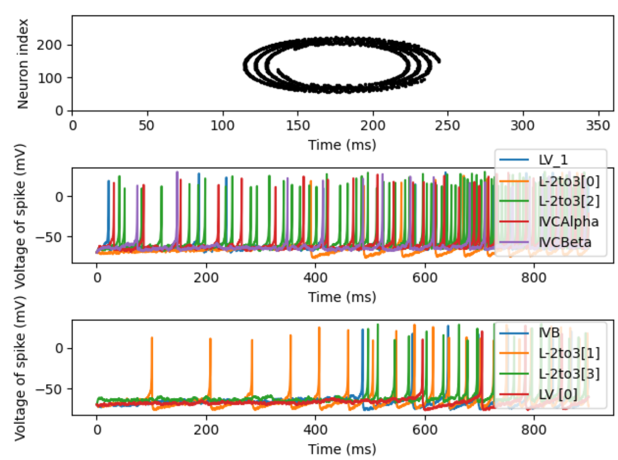

Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]


Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]


Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]

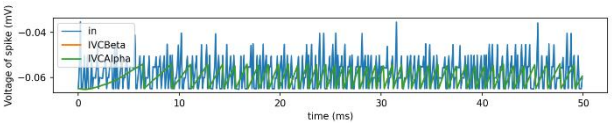

Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]


Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]

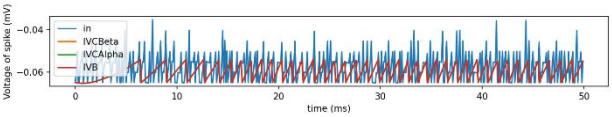

Neuron index
Time (ms)
Voltage of spike (mV)
LV_1
L-2to3[0]
L-2to3[2]
IVCAlpha
IVCBeta
IVB
L-2to3[1]
L-2to3[3]
LV [0]

O. Phase 1 LIF construction

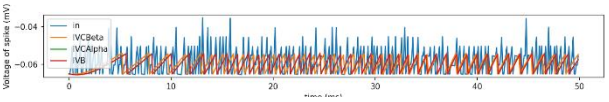
This section shows the construction of the LIF phase 1 model, before it was abandoned due to issues with STDP.



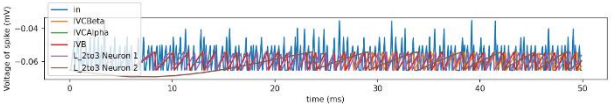
Implementing IVB

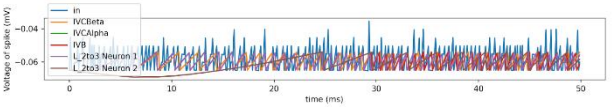


Implementing IVB and interconnection between IVB & IVCBeta, & IVBeta to IVCAlpha

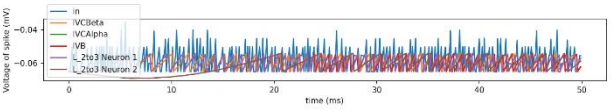


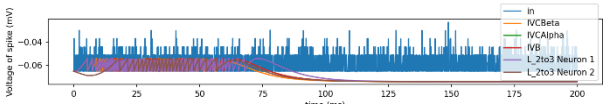
Implementing the abstract version of Layer || to |||



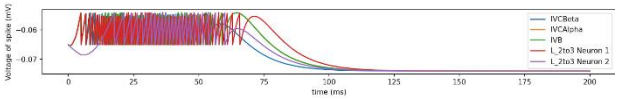


Final implementation of phase 1- taking outputs

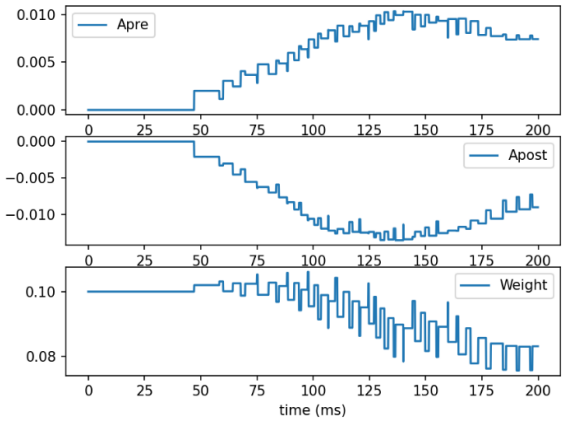




The model drives quite hard and then during this process the nodes in the model quit firing.



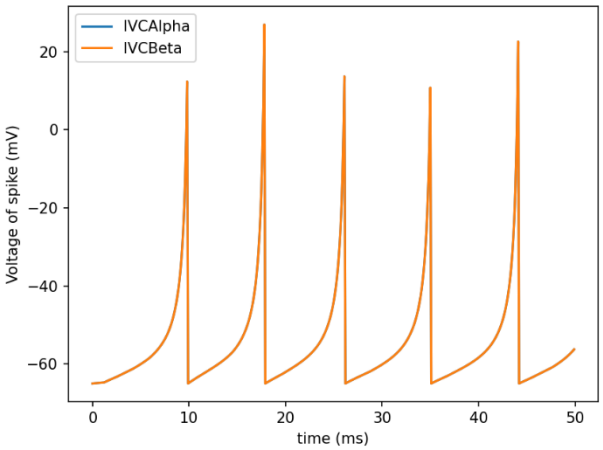
Looking at the connection between IVCBeta and the first node in the || & ||| layer. This random test to checks STDP is functioning but for some reason the weights in the model dropped to zero.



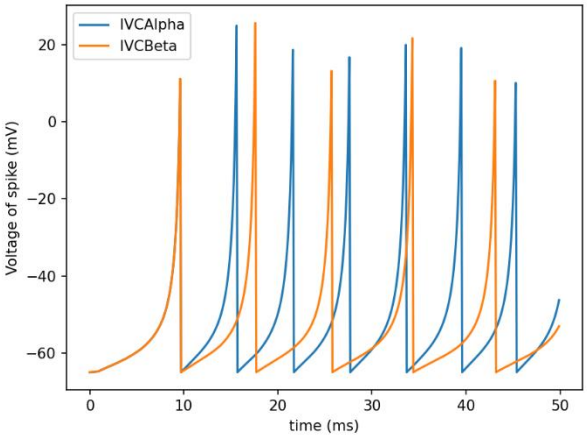
This testing of the structure with the LIF infrastructure is being abandoned at this point as the project needed to move forward and this was not near the main bulk of the project and therefore didn't need to be completed as it was mainly being done to delve into some of the concepts such as STDP in practice and to gain experience in how this process works.

P. Phase 1 Izhikevich construction

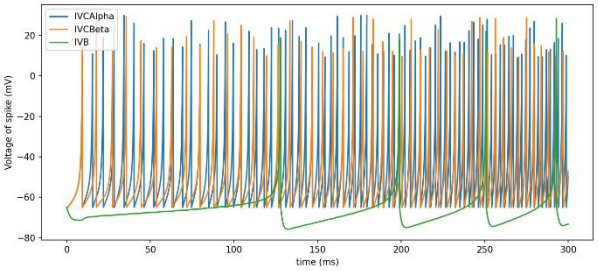
Instantiating IVCAlpha and IVCBeta



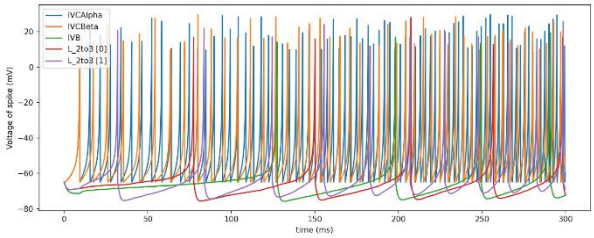
IvcBeta & IVCAlpha interconnection



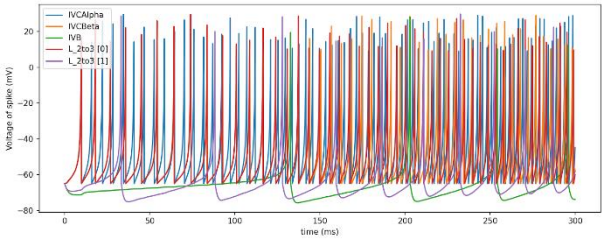
Connecting IVCBeta to IVB

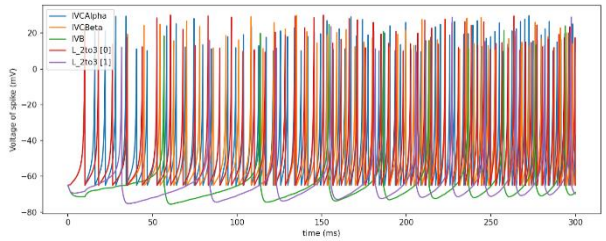


Including L_2to3

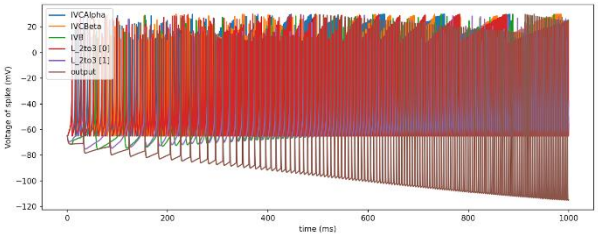


Changed L_2to3[0] to be inhibitory.

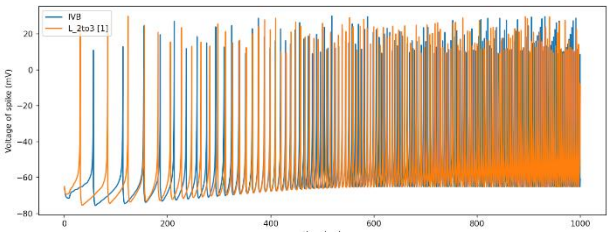




The final test of the architecture did assess having the output being one node, but this does not work well and shows that the output would then become unstable.



This singular output is removed, and the remaining behaviour is more expected of mechanisms like cortical stabilising.

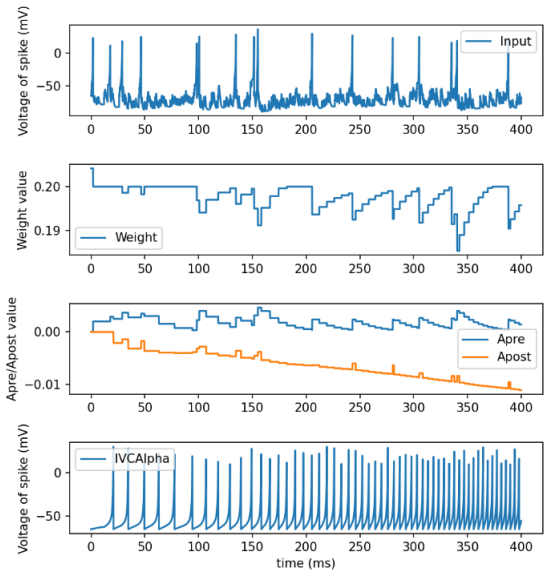


Q. Random weight testing

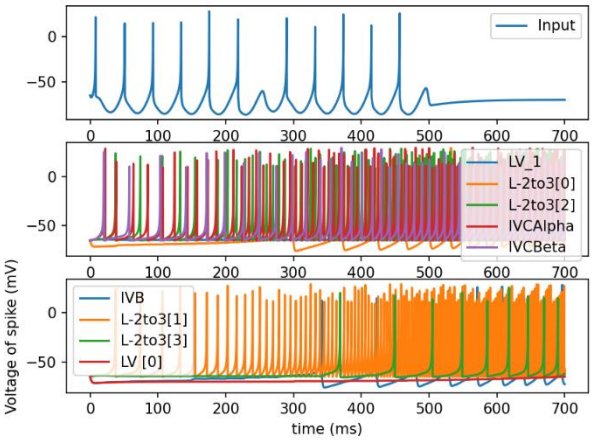
During the verification of STDP in the phase 1 stage all the weights in the network were instantiated to 0.1 to allow the changes to be seen in the weights. However, as mentioned in that section it causes different behaviour than to plots that were previously produced. Therefore, the next stages of testing are to change the values of the weights at the start of the simulation to be random instead of a constant value.

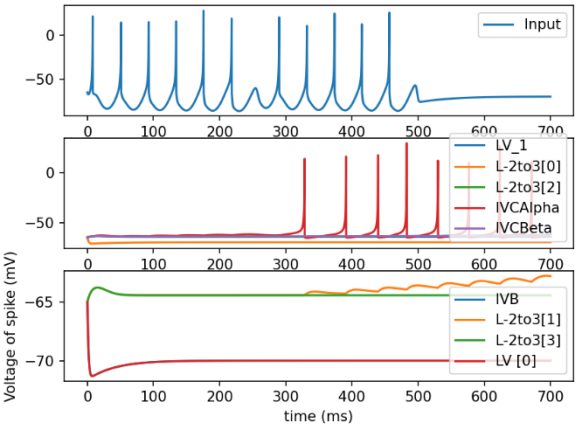
This is because it showed unexpected behaviour. It is theorised that using a random instantiation of weight values would allow a wider range of behaviours to be seen from the model. The way to implement this in python is to use the “rand()” function which returns a uniformly distributed variable between 0 and 1. Implementing the random function in the weights of the network causes different responses from the variables in the network and in the response of the neuron.

Shown below.

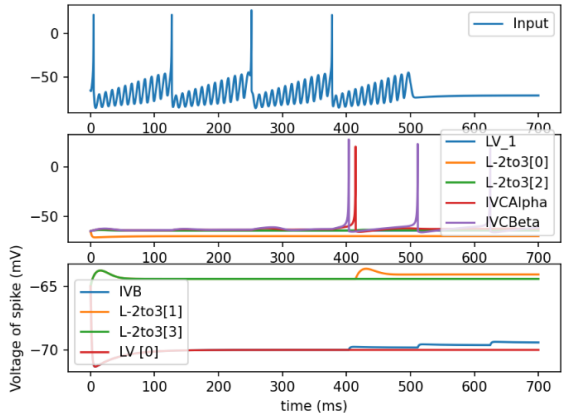


When testing this on the phase 2 model with Izhikevich it caused the model to react differently than when weights were constant. The simulations with a 25Hz sine wave with 10 amplitude produced the two plots below. Showing unstable behaviour.



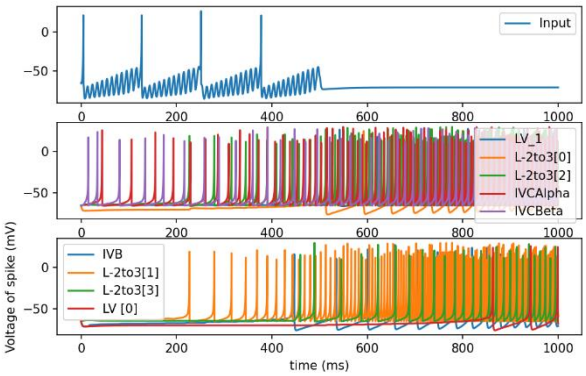


Even with 100Hz and amplitude 10, weak spiking behaviour is seen with only one or two nodes firing.



This is because the rand() function is returning weight values that are so small that they cause only small modulation in the current. Scaling this rand function by a constant value showed better behaviour.

In the plot below it is shown that there is no benefit to making the weights random in this model as behaviour is like setting weights to unity.



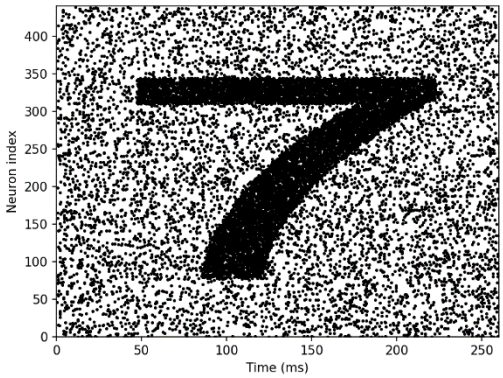
R. Event-based image function testing

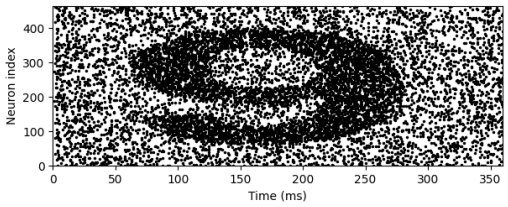
Documentation of brian2 function used I event based vision [48] originally uses the values below. This is shown in the documentation of the function.

```
A = 1.5
tau = 2*ms
eqs = '''
dv/dt = (A*ta(t, i)-v)/tau+0.8*xi*tau**-0.5 : 1
'''
```

[48]

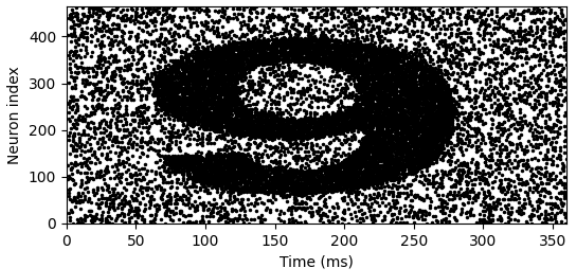
This produced different results initially seen in the images below.



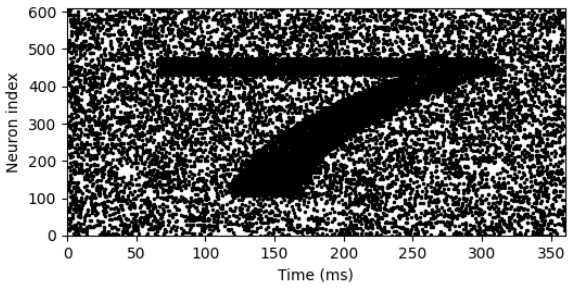


Keeping the standard deviation value, the same, changed the multiplier on the array storing the image values to 3.

```
#### Refactored but taken from Brian2############
# Basic LIF implementation where the reversal potential
# is replaced with a timed array of the spike events
eqs = '''
dv/dt = (3*timing_im(t, i)- v)/tau_M + 0.8*sqrt(2/tau_M)*xi: 1

'''
```

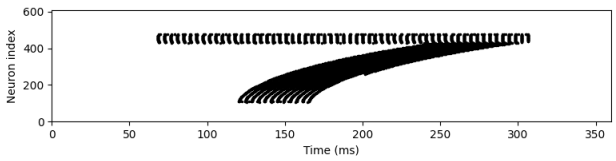



```
eqs = '''
dv/dt = (3*timing_im(t, i)- v)/tau_M + sqrt(2/tau_M)*xi: 1

'''
```



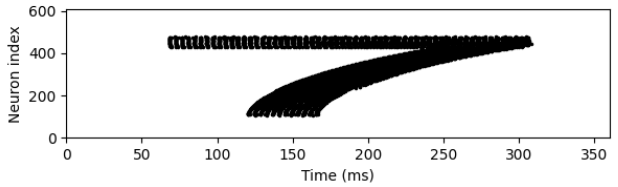
Changing noise to zero showed lower resolution of the object in the image.

```
eqs = '''
dv/dt = (3*timing_im(t, i)- v)/tau_M + 0*sqrt(2/tau_M)*xi: 1

'''
```



Trailed up to 0.02 standard noise deviation gave better image resolution.

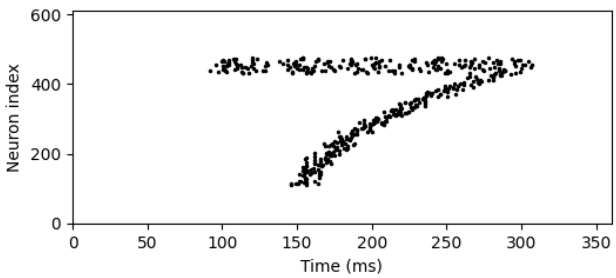
```
eqs = '''
dv/dt = (3*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
```



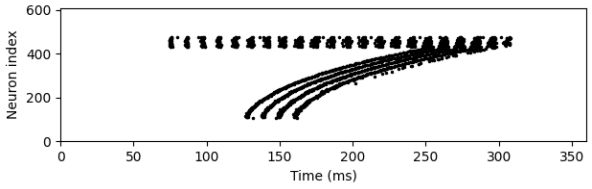
The previous values gave the best image resolution. However, a complexity and size argument must be mentioned. The dots all represent a neuron index, meaning there are over four hundred neurons representing this image. The hardware running the simulation wasn't a supercomputer and it needed to be used in a project where scale and complexity needed to be low. Therefore, a way to represent the image at a lower resolution without losing image shape was done below.

The value on the image array had to be greater than one as testing showed below the image resolution to be poor.

```
eqs = '''
dv/dt = (1*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
```

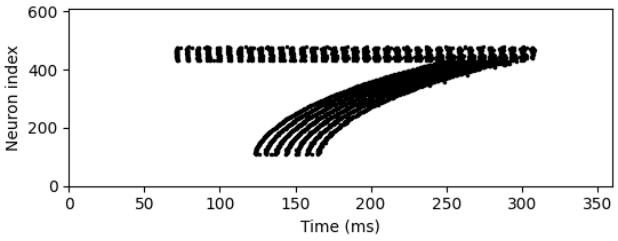



```
eqs = '''
dv/dt = (1.5*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
```

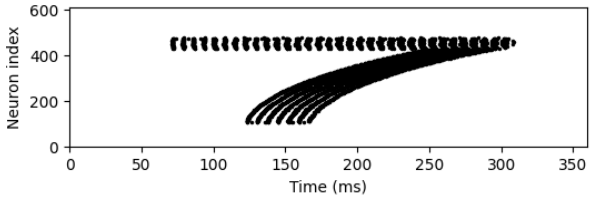


The final values trialled were 2 and 1.95. They produced plots that weren't dissimilar. Their image quality was adequate and the number of neurons representing the image reduced. Since the trialling of 1.95 showed like 2 it was used in the final project code along side the standard deviation value of 0.02.

```
eqs = '''
dv/dt = (2*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
```

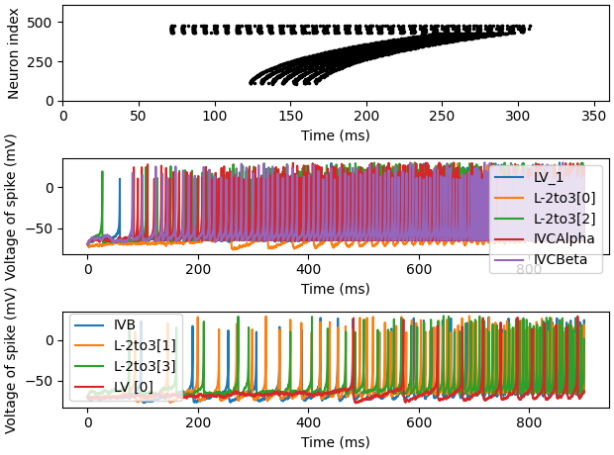



```
eqs = '''
dv/dt = (1.95*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
```

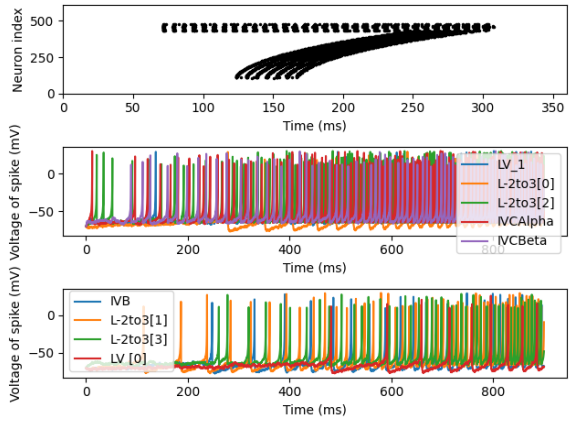


When attaching function to model, outlined in testing 6.8, the value of 0.2 was chosen. The subsequent plots showcase why this value was chosen. As it gives a weaker response than previously.

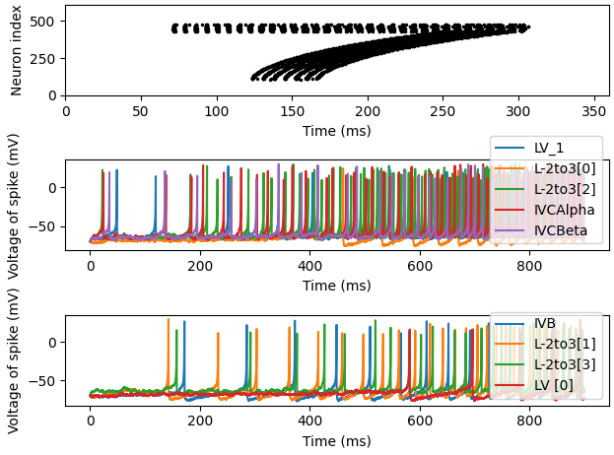
```
# Connecting the image neurons to the main model
image_input =  Synapses(im_neuron,input_neuron,model='w:1',on_pre ="v
image_input.connect()
```



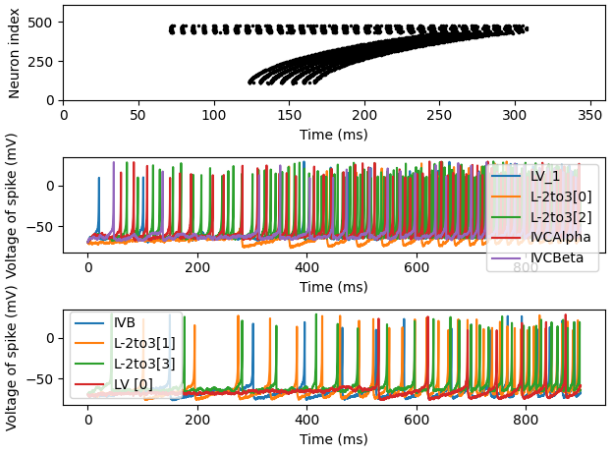

```
on_pre ="v_post +=0.5",
```




```
on_pre ="v_post +=0.3"
```

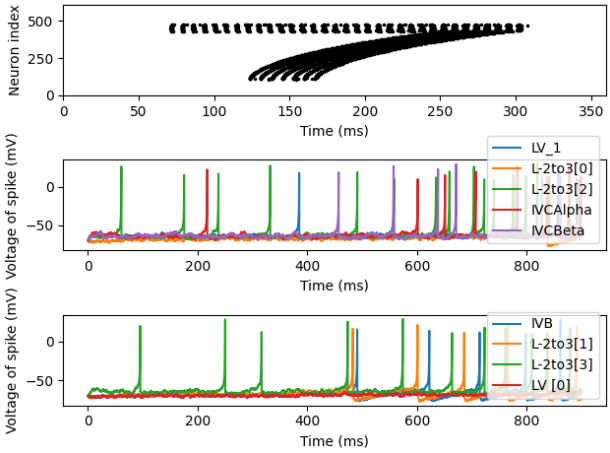



```
on_pre ="v_post +=0.2"
```



The “on_pre” syntax is chosen as logically want to scale response onto the next group of neurons. The other syntax elicits responses that are very weak. Therefore, it isn’t used.

```
on_post ="v_post +=0.2"
```



S. *Poisson, Individual & sinusoidal project code*

```
from brian2 import*
import matplotlib as plt

# Benjamin McCann
# Some code isnt put into functions as it changes the simulation output
# Project uses Brian2 package sourced at [69]

#Converted David Halliday Matlab Function in python [50]
# This function compares individual spike times in the array of input spikes
#to spikes in the output array
def cross_correlation_func(InputSpikes,OutputSpikes,binSize):
    #Largest time difference gives the range of which the
    # cross correlation function will look at
   # number of sampling intervals
    size_array =binSize
    lag_array=[]
    # Assign lag array same number of indexs as the number of intervals
    # e.g. if binsize is 50 it makes an array of length 101 so the graph can
    # be centred at 1 with +/- 50 either side
    for j in range(2*size_array  + 1):
        lag_array.append(0)
    time_lag = 0
    print(len(lag_array))
    neg_tot =50
    lag_tot =101
    neg_offset = neg_tot +0.5
    in_ind =0
    # go through the entire list and then compare each index in input spikes to
every value in output spikes
    for i in range(len(InputSpikes)):
        # set back to zero every time loop runs
        out_ind =0
        # Sets end of interval with the Inputspik[i] being -neg_offset from the
stop
        # Looks +/- binsize for every index, centred at 1
        #    lag_start < --------- InputSpikes[i] -------> lag_stop
        lag_stop = int(InputSpikes[i]) + neg_offset
        print("current array value",int(InputSpikes[i]))
        lag_start = lag_stop - lag_tot
        print("Lag start value",lag_start)
        print("Lag stop value",lag_stop)
        outputindex =[]
        # Python version of Matlabs "find()" function
        # Will find the index's in the output array
        #that meets the conditions needed.
        # Since the find function finds ALL indexs
        # that meet these requirements in pyhton
        #this is translated to appending an array after going through the array
entirely

        # If an index is within this range of -binsize +0.5 -----X----binsize +0.5
        # add to array
        for j in OutputSpikes:
            if(int(j) >lag_start):
                if(int(j) <=lag_stop):
```

```
                    outputindex.append(out_ind)

            out_ind +=1
        print("output index",outputindex)

        #go through all values that identify an index in the output
        #array that is a correlated value to the currently viewed input spike time
        # then increment the lag array in the index that correlates too the time
        # that the spike pair correlate
        for i in range(len(outputindex)):
                        print("value in output
array",int(OutputSpikes[outputindex[i]]))
                        #compare end of range to value in output array
                        indx = math.floor(lag_stop -
int(OutputSpikes[outputindex[i]])) + 1
                        print(" Hello" ,indx)
                        # As long as the index is within the array
                        if( indx < len(lag_array)):
                            # Increment the number of synchronous spike pairs
                            # at this time index
                            lag_array[indx] +=1
    plot(lag_array)
    title("Cross-Correlation function")
    xlabel("Lag")
    ylabel("Mean value")
    show()
# Above is converted David Halliday Matlab Function in python [50]


def
plot_output(statm,IVCBETA,IVCALPHA,IVB,L2to3_0,L2to3_1,L2to3_2,L2to3_3,LV_0,LV_1):

    #Plotting input node sim to [46]
    subplot(311)
    plot(statm.t/ms,statm.v[0],label ='Input')
    ylabel('Voltage of spike (mV)')
    legend()
    #Plotting outputs of hidden nodes
    subplot(312)
    plot(LV_1.t/ms,LV_1.v[0],label ='LV_1')
    plot(L2to3_0.t/ms,L2to3_0.v[0],label ='L-2to3[0]')
    plot(L2to3_2.t/ms,L2to3_2.v[0],label ='L-2to3[2]')
    plot(IVCALPHA.t/ms,IVCALPHA.v[0],label ='IVCAlpha')
    plot(IVCBETA.t/ms,IVCBETA.v[0],label ='IVCBeta')
    legend()
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    # Plotting outputs of system
    subplot(313)
    plot(IVB.t/ms,IVB.v[0],label ='IVB')
    plot(L2to3_1.t/ms,L2to3_1.v[0],label ='L-2to3[1]')
    plot(L2to3_3.t/ms,L2to3_3.v[0],label ='L-2to3[3]')
    plot(LV_0.t/ms,LV_0.v[0],label ='LV [0]')
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    legend()
    tight_layout()
    show()
```

```
def plot_synchrony_output(spikmon):
    plot(spikmon.t/ms,spikmon.i,'.')
    # Draw lines at specifc points
    # to help show synchrony
    axvline(x = 50, color = 'r')
    axvline(x = 100, color = 'r')
    axvline(x = 150, color = 'r')
    axvline(x = 200, color = 'r')
    axvline(x = 250, color = 'r')
    axvline(x = 300, color = 'r')
    xlabel("Time (ms)")
    ylabel("Index in PVC model")
    show()

def save_results(statm_1,statm_3,statm_5,statm_7,statm_8):
    ##### Save results!!!!!!!
    # Open or creates a file with the name in the first txt field
    f= open("Results.txt","a+")
    # Write specifics of test
    f.write("\n Parameters e.g.HZ or Amplitude: ")
    f.write("\nInput spike times\n")
    # write the times at which spikes occured in
    #spikemonitor statm_1
    f.write(str(statm_1.t))
    f.write("\nOutput spike times\n ")
    f.write("\n IVB spike time\n")
    f.write(str(statm_3.t))
    f.write("\n L-2to3[1]")
    f.write(str(statm_5.t))
    f.write("\n L-2to3[3]")
    f.write(str(statm_7.t))
    f.write("\n LV [0]")
    f.write(str(statm_8.t))
    f.close()

# Neuron membrane relaxation time constant
tau_M = 10*ms
# Time constants in exponential terms for STDP
#Abbot Song & Miller [45]
tau_pre= 20*ms
tau_post =20*ms
# Node resting potential
v_r = -70
run_time = 700*ms
# Abbot Song & Miller [45] #
gmax = 0.2
dApre = 0.02
dApost = -dApre * tau_pre / tau_post * 1.05
dApost *= gmax
dApre *= gmax
#[45]
stdp_eqn = '''
dApre/dt = -Apre/tau_pre : 1 (event-driven)
dApost/dt = -Apost/tau_post : 1 (event-driven)
w : 1
'''

######
# E.Izhikevich 2006[46]
```

```
pre_eqn ='''
I += w
Apre +=  dApre
w = clip(w + Apost, 0, gmax)'''
#############

# Equations used in the Middle/ Output nodes
# Used when using Poisson Input [5][23]
izhikevich_eqns = '''
dv/dt = (0.04*v*v + 5*v + 140 -u + I )/ms + 2*sqrt(2/tau_M)*xi : 1
du/dt = a*(b*v - u)/ms : 1
a : 1
b : 1
c : 1
d : 1
I : 1
'''

### Use these equations when using sinusoidal & individual times input only
# Not needed for Poisson input as the syntax works differently
# Sine and individual events stored into an array, this array function of
#time is represented here as I_experiment(t) - then stored as the driving
# current of the node I.
izhikevich_eqns_sine_indv = '''
dv/dt = (0.04*v*v + 5*v + 140 -u + I)/ms +  2*sqrt(2/tau_M)*xi : 1
du/dt = a*(b*v - u)/ms : 1
a : 1
b : 1
c : 1
d : 1
I = I_experiment(t): 1
'''

# Use the one below when using Sinusoidal & Individual inputs
input_neuron = NeuronGroup(1,izhikevich_eqns_sine_indv,threshold = 'v >=30',reset
='v = c; u +=d;',method='euler')

# Use the one below when using Poisson inputs
#input_neuron = NeuronGroup(1,izhikevich_eqns,threshold = 'v >=30',reset ='v = c; u
+=d;',method='euler')


#-----------------------------------------------------------------------
#POISSON INPUT
# Use below for Poisson Input (uncomment line below )
pos = PoissonInput(input_neuron,'v',100,10*Hz,weight=1)
#-----------------------------------------------------------------------

#\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
# SINUSOIDAL INPUT
# There are other ways of doing this sinusoidal input, but this syntax in brian2
#[69]
# was used  because it was needed to input indvidual events as well
#this timed array syntax was needed

# Only run for time 600ms in increments dictated by the system clock
# UNCOMMENT BELOW FOR SINE INPUT
record_t = arange(int(run_time/defaultclock.dt))*defaultclock.dt
# Record the sine wave of Amplitude X with freq Y at times T with spacing
```

```
# determined by the default clock (system clock)
# UNCOMMENT BELOW FOR SINE INPUT
I_experiment = TimedArray(15*sin(2*pi*50*Hz*record_t), dt=defaultclock.dt)
#\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\


# INDIVIDUAL TIMED ARRAY (individual responses)
# Put events into array below - events of different current values
# Stimulates incoming action potential
#UNCOMMENT FOR INDV EVENTS
#times = [0,6,0,7,0,0,0]
# Events spaced apart by dt value - always starts at zero
#UNCOMMENT FOR INDV EVENTS
#I_experiment = TimedArray(times,dt = 100*ms)


# Model neuron group
PVC =NeuronGroup(9,izhikevich_eqns,threshold = 'v >=30',reset ='v = c; u
+=d;',method='euler')
# All synapses instantiated with unity weight.Trialed random weights (seen in
appendix) but caused
# weights to drop to zero and model to stop working.

# Input to IVCBeta
in_PVC_0 = Synapses(input_neuron,PVC[0],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_0.connect()
in_PVC_0.w =1

# Input to IVCAlpha -correct
in_PVC_1 = Synapses(input_neuron,PVC[1],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_1.connect()
in_PVC_1.w =1


# Input to || & ||| node 1 - correct
in_PVC_4 = Synapses(input_neuron,PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_4.connect()
in_PVC_4.w =1


# IVCBeta to IVB - correct
PVC_0_PVC_2 = Synapses(PVC[0],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_2 .connect()
PVC_0_PVC_2 .w =1


# IVCBeta to IVCAlpha
PVC_0_PVC_1 = Synapses(PVC[0],PVC[1],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_1 .connect()
```

```
PVC_0_PVC_1 .w =1


# IVCBeta to L || & ||| node 1 - correct
PVC_0_PVC_4= Synapses(PVC[0],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_4 .connect()
PVC_0_PVC_4.w =1


# IVCAlpha to node 2  L || & ||| - correct
PVC_1_PVC_5= Synapses(PVC[1],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_1_PVC_5 .connect()
PVC_1_PVC_5.w =1


# IVB to node 3 in L || & |||- correct
PVC_2_PVC_6 = Synapses(PVC[2],PVC[6],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_6 .connect()
PVC_2_PVC_6.w =1


# IVB to Node 1
PVC_2_PVC_5 = Synapses(PVC[2],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_5 .connect()
PVC_2_PVC_5.w =1


# IVB to IVCBeta
PVC_2_PVC_0 = Synapses(PVC[2],PVC[0],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_0 .connect()
PVC_2_PVC_0.w =1

############################
# Layer || & ||| connections
############################

# Layer || & ||| node 0 - PVC_3

# L || & ||| Node 0 to Node 1 - correct
PVC_3_PVC_4 =Synapses(PVC[3],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                     w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_4 .connect()
PVC_3_PVC_4.w =1


################
# L || & ||| node 0 to Lv node 1- correxct
PVC_3_PVC_8=Synapses(PVC[3],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
```

```
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_8 .connect()
PVC_3_PVC_8.w =1


# L || & ||| node 0 to Lv node 0 - correct
PVC_3_PVC_7=Synapses(PVC[3],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_7 .connect()
PVC_3_PVC_7.w =1

# node 1 PVC_4
# Node 1 lAYER || & ||| to IVB - correct
PVC4_PVC2 =Synapses(PVC[4],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC4_PVC2 .connect()
PVC4_PVC2.w =1

#
# Node 3 lAYER || & ||| to IVB -corrrect
PVC6_PVC2 =Synapses(PVC[6],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC6_PVC2 .connect()
PVC6_PVC2.w =1

#node 2 - PVC_5
# 2 to 1 - correct
PVC_5_PVC_4 =Synapses(PVC[5],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_4 .connect()
PVC_5_PVC_4.w =1
# Layer || to ||| node 2 to node 0 - correct
PVC_5_PVC_3 =Synapses(PVC[5],PVC[3],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_3 .connect()
PVC_5_PVC_3.w =1

# LV connections - correct
PVC_7_PVC_8  = Synapses(PVC[7],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_7_PVC_8 .connect()
PVC_7_PVC_8.w =1
#
PVC_8_PVC_7  = Synapses(PVC[8],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_8_PVC_7 .connect()
PVC_8_PVC_7.w =1

#
PVC_8_PVC_4  = Synapses(PVC[8],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
            on_post='''Apost += dApost
```

```
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_8_PVC_4 .connect()
PVC_8_PVC_4.w =1


#Setting Excitatory Regular Firing Patterns for Neurons  [5][23]
#-----------------------------------------------------------
input_neuron.a = 0.02
input_neuron.b = 0.2
input_neuron.c = -65
input_neuron.d = 8
input_neuron.u =0.2*-65
input_neuron.v = v_r
#------node 0 L || & |||
PVC[3].a = 0.02
PVC[3].b = 0.2
PVC[3].c = -65
PVC[3].d = 8
PVC[3].u =0.2*-65
PVC[3].v =v_r
#-------------------
PVC[4].a = 0.02
PVC[4].b = 0.2
PVC[4].c = -65
PVC[4].d = 8
PVC[4].u =0.2*-65
PVC[4].v =v_r
#-------------------
PVC[2].a = 0.02
PVC[2].b = 0.2
PVC[2].c = -65
PVC[2].d = 8
PVC[2].u =0.2*-65
PVC[2].v = v_r
#--------------------
PVC[7].a = 0.02
PVC[7].b = 0.2
PVC[7].c = -65
PVC[7].d = 8
PVC[7].u =0.2*-65
PVC[7].v = v_r

#Setting Inhibitatory Thalamo-Cortical Firing Patterns for Neurons [23]
#-------------------------------------------------------------------
PVC[1].a = 0.02
PVC[1].b = 0.25
PVC[1].c = -65
PVC[1].d = 0.05
PVC[1].u =0.25*-65
PVC[1].v = v_r
#-------------------
PVC[0].a = 0.02
PVC[0].b = 0.25
PVC[0].c = -65
PVC[0].d = 0.05
PVC[0].u =0.25*-65
PVC[0].v = v_r
#-----------------
PVC[5].a = 0.02
```

```
PVC[5].b = 0.25
PVC[5].c = -65
PVC[5].d = 0.05
PVC[5].u =0.25*-65
PVC[5].v =v_r
#------------------
PVC[6].a = 0.02
PVC[6].b = 0.25
PVC[6].c = -65
PVC[6].d = 0.05
PVC[6].u =0.25*-65
PVC[6].v =v_r
#------------------
PVC[8].a = 0.02
PVC[8].b = 0.25
PVC[8].c = -65
PVC[8].d = 0.05
PVC[8].u =0.25*-65
PVC[8].v =v_r
#------------------

#Setting up variables to minitor the states of the different outputs and hidden
nodes
statm = StateMonitor(input_neuron,'v', record=0)
statm_1 = SpikeMonitor(input_neuron)
IVCBETA = StateMonitor(PVC[0],'v', record=0)
IVCALPHA = StateMonitor(PVC[1],'v', record=0)
IVB = StateMonitor(PVC[2],'v', record=0)
IVB_SM = SpikeMonitor(PVC[2])
# L|| to ||| --------------------------------
L2to3_0 = StateMonitor(PVC[3],'v', record=0)
L2to3_1 = StateMonitor(PVC[4],'v', record=0)
L2to3_1_SM = SpikeMonitor(PVC[4])
L2to3_2 = StateMonitor(PVC[5],'v', record=0)
L2to3_3 = StateMonitor(PVC[6],'v', record=0)
L2to3_3_SM = SpikeMonitor(PVC[6])
# LV ----------------------------------------
LV_0 = StateMonitor(PVC[7],'v', record=0)
LV_0_SM = SpikeMonitor(PVC[7])
LV_1 = StateMonitor(PVC[8],'v', record=0)

spikmon =SpikeMonitor(PVC)


#Run Experiment
run(run_time,report ='text')

# Plotting Cross-correlation
#cross_correlation_func(statm_1.t/ms,L2to3_1_SM.t/ms,50)

# Save results from test
#save_results(statm_1,IVB_SM,L2to3_1_SM,L2to3_3_SM,LV_0_SM)

# Plot a synchrony plot of spike times of output nodes
#plot_synchrony_output(spikmon)

#Plot input to output response including responses of hidden nodes
plot_output(statm,IVCBETA,IVCALPHA,IVB,L2to3_0,L2to3_1,L2to3_2,L2to3_3,LV_0,LV_1)
```

T. Vision project code

```
from brian2 import*
import matplotlib as plt
from matplotlib.image import imread

# Benjamin McCann
# : Neuromorphic Computing: Modelling the behaviour of the Primary visual cortex
with Spiking Neural Networks
# Architecture not in functions as this changes simulation output

def save_results(statm_0,IVB_SM,L2to3_1_SM,L2to3_3_SM,LV_0_SM):
    ####### Uncomment to save results
    f= open("Number9runs.txt","a+")
    f.write("\n Input spikes \n")
    for i in statm_0.t:
        f.write(str(i))
    f.write("\n IVB spike time\n")
    for i in statm_3.t:
        f.write(str(i))
    f.write("\n L-2to3[1]\n")
    for i in statm_5.t:
        f.write(str(i))
    f.write("\n L-2to3[3]\n")
    for i in statm_7.t:
        f.write(str(i))
    f.write("\n LV [0]\n")
    for i in statm_8.t:
        f.write(str(i))
    f.write("\n")
    f.close()

# plots the outputs of the model
def
plot_output(spikmon,time,number_neurons,IVCBETA,IVCALPHA,IVB,L2to3_0,L2to3_1,L2to3_
2,L2to3_3,LV_0,LV_1):

    subplot(311)# part also documented[48]
    plot(spikmon.t/ms, spikmon.i, '.k', ms=3)
    xlim(0, time)
    ylim(0, number_neurons)
    xlabel('Time (ms)')
    ylabel('Neuron index')
    #Plotting outputs of hidden nodes
    subplot(312)
    plot(LV_1.t/ms,LV_1.v[0],label ='LV_1')
    plot(L2to3_0.t/ms,L2to3_0.v[0],label ='L-2to3[0]')
    plot(L2to3_2.t/ms,L2to3_2.v[0],label ='L-2to3[2]')
    plot(IVCALPHA.t/ms,IVCALPHA.v[0],label ='IVCAlpha')
    plot(IVCBETA.t/ms,IVCBETA.v[0],label ='IVCBeta')
    legend()
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    # Plotting outputs of system
    subplot(313)
    plot(IVB.t/ms,IVB.v[0],label ='IVB')
    plot(L2to3_1.t/ms,L2to3_1.v[0],label ='L-2to3[1]')
    plot(L2to3_3.t/ms,L2to3_3.v[0],label ='L-2to3[3]')
```

```
    plot(LV_0.t/ms,LV_0.v[0],label ='LV [0]')
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    legend()
    tight_layout()
    show()

tau_M = 10*ms
v_r = -70
sigma = 2
# Change this to change simulation length
time_ran = 900*ms

#STDP Source Abbor ,Song, Miller [45]
tau_pre= 20*ms
tau_post =20*ms
gmax = 0.2
dApre = 0.01
dApost = -dApre * tau_pre / tau_post * 1.05
dApost *= gmax
dApre *= gmax

stdp_eqn = '''
dApre/dt = -Apre/tau_pre : 1 (event-driven)
dApost/dt = -Apost/tau_post : 1 (event-driven)
w : 1
'''
###
### E.Izhikevich [46]
pre_eqn ='''
I += w
Apre += dApre
w = clip(w + Apost, 0, gmax)'''
#################

# E.Izhikevich [5][23]
izhikevich_eqns = '''
dv/dt = (0.04*v*v + 5*v + 140 -u + I)/ms + sigma*sqrt(2/tau_M)*xi: 1
du/dt = a*(b*v - u)/ms : 1
a : 1
b : 1
c : 1
d : 1
I : 1
'''

# IF IT SAYS IndexError: too many indices for array: array is 2-dimensional, but 3
were indexed USE [::-1, :]
#IF ValueError: too many values to unpack (expected 2) USE [::-1, :,0]
# otherwise use[::-1,:,0]
# You have access to 9-> 0 numbers by changing the X in number-X.png to whatever
number you want

######### Refactorered but function from Brian2 documentation [48]
greyscale_img = (1-imread('number-8.png'))[::-1, :,0].T
#greyscale_img = (1-imread('machine.png'))[::-1, :,0].T

# .shape returns number shape of image by X by X , second
# Value gives dimensions needed for number of nodes of the input image
```

```
time,number_neurons = greyscale_img.shape
timing_im = TimedArray(greyscale_img, dt=1*ms)
################################################

#### Refactorered but function from Brian2 documentation [48]
# Basic LIF implementation where the reversal potential
# is replaced with a timed array of the spike events
eqs = '''
dv/dt = (1.95*timing_im(t, i)- v)/tau_M + 0.02*sqrt(2/tau_M)*xi: 1

'''
################################################


# Neurons exposed to image
#-------------------------------------------------
# Threshold condiitons [48]
im_neuron = NeuronGroup(number_neurons, eqs, threshold='v >1', reset='v = 0',
method='euler')
# Usual input neuron group instantiation

input_neuron = NeuronGroup(1,izhikevich_eqns,threshold = 'v >=30',reset ='v = c; u
+=d;',method='euler')
# Connecting the image neurons to the main model
image_input =Synapses(im_neuron,input_neuron,model='w:1',on_pre ="v_post
+=0.2",method = 'euler')
image_input.connect()
image_input.w =1
#-------------------------------------------------


#### Setting up model architecture
PVC =NeuronGroup(9,izhikevich_eqns,threshold = 'v >=30',reset ='v = c; u
+=d;',method='euler')
# Input to IVCBeta
in_PVC_0 = Synapses(input_neuron,PVC[0],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_0.connect()
in_PVC_0.w =1
# Input to IVCAlpha -correct
in_PVC_1 = Synapses(input_neuron,PVC[1],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_1.connect()
in_PVC_1.w =1
# Input to || & ||| node 1
in_PVC_4 = Synapses(input_neuron,PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_4.connect()
in_PVC_4.w =1
##############################
# IVCBeta to IVB
PVC_0_PVC_2 = Synapses(PVC[0],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_2 .connect()
PVC_0_PVC_2 .w =1
```

```
# IVCBeta to IVCAlpha
PVC_0_PVC_1 = Synapses(PVC[0],PVC[1],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_1 .connect()
PVC_0_PVC_1 .w =1
#
# IVCBeta to L || & ||| node 1
PVC_0_PVC_4= Synapses(PVC[0],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_4 .connect()
PVC_0_PVC_4.w =1
#########################

# IVCAlpha to node 2  L || & |||
PVC_1_PVC_5= Synapses(PVC[1],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_1_PVC_5 .connect()
PVC_1_PVC_5.w =1

########################

# IVB to node 3 in L || & |||
PVC_2_PVC_6 = Synapses(PVC[2],PVC[6],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_6 .connect()
PVC_2_PVC_6.w =1

# IVB to Node 1
PVC_2_PVC_5 = Synapses(PVC[2],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_5 .connect()
PVC_2_PVC_5.w =1

# IVB to IVCBeta
PVC_2_PVC_0 = Synapses(PVC[2],PVC[0],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_0 .connect()
PVC_2_PVC_0.w =1
########################
# Layer || & ||| node 0 - PVC_3

# L || & ||| Node 0 to Node 1
PVC_3_PVC_4 =Synapses(PVC[3],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_4 .connect()
PVC_3_PVC_4.w =1
#################
# L || & ||| node 0 to Lv node 1
PVC_3_PVC_8=Synapses(PVC[3],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
              on_post='''Apost += dApost
                      w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_8 .connect()
```

```
PVC_3_PVC_8.w =1
# L || & ||| node 0 to Lv node 0
PVC_3_PVC_7=Synapses(PVC[3],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_7 .connect()
PVC_3_PVC_7.w =1

# Node 1 lAYER || & ||| to IVB
PVC4_PVC2 =Synapses(PVC[4],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC4_PVC2 .connect()
PVC4_PVC2.w =1
#
# Node 3 lAYER || & ||| to IVB
PVC6_PVC2 =Synapses(PVC[6],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC6_PVC2 .connect()
PVC6_PVC2.w =1
#node 2 - PVC_5
# 2 to 1
PVC_5_PVC_4 =Synapses(PVC[5],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_4 .connect()
PVC_5_PVC_4.w =1
# Layer || to ||| node 2 to node 0
PVC_5_PVC_3 =Synapses(PVC[5],PVC[3],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_3 .connect()
PVC_5_PVC_3.w =1

# LV connections
PVC_7_PVC_8  = Synapses(PVC[7],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_7_PVC_8 .connect()
PVC_7_PVC_8.w =1
#
PVC_8_PVC_7  = Synapses(PVC[8],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_8_PVC_7 .connect()
PVC_8_PVC_7.w =1
#
PVC_8_PVC_4  = Synapses(PVC[8],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_8_PVC_4 .connect()
PVC_8_PVC_4.w =1
################ End of connection of architecture


#Setting Excitatory Regular Firing Patterns for Neurons [5][23]
#---------------------------------------------------------
input_neuron.a = 0.02
```

```
input_neuron.b = 0.2
input_neuron.c = -65
input_neuron.d = 8
input_neuron.u =0.2*-65
input_neuron.v = v_r
#------node 0 L || & |||
PVC[3].a = 0.02
PVC[3].b = 0.2
PVC[3].c = -65
PVC[3].d = 8
PVC[3].u =0.2*-65
PVC[3].v =v_r
#-------------------
PVC[4].a = 0.02
PVC[4].b = 0.2
PVC[4].c = -65
PVC[4].d = 8
PVC[4].u =0.2*-65
PVC[4].v =v_r
#-------------------
PVC[2].a = 0.02
PVC[2].b = 0.2
PVC[2].c = -65
PVC[2].d = 8
PVC[2].u =0.2*-65
PVC[2].v = v_r
#--------------------
PVC[7].a = 0.02
PVC[7].b = 0.2
PVC[7].c = -65
PVC[7].d = 8
PVC[7].u =0.2*-65
PVC[7].v = v_r

#Setting Inhibitatory Thalamo-Cortical Firing Patterns for Neurons [23]
#-----------------------------------------------------------------------
PVC[1].a = 0.02
PVC[1].b = 0.25
PVC[1].c = -65
PVC[1].d = 0.05
PVC[1].u =0.25*-65
PVC[1].v = v_r
#------------------
PVC[0].a = 0.02
PVC[0].b = 0.25
PVC[0].c = -65
PVC[0].d = 0.05
PVC[0].u =0.25*-65
PVC[0].v = v_r
#-----------------
PVC[5].a = 0.02
PVC[5].b = 0.25
PVC[5].c = -65
PVC[5].d = 0.05
PVC[5].u =0.25*-65
PVC[5].v =v_r
#------------------
PVC[6].a = 0.02
PVC[6].b = 0.25
```

```
PVC[6].c = -65
PVC[6].d = 0.05
PVC[6].u =0.25*-65
PVC[6].v =v_r
#-------------------
PVC[8].a = 0.02
PVC[8].b = 0.25
PVC[8].c = -65
PVC[8].d = 0.05
PVC[8].u =0.25*-65
PVC[8].v =v_r
#------------------

#Setting up variables to minitor the states of the different outputs and hidden
nodes
statm = StateMonitor(input_neuron,'v', record=0)
statm_0 = SpikeMonitor(input_neuron)
IVCBETA = StateMonitor(PVC[0],'v', record=0)
IVCALPHA = StateMonitor(PVC[1],'v', record=0)
IVB = StateMonitor(PVC[2],'v', record=0)
IVB_SM = SpikeMonitor(PVC[2])
# L|| to ||| --------------------------------
L2to3_0 = StateMonitor(PVC[3],'v', record=0)
L2to3_1 = StateMonitor(PVC[4],'v', record=0)
L2to3_1_SM = SpikeMonitor(PVC[4])
L2to3_2 = StateMonitor(PVC[5],'v', record=0)
L2to3_3 = StateMonitor(PVC[6],'v', record=0)
L2to3_3_SM = SpikeMonitor(PVC[6])
# LV ----------------------------------------
LV_0 = StateMonitor(PVC[7],'v', record=0)
LV_0_SM = SpikeMonitor(PVC[7])
LV_1 = StateMonitor(PVC[8],'v', record=0)

spikmon = SpikeMonitor(im_neuron)

#
run(time_ran,report ='text')


# Save results from test
#save_results(statm_0,IVB_SM,L2to3_1_SM,L2to3_3_SM,LV_0_SM)

#Plot input to output response including responses of hidden nodes
plot_output(spikmon,time,number_neurons,IVCBETA,IVCALPHA,IVB,L2to3_0,L2to3_1,L2to3_
2,L2to3_3,LV_0,LV_1)
```

U. Event-based Phase 2 Demo code

```
import cv2
from brian2 import*
from matplotlib.image import imread
import keyboard

# Functions
# Library cv2 is from [57][58]
# Uses Brian2 [69]

# there is code to attempt to compile this slow
#model intpo C++ but Brian2's compilation
# has functions that cant be found?

# Model runs slow so to take continuous input
# either close the plot and rerun
# or put show(block ='false') at end


# 0 for main camera e.g. laptop webcam , 1 for external e.g. plugged in webcam
# (sometimes other way around)
cam = cv2.VideoCapture(0)
tau_M = 10*ms
#Tau STDP time constants [45]
tau_pre= 20*ms
tau_post =20*ms
v_r = -70


#STDP Source Abbor ,Song, Miller [45]
gmax = 0.2
dApre = 0.01
dApost = -dApre * tau_pre / tau_post * 1.05
dApost *= gmax
dApre *= gmax
###############

 #### Refactored function from Brian2 [48]###########
# Basic LIF implementation where the reversal potential
# is replaced with a timed array of the spike events
#Different constants to toher vision files as this gave better
# results
eqs = '''
dv/dt = (2.2*timing_im(t, i)- v)/tau_M + 0.2*sqrt(2/tau_M)*xi: 1

'''
################################################
#STDP Source Abbor ,Song, Miller [45]
stdp_eqn = '''
dApre/dt = -Apre/tau_pre : 1 (event-driven)
dApost/dt = -Apost/tau_post : 1 (event-driven)
w : 1
'''
###
# E.Izhikevich  [46]
pre_eqn ='''
I += w
Apre += dApre
```

```
w = clip(w + Apost, 0, gmax)'''

izhikevich_eqns = '''
dv/dt = (0.04*v*v + 5*v + 140 -u + I)/ms + 2*sqrt(2/tau_M)*xi: 1
du/dt = a*(b*v - u)/ms : 1
a : 1
b : 1
c : 1
d : 1
I : 1
'''

read_corr = True
# Continually takes image from webcam for live feed
# Only runs whilst webcam reports it has been read correctly
while read_corr:
    ##### Output C++ so can run with CUDA
    #set_device("cpp_standalone", build_on_run=False)

    # reads input from camera
    # first field boolean if it is read correct - not used
    read_corr, image = cam.read()
    print(read_corr)
    # resize the image according to fx and fy
    frame = cv2.resize(image,None,fx=1, fy=1)
    # Write the captured image from the webcam to files
    cv2.imwrite("webcam.png",image)
    # Implement function to convert to event based image [48]####
    image_n = (1-imread('webcam.png'))[::-1, :,0].T
    time,num_neurons = image_n.shape
    im_neuron = NeuronGroup(num_neurons, eqs, threshold='v >1', reset='v = 0',
method='euler')
    timing_im = TimedArray(image_n, dt=1*ms)
    ##]########################################
    # Run The phase 2 primary visual cortex model
    input_neuron = NeuronGroup(1,izhikevich_eqns,threshold = 'v >=30',reset ='v =
c; u +=d;',method='euler')
    # Connecting the image neurons to the main model
    image_input =  Synapses(im_neuron,input_neuron,model='w:1',on_pre ="v_post
+=0.2",method = 'euler')
    image_input.connect()
    image_input.w =1
    #### Setting up model architecture
    PVC =NeuronGroup(9,izhikevich_eqns,threshold = 'v >=30',reset ='v = c; u
+=d;',method='euler')
    # Input to IVCBeta - correct
    in_PVC_0 = Synapses(input_neuron,PVC[0],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
    in_PVC_0.connect()
    in_PVC_0.w =1
    # Input to IVCAlpha -correct
    in_PVC_1 = Synapses(input_neuron,PVC[1],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
    in_PVC_1.connect()
    in_PVC_1.w =1
    # Input to || & ||| node 1 - correct
    in_PVC_4 = Synapses(input_neuron,PVC[4],model =stdp_eqn,on_pre=pre_eqn,
```

```
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
in_PVC_4.connect()
in_PVC_4.w =1
##############################
# IVCBeta to IVB - correct
PVC_0_PVC_2 = Synapses(PVC[0],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_2 .connect()
PVC_0_PVC_2 .w =1
# IVCBeta to IVCAlpha
PVC_0_PVC_1 = Synapses(PVC[0],PVC[1],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_1 .connect()
PVC_0_PVC_1 .w =1
#
# IVCBeta to L || & ||| node 1 - correct
PVC_0_PVC_4= Synapses(PVC[0],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_0_PVC_4 .connect()
PVC_0_PVC_4.w =1
#########################

# IVCAlpha to node 2  L || & ||| - correct
PVC_1_PVC_5= Synapses(PVC[1],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_1_PVC_5 .connect()
PVC_1_PVC_5.w =1

#######################

# IVB to node 3 in L || & |||- correct
PVC_2_PVC_6 = Synapses(PVC[2],PVC[6],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_6 .connect()
PVC_2_PVC_6.w =1

# IVB to Node 1
PVC_2_PVC_5 = Synapses(PVC[2],PVC[5],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_5 .connect()
PVC_2_PVC_5.w =1

# IVB to IVCBeta
PVC_2_PVC_0 = Synapses(PVC[2],PVC[0],model =stdp_eqn,on_pre=pre_eqn,
                on_post='''Apost += dApost
                        w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_2_PVC_0 .connect()
PVC_2_PVC_0.w =1

##############################
# Layer || & ||| connections
##############################
```

```
# Layer || & ||| node 0 - PVC_3

# L || & ||| Node 0 to Node 1 - correct
PVC_3_PVC_4 =Synapses(PVC[3],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_4 .connect()
PVC_3_PVC_4.w =1
################
# L || & ||| node 0 to Lv node 1- correxct
PVC_3_PVC_8=Synapses(PVC[3],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_8 .connect()
PVC_3_PVC_8.w =1
# L || & ||| node 0 to Lv node 0 - correct
PVC_3_PVC_7=Synapses(PVC[3],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_3_PVC_7 .connect()
PVC_3_PVC_7.w =1
# node 1 PVC_4
# Node 1 lAYER || & ||| to IVB - correct
PVC4_PVC2 =Synapses(PVC[4],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC4_PVC2 .connect()
PVC4_PVC2.w =1
#
# Node 3 lAYER || & ||| to IVB -corrrect
PVC6_PVC2 =Synapses(PVC[6],PVC[2],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC6_PVC2 .connect()
PVC6_PVC2.w =1
#node 2 - PVC_5
# 2 to 1 - correct
PVC_5_PVC_4 =Synapses(PVC[5],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_4 .connect()
PVC_5_PVC_4.w =1
# Layer || to ||| node 2 to node 0 - correct
PVC_5_PVC_3 =Synapses(PVC[5],PVC[3],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_5_PVC_3 .connect()
PVC_5_PVC_3.w =1

# LV connections - correct
PVC_7_PVC_8  = Synapses(PVC[7],PVC[8],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_7_PVC_8 .connect()
PVC_7_PVC_8.w =1
#
PVC_8_PVC_7  = Synapses(PVC[8],PVC[7],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
```

```
PVC_8_PVC_7 .connect()
PVC_8_PVC_7.w =1
#
PVC_8_PVC_4  = Synapses(PVC[8],PVC[4],model =stdp_eqn,on_pre=pre_eqn,
             on_post='''Apost += dApost
                    w = clip(w + Apre, 0, gmax)''',method = 'euler')
PVC_8_PVC_4 .connect()
PVC_8_PVC_4.w =1
################ End of connection of architecture

#Setting Excitatory Regular Firing Patterns for Neurons  [5][23]
#-----------------------------------------------------------
input_neuron.a = 0.02
input_neuron.b = 0.2
input_neuron.c = -65
input_neuron.d = 8
input_neuron.u =0.2*-65
input_neuron.v = v_r
#------node 0 L || & |||
PVC[3].a = 0.02
PVC[3].b = 0.2
PVC[3].c = -65
PVC[3].d = 8
PVC[3].u =0.2*-65
PVC[3].v =v_r
#-------------------
PVC[4].a = 0.02
PVC[4].b = 0.2
PVC[4].c = -65
PVC[4].d = 8
PVC[4].u =0.2*-65
PVC[4].v =v_r
#-------------------
PVC[2].a = 0.02
PVC[2].b = 0.2
PVC[2].c = -65
PVC[2].d = 8
PVC[2].u =0.2*-65
PVC[2].v = v_r
#--------------------
PVC[7].a = 0.02
PVC[7].b = 0.2
PVC[7].c = -65
PVC[7].d = 8
PVC[7].u =0.2*-65
PVC[7].v = v_r

#Setting Inhibitatory Thalamo-Cortical Firing Patterns for Neurons [23]
#---------------------------------------------------------------------
PVC[1].a = 0.02
PVC[1].b = 0.25
PVC[1].c = -65
PVC[1].d = 0.05
PVC[1].u =0.25*-65
PVC[1].v = v_r
#------------------
PVC[0].a = 0.02
PVC[0].b = 0.25
PVC[0].c = -65
```

```
PVC[0].d = 0.05
PVC[0].u =0.25*-65
PVC[0].v = v_r
#------------------
PVC[5].a = 0.02
PVC[5].b = 0.25
PVC[5].c = -65
PVC[5].d = 0.05
PVC[5].u =0.25*-65
PVC[5].v =v_r
#-------------------
PVC[6].a = 0.02
PVC[6].b = 0.25
PVC[6].c = -65
PVC[6].d = 0.05
PVC[6].u =0.25*-65
PVC[6].v =v_r
#--------------------
PVC[8].a = 0.02
PVC[8].b = 0.25
PVC[8].c = -65
PVC[8].d = 0.05
PVC[8].u =0.25*-65
PVC[8].v =v_r

IVCBETA = StateMonitor(PVC[0],'v', record=0)
IVCALPHA = StateMonitor(PVC[1],'v', record=0)
IVB = StateMonitor(PVC[2],'v', record=0)
IVB_SM = SpikeMonitor(PVC[2])
# L|| to ||| --------------------------------
L2to3_0 = StateMonitor(PVC[3],'v', record=0)
L2to3_1 = StateMonitor(PVC[4],'v', record=0)
L2to3_1_SM = SpikeMonitor(PVC[4])
L2to3_2 = StateMonitor(PVC[5],'v', record=0)
L2to3_3 = StateMonitor(PVC[6],'v', record=0)
L2to3_3_SM = SpikeMonitor(PVC[6])
# LV -----------------------------------------
LV_0 = StateMonitor(PVC[7],'v', record=0)
LV_0_SM = SpikeMonitor(PVC[7])
LV_1 = StateMonitor(PVC[8],'v', record=0)

spikmon = SpikeMonitor(im_neuron)

# Run brian2 simulation
run(time*ms,report ='text')
#device.build(directory='output', compile=True, run=True, debug=False)
#################################################################
# Plotting event based camera output
subplot(311) # part similar [48]
plot(spikmon.t/ms,spikmon.i, '.k', ms=3)
xlim(0,time)
ylim(0, num_neurons)
xlabel('Time (ms)')
ylabel('Neuron index')
#################################################################
#Plotting model response
#\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\\/\/\/\\/
subplot(312)
plot(LV_1.t/ms,LV_1.v[0],label ='LV_1')
```

```
    plot(L2to3_0.t/ms,L2to3_0.v[0],label ='L-2to3[0]')
    plot(L2to3_2.t/ms,L2to3_2.v[0],label ='L-2to3[2]')
    plot(IVCALPHA.t/ms,IVCALPHA.v[0],label ='IVCAlpha')
    plot(IVCBETA.t/ms,IVCBETA.v[0],label ='IVCBeta')
    legend()
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    # Plotting outputs of system
    subplot(313)
    plot(IVB.t/ms,IVB.v[0],label ='IVB')
    plot(L2to3_1.t/ms,L2to3_1.v[0],label ='L-2to3[1]')
    plot(L2to3_3.t/ms,L2to3_3.v[0],label ='L-2to3[3]')
    plot(LV_0.t/ms,LV_0.v[0],label ='LV [0]')
    xlabel('Time (ms)')
    ylabel('Voltage of spike (mV)')
    legend()
    tight_layout()
    #/\\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\\/\/\/\/\/\/\/\\

    # Dont pause whole simulation because of plot "block =False"
    #show image capture to screen
    cv2.imshow('Input', frame)
    show()
    #show image capture to screen
    cv2.imshow('Input', frame)
    pause(0.9)
    #block =False
    # Hold it on screem
    # Clear the plot to allow next frame to show
    clf()

    # Uncomment these lines to aid compillation of C++ code
##    device.reinit()
##    device.activate()
    # Have exit key to allow for closure
    if keyboard.is_pressed('esc'):
        break


cam.release()
cv2.destroyAllWindows()
```