DUKE UNIVERSITY, BIOMEDICAL ENGINEERING

Project 1 Final Report

Jaehong Yoon

October 13, 2016

CONTENTS

1 INTRODUCTION

The thinking process of any creature is a miracle which has attracted interest around various fields. When with modern technology only limited knowledge of this enigmatic organ has been discovered. After the advent of the modern neuroscience with Golgi's invention of silver staining [1], various approaches have been followed to explore the enigma of how brain works, especially how electrical signals travel down the axon and pass between neurons via synapse. The equations formulated by Hodgkin and Huxley in 1952 [2] was a great impact on the field as it first demonstrated how voltage gated ion channels involve in transferring signals in neurons. Although a simple and empirical in nature, Hodgkin and Huxley type model (HH model) gives incite of physiological characteristics of action potential (AP). More recently, Izhikevich [3] has formulated a simple but robust model for membrane potential of neuron. The integrate and fire type model (IAF model) is computationally simple, yet capable of building various firing pattern, especially on cortical neurons which has been used for computational simulation of large-scale brain modeling and neuromorphic networks[4, 5, 6].

One of the fields which incorporates the membrane potentials and synaptic current in real-life problem is an neuroethology. The intelligence guiding the insect to maneuver around the surrounding is nothing like cognitive ability of human, yet is marvelous ability when considered how small their brain capacity are. Initial concept of artificial insect comes from Italian neuroscientist Valentino Braitenberg [7]. Braitenberg got interest in the work of brain in 1948 after his first observation of human brain tissue in vitro under microscope. The incredible complex behavior and connection of primitive neural structure of insects fascinated the scientist. He later raised a simple neuro-mimic robotic circuit which exhibits complex behavior. Although insects incorporate hundreds of thousands of nerve cells to make decision, the artificial insect still gives insight of how to design a robust but simple adaptive control system [8]. The adaptive ability of nature, even in its most primitive form, raise an idea of constructing artificial neural network for autonomous system. These system interact with its surrounding by collecting data from environment, change its behavior and even feedback the data to the system. The system is more advanced than traditional neural networks. Instead of training in fixed set of training data, the artificial insect sets its strategy by behaving and training at runtime in its environment [9].

This report shows a noble artificial insect designed to maneuver in its environment and interact with four events to set its survival strategy; i) food, ii) poison, iii) mate and iv) enemy. The simulated bug has an artificial nervous system built using Python package Brian2. The synaptic connection between sensors and motor neurons enable to perform some fundamental tasks of either 1) escape or 2) approach to the given object. Three synapse model, i) alpha synapse, ii) NMDA and iii) $GABA_{\beta}$ was used to introduce complex decision making process of this artificial insect. The artificial nervous system designed for this project will give insight of how complex behavior can be made with only few neurons.

2 METHOD

This section will introduce the simulating method and package used for designing the artificial insect of this project. For neurons, Izhikevich model was implemented for firing pattern and three different synapse model was used for each sensor neurons. Four sensors were used for detection of objects. Braitenberg circuits were modified for design of synaptic connection [7].

2.1 BRIAN 2

Brain 2 is a simulator package of Python which enables to design spiking neural network. One can plug in a set of equations describing the neuron and connect each neuron using "Synapse" function [10]. The package enables to use quantities with physical dimensions. Standard SI units can be used with prefixes. Brian script starts with importing **Brain 2** package, same as other python packages. Neuron can be modeled with defining the differential equations and number of the neuron in a single group. The equations can be written in simple text base with **'''** notation for multi-line strings. Units for the variables should be written at the end of each line of one**: unit**. Units should be written either 1 (no unit) or in plain SI units as aforementioned. The package includes **matplotlib** package which can be used to plot figures as in MATLAB.

After the equations are set, one can start building the neuron using the class **NeuronGroup**. The number of neurons and equation for defining firing should be given for the class. The integration method can be selected or not declared. If not declared, the integration will automatically selected to be "euler" method by default. The incident of firing can be traced by setting the "threshold." A refractory may be given either 1) for fixed time or 2) during the membrane voltage is at certain range. One of the unique function of class **NeuronGroup** of Brian 2 is that one can reset the voltage whenever the membrane potential reaches the threshold. This can be done by setting the "reset." The reset is crucial for success of your model as the differential equations for neuron will depend on current status of defined variables.

Two monitor method can be implemented to check variables. The object **StateMonitor** records the values of the variables in an list for given neuron. The recorded values may be used latter to analyze firing or conductance change pattern for given stimuli. On the other hand **SpikeMonitor**, records the spikes or firing of a neuron. Internally, when the membrane voltage exceed threshold (reach supra-threshold), Brian will register the incident as "spike." **SpikeMonitor** object enables user to access the 1) spike time and 2) spiking rate. For this project, only **StateMonitor** was used to analyze firing and conductance change pattern of the insect accordingly to the event it encounters.

Next the user can define the synapse using **Synapses** object. The **Synapses** object

takes source and target neurons as input. When the source and target neuron of synapse is defined, the actual synapse can be connected by typing Synpases.connect(source = 0, target = 1). This way, the synapse connects from source neuron to the target synapse. The pseudo-code below gives a example of how to define **NeuronGroup** and **Synapses**.

```
from Brain2 import *

# define paramaters
v_th = 20.*mV
v_reset = 0.*mV
tau = 10.*ms
num_neuron = 1 # number of neurons per group

# define differential equations
# for neurons
eqs = '''
dv/dt = v/tau : volt
'''

# construct NeuronGroups
# two neurons are constructed in this example
Neuron1 = NeuronGroup(num_neuron, eqs,
                threshold = 'v > v_th',
                refractort = 1.0*ms,
                method = 'linear'
                )

Neuron2 = NeuronGroup(num_neuron, eqs,
                threshold = 'v > v_th',
                refractort = 1.0*ms,
                method = 'linear'
                )

# set monitor object
state = StateMonitor(Neuron1,'v', record = True)
spike = SpieMonitor(Neuron2)

# connect Neuron1 to Neuron2 (Synapse)
Synapse = Synapses(Neuron1, Neuron2, clock = Neuron1.clock,
                                    model = '''
                                    w : volt
                                    '''
                                    on_pre = '''
                                    v += w
```

```
                                                    , , ,
)
Synapse.connect(i=[0],j=[0])
Synapse.w = 100*mV
Synapse.delay = 5*ms
```

The "model" defines the weight to be given to the post-synaptic neuron when spike occur on pre-synaptic neuron. The value of weight "w" can be set to be either a function of variable or fixed value. Here it was set to be 100 mV (**Synapse.w = 100*mV**)the "on_pre" syntax declares to add defined weight to the post-synaptic neuron when pre-synaptic neuron fires. One may also define the delay for synapse. For this example code, delay was set to be 5 msec (**Synapse.delay = 5*ms**). Implementing delay will give more realistic simulation as most neurons will have time interval for neurotransmitter to reach post-synaptic neuron.

2.2 NERVE SYSTEM

The success of artificial nerve system depends on both the model of spiking dynamics and the model of synapse. Depending on one's choice for these two crucial parts will make difference in the performance. For successful modeling, therefore, both appropriate spiking model as well as synapse model should be implemented for physiologically meaningful and computationally realistic. Hodgkin–Huxley model (HH model), which is most important model in computational neuroscience, is biologically most plausible, was the first model spiking model to be developed. The model describes how firing of neuron occurs and propagated with sets of nonlinear differential equations. Dr. Hodgkin and Huxley explored the firing pattern and conductance change of giant squid's axon. By implementing Kirchhoff's law, the membrane of axon was designed as circuit with membrane capacitance (C_m) and voltage-gated Sodium and Potassium channels [2]. For model with synaptic current, the model can be modified as following [11].

$$\begin{aligned} C_m \frac{dv}{dt} + g_{Na}(v - E_{Na}) + g_K(v - E_K) + g_{leak}(v - E_{leak}) \\ + g_{ex}(v - E_{ex}) + g_{inh}(v - E_{inh}) = I_{stim}(t) \end{aligned} \tag{2.1}$$

Although the HH model gives the best biological plausibility, the implementation cost is huge. With four ODEs and ten parameters, the computational power required to simulate the model increase exponential as number of neurons increase [3]. The best alternative of HH model is Izhikevich model [12] which can generate various type of spiking model with two simple ODE as following.

$$
\begin{aligned}
\dot{v} &= 0.04v^2 + 5v + 140 - u + I \\
\dot{u} &= a(bv - u)
\end{aligned}
\tag{2.2}
$$

$$
\text{resets: if v} \geq 30 \text{ mV, then } = \begin{cases} v \longleftarrow c \\ u \longleftarrow u + d \end{cases}
$$

v and u of the equation 2.2 represent a membrane potential and recovery variable respectively. The recovery variable account for inactivation of Sodium ion channel which puts negative feedback to the membrane potential. When the membrane voltage reach threshold, **Brian 2** resets the membrane voltage and recovery variable to defined values. Even with two simple ODEs, it can be used to model all types of known cortical neuron firing pattern [3]. For computational ease, this model was used to define spiking along the axon.

For synapse, three different models were used to implement insect movement accordingly to the given environment. The insect was designed to escape from both poisoned food and enemy and approach food and mate inside its environment. For food and poison detection, alpha function synapse was used. Alpha function is a great model for most of the synapse as it describes the rising phase with certain time constant. The limits of alpha function is that due to single time constant, conductance change of synapse from rapid binding and slow unbinding of the transmitter cannot be modeled appropriately [13]. For more physiologically meaningful model, modified alpha function with two time constant (each for rise and decay phase) were implemented. Equation 2.3 describes the ODE of alpha synapse conductance.

$$
\begin{aligned}
\dot{g} &= \frac{-g}{\tau_{decay}} + z(t) \\
\dot{z} &= \frac{-z}{\tau_{rise}} + \delta(t)\bar{g}_{syn2} \\
\text{peak time } &= \frac{\tau_{decay}\tau_{rise}}{\tau_{decay} - \tau_{rise}} ln(\frac{\tau_{decay}}{\tau_{rise}}) \\
\bar{g_{syn2}} &= \frac{g_{peak}}{(\frac{\tau_{decay}\tau_{rise}}{\tau_{decay} - \tau_{rise}})(exp(-\frac{\text{peak time}}{\tau_{decay}}) - exp(-\frac{\text{peak time}}{\tau_{rise}}))}
\end{aligned}
\tag{2.3}
$$

For synapse between motor and mate sensor neuron, NMDA channel was implemented. The NMDA is a slow voltage-gated channel which activates by glutamate [14]. This channel requires AMPA receptor to activate first to depolarize the membrane. When the membrane is sufficiently depolarized, Magnesium ion blocking the channel is relieved and the channel activates. The NMDA is crucial for long-term potentiation (LTP) [15]. To implement LTP-eque learning behavior, the author has coded the synaptic conductance between

mate sensor and motor to increase by 10 % when the bug success mating. This way, the insect will develop a cognitive process to approach mate with more confidence as it *learns*. Furthermore, the slow opening of NMDA will enable to make the insect jitter in first interaction with the mate which will represent the *shyness*. Equation 2.4 describes the ODE of NMDA receptor synapse [11].

$$
\begin{aligned}
I_{NMDA}^{post} &= g_{NMDA} B(V_{post}) s(t) (V_{post} - E_{NMDA}) \\
\frac{ds}{dt} &= \alpha T(1-s) - \beta s \begin{cases} \alpha = 0.072 \text{ mM}^{-1}\text{ms}^{-1} \\ \beta = 0.0066 \text{ ms}^{-1} \end{cases} \\
B(V) &= \frac{1}{1 + \frac{e^{-0.062V}[Mg]^{2+}}{3.57}}
\end{aligned}
\tag{2.4}
$$

One of the

Finally, for synaptic connection between motor and enemy sensor neurons, GABA_β type model was used. GABA_β is a G-coupled protein receptor type neurotransmitter which slow inhibitory synapse. However, the cascade of process enables amplification of signal [14]. Therefore, the synaptic conductance of GABA_β channel reach can reach greater peak conductance and last long. Normally, this neurotransmitter acts as a slow inhibitory signal. However, for this project, the synapse was used as excitatory synapse by modifying the reversal potential. This enables the bug to achieve fast speed and long duration time to escape from the enemy. The ODEs for GABA_β synapse is present in equation 2.5.

$$
\begin{aligned}
I_{GABA_\beta}^{post} &= g_{GABA_\beta} \frac{s^4}{s^4 + K_d} (V_{post} - E_{GABA_\beta}) \\
\frac{dr}{dt} &= \alpha T(1-r) - \beta r \\
\frac{ds}{dt} &= K_3 r - K_4 s.
\end{aligned}
\tag{2.5}
$$

The values for α, β, and K's can be found in [13].

2.3 Artificial Insect

Methodology of artificial insect synaptic connection of Braitenberg [7] was implemented for construction of artificial insect for this project. Inspired by intellect of insects, Braitenberg has postulated simple neuromorphic circuit which results in seemingly complex or even intelligent behavior. The "vehicle", an autonomous agent, maneuvers around its environment

accordingly to the stimulus. Sensors attached to these vehicles measure the stimulus as function of distance and the motor derives movement accordingly to the configuration of the connection between motor and sensor. For instance, if the sensors and motor sensors are connected ipsilateral and the sensor is excitatory, the vehicle will rush toward the object which is detected by the sensors. The vehicle can strive to achieve or avoid the situation which produce complex combination of behavior just as real insect. Braitenberg have originally produced four combinations of motor-sensor synaptic connection for different situation as in Table 1

Table 1: Summary of Braitenberg vehicle types accordingly to their sensors and synapse configuration

Sensor Type	Sensor-Motor Synapse wiring	Behavior
Excitatory	Contralateral	Aggression
Inhibitory	Contralateral	Love
Exitatory	Ipsilateral	Fear
Inhibitory	Ipsilateral	Liking

For the artificial insect of this project, two Braitenberg-esque behaviors were implemented; aggression and fear. For food and mate, artificial insect was designed to achieve the objects. As "aggression" provide a mean of direct approaching movement, it was used to design synapses for food and mate. However, the insect was supposed to avoid poison and enemy for survivial. Fear (or coward in some text) connection was therefore used to design synapses for these objects. The fear type synapse, which connects sensor and motor contralateral, will make the insect detour and avoid the object as speed of motor farther from the object become faster. For each sensors, different synaptic currents were implemented accordingly to desired movement as aforementioned in section **2.2 Nerve system**. Table 2 summarize the synaptic current and configuration for each sensors used in this project. Schematics of the artificial insect and synapse configuration is shown in Figure 1.

Table 2: Summary of synaptic current and Braitenberg type behaviors implemented for each sensor of the artificial insect in this project.

Object	Synaptic Currenet	Braitenberg Behavior
Food	Alpha function	Aggression
Poison	Alpha function	Fear
Mate	NMDA	Aggression
Enemy	$GABA_{\beta}$	Fear

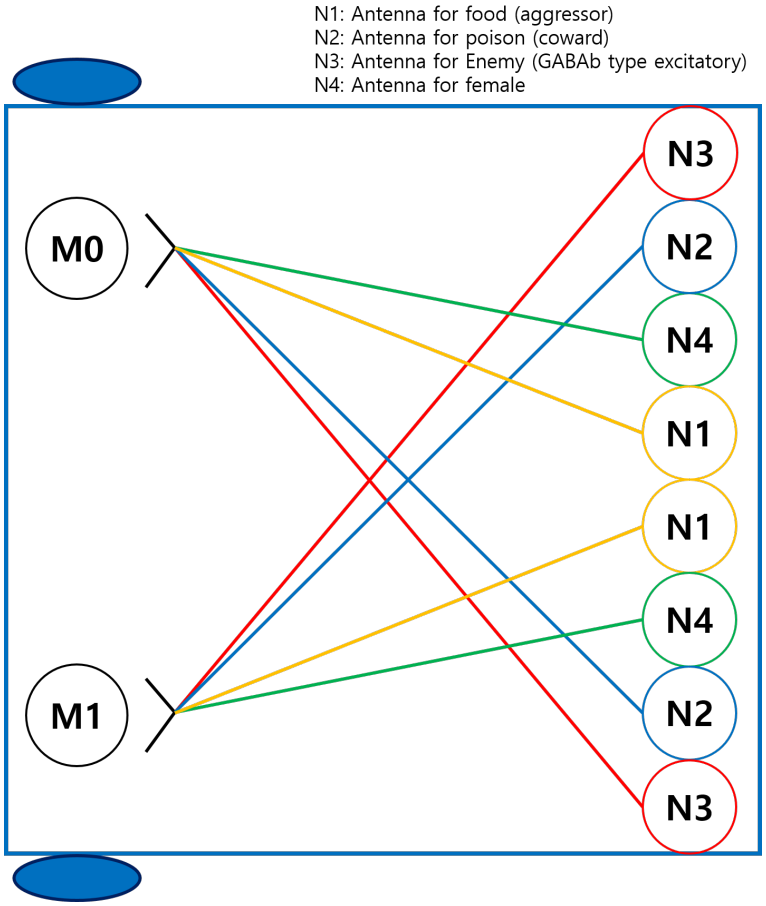

Figure 1: Schematic of artificial insect synaptic configuration and type of behaviors

2.4 Experiment

To simulate the insect, an environment with four objects (food, poison, mate, and enemy) was constructed. Inside the environment, the insect was placed in random position for start. Depending on the object it encounters, the insect changed its courses accordingly. The position of food and mate changed when the insect touched the objects and position of objects and artificial insect was updated continuously for simulation. Sensors and motor connections were made as aforementioned in section **2.3 Artificial Insect**. The algorithm of simulation is shown in Algorithm box 1 below. Code for simulation can be found in Appendix 1.

Algorithm Box 1 Algorithm for simulation of artificial insect

Step1: Set global variables: (food/poison/mate/enemy coordinates/counts)

Step2: Define ODEs of
(1) Food sensor: Izhikevich + Alpha synapse
(2) Poison sensor: Izhikevich + Alpha synapse
(3) Mate sensor: Izhikevich + NMDA synapse
(1) Enemy sensor: Izhikevich + $GABA_{\beta}$ synapse

Step3: Define Sensor/Motor Neurons

Step4: Connect Sensors to motor

Step5:
for t = 0:dt:Duration **do**

$\dot{bug}.x$ = (g_{food}(v-E_{food}) + g_{poison}(v-E_{poison})+g_{mate}(v-E_{mate})
+ g_{Enemy}(v-E_{Enemy})) × sin(θ)
$\dot{bug}.y$ = (g_{food}(v-E_{food}) + g_{poison}(v-E_{poison})+g_{mate}(v-E_{mate})
+ g_{Enemy}(v-E_{Enemy})) × cos(θ)

if (bug.x - food.x)^2 + (bug.y - food.y)^2 < 10 **then**
food.x = random position
food.y = random position
end if

if (bug.x - mate.x)^2 + (bug.y - mate.y)^2 < 10 **then**
mate.x = random position
mate.y = random position
end if

bug.x = bug.x + $\dot{bug}.x$dt
bug.y = bug.y + $\dot{bug}.y$dt

Update bug, food, poison, mate, enemy position

end for

The enemy and mate were artificial insects coded for special objectives as following.

$$\begin{aligned} &\text{enemy : catch both bug and enemy} \\ &\text{mate : avoid enemy and mate with bug} \end{aligned} \tag{2.6}$$

The mate was immune to poison so even if the mate hits the poison, it survives. Whenever the bug hits the poison, position for poison was updated. When the enemy catches the prey (bug or mate) the number of catch was incremented by 1 and the prey was relocated at random position which does not overlap with both poison and enemy.

3 Results

3.1 Synapse

Before simulation of final project, the behavior of synapse were first observed. Since the synaptic conductance (g_{alpha}, g_{GABA_β}, g_{NMDA}) were directly used as synaptic current to the motor neuron, the synaptic conductance of each synapse were simulated. Result of simulation is present in figure 2. Code for the simulation is present in Appendix 2.

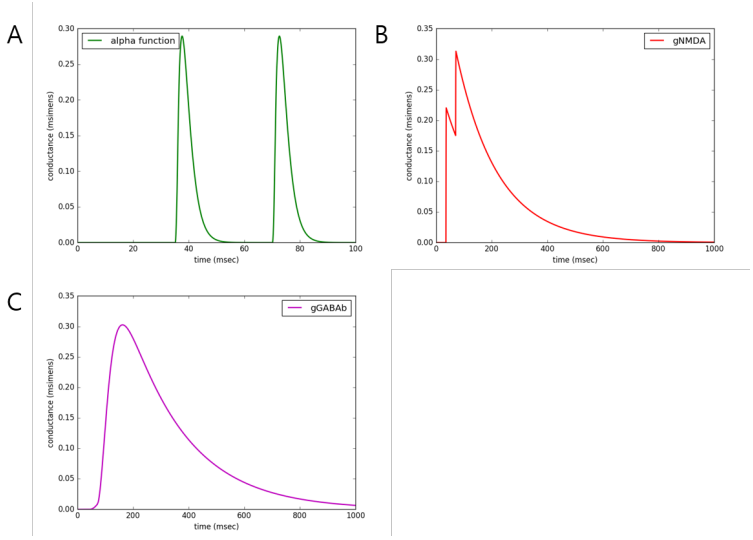

Figure 2: plot of conductance change of neurotransmitters and alpha function. A) g_{alpha}, B) g_{NMDA}, and C) g_{GABA_β}. One can observe that NMDA and $GABA_{beta}$ conductance decay slower than alpha synapse conductance

To test the behavior of the bug, a pseudo-bug simulation was conducted. The scenario was same as in Task 3. The bug encounter i) two food close to left or right sensor, ii)

poison , iii) mate, iv) and enemy in given order. The result of simulation can be found in figure 3. Code for the simulation is present in Appendix 3.

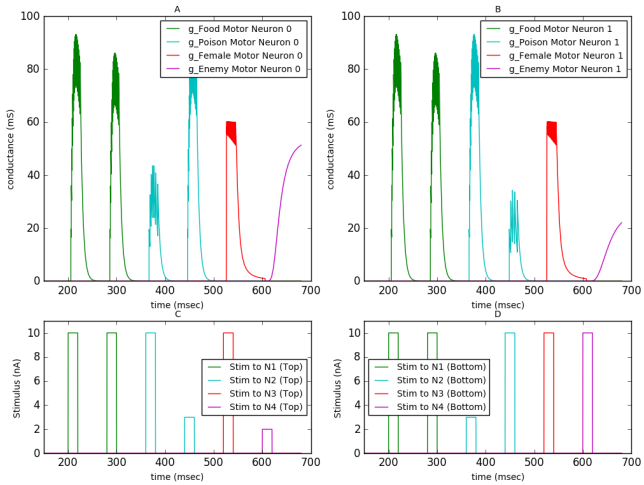

Figure 3: Result of simulation for artificial bug. The synaptic current to motor neuron for given stimulus is shown in (A) and (B) for right and left sensors respectively. The simulation to right and left sensor are shown in (C) and (D).

3.2 Bug Simulation

For simulation of artificial bug, the bug was put with four objects (food, poison, mate, and enemy) at given location. The scenario of experiment is same as in section **2.4 Experiment**. Both mate and enemy was coded to maneuver around the environment and make decision as aforementioned. Figure 4 is a snap shot of the experiment and Figure 5 shows a path way of bug and mate during the simulation.

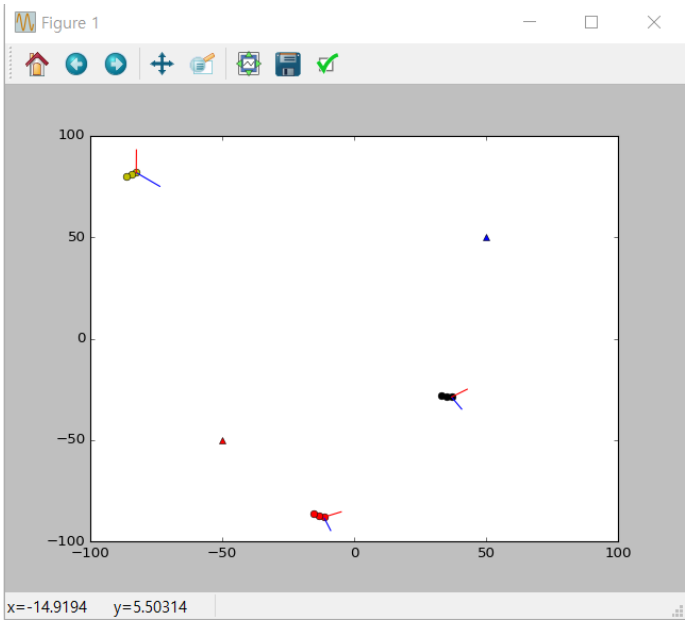

Figure 4: Snapshot of artificial bug simulation. The black, yellow and red bugs represent the artificial bug, mate and enemy respectively. The food was plotted as red triangle whereas poison was given as blue triangle.

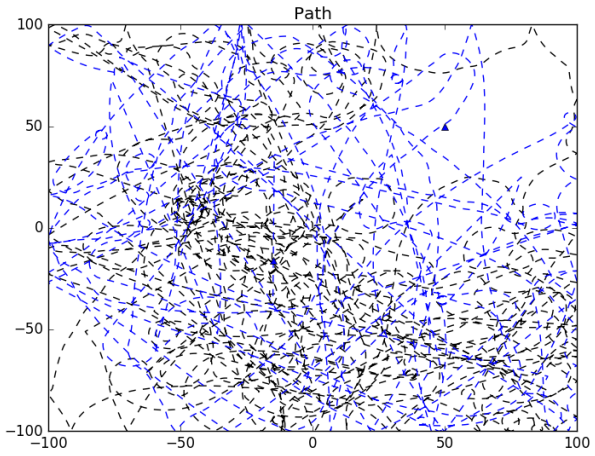

Figure 5: Pathway of bug and mate. Blue dotted line represents the path of mate and black dotted line shows the path of the artificial insect (bug). Due to complex decision making, the path is somewhat chaotic compared to when only a food or a poison is placed.

Summary of bug simulation result is in Table 3

Table 3: Summary of Bug Simulation Result

Predator catch number		Mating	Food Consumption	Poison Death
Bug	Mate			
10	3	4	28	0
Total Death of Bug	10			

For simulation of the artificial insect, please refer to supplementary material **Bug Simulation.mp4**.

4 DISCUSSION

Through out the project, the class explored the basic functions of **Brain2** and built their own artificial bug accordingly. The class simulated the integrate and fire (IAF) and HH type model for neurons and explored characteristics of Izhikevich model, which is both physiologically meaningful and computationally powerful, in task 2. In task 3, synapses between defined neurons were constructed using **Synapse** object of **Brain2**. The overall methods were integrated in the final project to build Braitenberg vehicle which shows a seemingly complex and somewhat intelligent behavior with simple neuromorphic circuits. Using the different characteristics of synaptic conductance, the vehicle was able to maneuver around the environment and make judgment of next behavior by interacting with other objects.

Before the actual simulation of artificial insect (Briatenberg vehicle), the author first explored the characteristics of synaptic conductances and their supposed behavior for given circuit design. As presented in **Result** section, the $GABA_{\beta}$ and NMDA conductance showed prolonged decay whereas alpha function conductance showed fast rise and fall. The artificial insect was supposed to continuously monitor the location of food and poison. The fast rise and fall of alpha function matched with the need by adopting to the new location and tracking the relative location of both objects by updating the stimulus fast. For mating, the author implemented NMDA as the receptor is related to LTP based learning [15]. The to sharp rise and slow decay of NMDA, a temporal summation of signals from the receptor was more likely than the others. Also the abrupt change from rising to decaying phase was expected to introduce a jittery movement which replicates the "shyness" of mating. The LTP for this synapse was implemented by incrementing the maximum conductance of the receptor by 10% for every success of mating. This was physiologically meaningful as post-synaptic depolarization should precede pre-synaptic release of transmitter; the event should occur while the synapse is activated [15]. The $GABA_{\beta}$, which showed prolonged decay was used for synapse between enemy sensor neuron and motor neuron. Although $GABA_{\beta}$ is used as inhibitory synapse in nature, it was used a excitatory to enable bug to flee from the enemy for enough time and get

away from it as far as possible. The pseudo-simulation result showed a promising result by showing how the neuro-circuit will response to the environmental factors.

The simulation of insect showed how Briatenberg vehicle can make complex decision with only limited neuro-circuits. The insect was able to mate and eat food and escape from food and poison with only 10 neurons (8 sensors and 2 motor neurons) and 8 synapses. Although the insect got captured by enemy multiple times, there was clear reason for this. First of all, because of the limited space of environment, bug seldom stuck in corner which enabled to enemy bug to capture its prey more easily (3 min 58 sec in **Bug Simulation.mp4**). Also the single food also increased the risk factor as well. As the bug gets close to the food, the mate followed the bug for mating. Therefore, when the enemy was in close distance to the food, all it had to do is wait for both prey to get close enough for it to eat either of them. Regarding the limited design of environment, the insect made great strategy and showed great success in food consumption (28 times) and mating (4 times) as in Table 3. Moreover, the LTP learning implemented for mating made the insect and its mate to success in mating in later part of the simulation more often as intended; modify strategy based on learning for higher success.

However, there were some limits as well. First of all, due to complex scenario (3 moving objects, 2 stationary objects), the insect showed confusing decision when multiple objects were adjacent to it. This problem could be modified by carefully weighing each factors or giving weights (maximum value of g) relative to the distance between object and the insect. The escaping strategy of the insect from enemy (predator) was not efficient compared to other decision making process. When the enemy's relative location to left and right sensor did not differ much, the firing rate and peak conductance of $GABA_\beta$ did not differ much for both sensors. Therefore, when the enemy is in such position, either mate or bug was not able to escape from the efficiently. Also the slow rising time of $GABA_\beta$ made the bug to wait for certain time to gather enough speed. When the bug recognized the enemy at close range, this gave advantage for the enemy to catch its prey (4 min 20 sec in **Bug Simulation.mp4**) since the enemy regarded prey as food and alpha function was used to model its according synapse. Due to the effect, $GABA_\beta$ receptor, which is G-protein coupled receptor, is used to derive prolonged physiological effect rather than rapid adaptation []. This was also seen in the result of simulation by comparing the number of food consumption and number of death of the bug. The bug was able to rapidly modify its path when food or poison was imminent and make efficient decision. However, the slow rising phase of $GABA_\beta$ was somewhat bothering for the insect to correct is pathway immediately.

This report has discussed the application of task1 through task3 for building a Braitenberg vehicle which maneuver around the given environment and set optimal strategy for survival. Braitenberg vehicle, which is an autonomous agents with simple neuron circuits can behave to show complex movement by adjusting to the environmental factors. Although the bug showed some jittery movement due to complex computation, the bug showed great survival rate. Further research for optimizing weights for each objects will make the survival rate of artificial bug higher.

5 REFERENCE

[1] Golgi, Camillo. "The neuron doctrine: theory and facts." Nobel lecture 1921 (1906): 190-217.

[2] Hodgkin, Alan L., and Andrew F. Huxley. "Propagation of electrical signals along giant nerve fibres." Proceedings of the Royal Society of London. Series B, Biological Sciences (1952): 177-183.

[3] Izhikevich, Eugene M. "Simple model of spiking neurons." IEEE Transactions on neural networks 14.6 (2003): 1569-1572.

[4] Indiveri, Giacomo, et al. "Neuromorphic silicon neuron circuits." Frontiers in neuroscience 5 (2011): 73.

[5] Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural Networks 61 (2015): 85-117.

[6] Ali, Mohsin M., Kristin K. Sellers, and Flavio Fröhlich. "Transcranial alternating current stimulation modulates large-scale cortical network activity by network resonance." The Journal of Neuroscience 33.27 (2013): 11262-11275.

[7] Braitenberg, Valentino. Vehicles: Experiments in synthetic psychology. MIT press, 1986.

[8] Beer, Randall D., Hillel J. Chiel, and Leon S. Sterling. "An artificial insect." American Scientist 79.5 (1991): 444-452.

[9] Salomon, Ralf. "Evolving and optimizing Braitenberg vehicles by means of evolution strategies." Int. J. Smart Eng. Syst. Design 2.1 (1999): 69-77.

[10] Brian authors. Brian 2 documentation . 2016(September 13, 2016), 548.

[11] Henriquez, George. “Lecture 5. Synapses.” BME 503. Duke University. 14 Sep. 2016.

[12] Izhikevich, Eugene M. "Which model to use for cortical spiking neurons?." IEEE transactions on neural networks 15.5 (2004): 1063-1070.

[13] Roth, Arnd, and Mark CW van Rossum. "6Modeling Synapses." (2009).

[14] Purves, Dale, et al. "Neuroscience." Sunderland, MA: Sinauer Associates (2001).

[15] Saal, Daniel, et al. "Drugs of abuse and stress trigger a common synaptic adaptation in dopamine neurons." Neuron 37.4 (2003): 577-582.

[16] Curtis, D. R., and G. Lacey. "Prolonged GABAB receptor-mediated synaptic inhibition in the cat spinal cord: an in vivo study." Experimental brain research 121.3 (1998): 319-333.

6 Appendix

6.1 Appendix1: Code for Artificial Bug Simulation

```
# import libraries/packages
from brian2 import *
import matplotlib.pyplot as plt

#-----------------------------------------------------
# set map size (given in skeleton code)
map_size = 100
# set global variables
global poisonx, poisony, poison_plot, poison_count
global bug_plot, sr_plot, sl_plot
global foodx, foody, food_count, food_plot
global mate_plot, sr_mate_plot, sl_mate_plot, mate_count
global enemy_plot, sr_enemy_plot, sl_enemy_plot, enemy_count
global death_count

#-----------------------------------------------------
#set coordinates for four objects
# 1. poison
poisonx = 50
poisony = 50

# 2.food
foodx = -50
foody = -50
food_count = 0

# 3.others
mate_count = 0
death_count = 0
poison_count = 0
enemy_count = 0

#-----------------------------------------------------
# set bug sensor
# Sensor neurons = Izhichevich model
a = 0.02; b = 0.2;
c = -65; d = 0.5
```

```
# The virtual bug
taum = 4*ms
base_speed = 9.5
turn_rate = 5*Hz

# NMDA (mate) neurotransmitter variables
Mg = 2 # unit mM. For ease of calculation, we do not give unit for this
       # found in Chapter 8 of "Foundations of Neuroscience" p. 163
betaN = 0.0066/ms
alphaN = 0.072/ms
v_th = 0*mV # reversal potential
transwdth = 1*ms

I0 = 100
g_synpk = 0.12*nsiemens # peak synaptic conductance
# maximum synaptic conductance
g_synmax = g_synpk/(taum/ms*exp(-1))
Cm=1.0*ufarad/cm**2
area=20000*umetre**2

# GABA_b synaptic neuotransimtters
K3 = 0.18/ms
K4 = 0.034/ms
Kd = 100 # found in Chapter 8 of "Foundations of Neuroscience" in p.164
betab = 0.0012/ms
alphab = 0.09/ms
n = 4 # power of GABAb s
adjust = 50

#-----------------------------------------------------
# bug motor neurons
# 1/2. poison/food = alpha synapse
# 3. mate = NMDA synapse
# 4. enemy = GABAb type excitatory synapse
bug_eqs = '''

# motor neurons
motorr=  (20*g_ampa2+10*g_ampa3+10*g_ampa5+20*g_ampa8)/nS: 1
motorl = (20*g_ampa1+10*g_ampa4+10*g_ampa6+20*g_ampa7)/nS: 1

# speed
speed = (motorl + motorr)/2 + base_speed : 1

# angle and displacement calculation
```

```
dangle/dt = (motorr - motorl)*turn_rate : 1
dx/dt = speed*cos(angle)*15*Hz : 1
dy/dt = speed*sin(angle)*15*Hz : 1

#-----------------------------------------------------
# for I1 (right side sensors)
# poison right side (g_ampa1)
dg_ampa1/dt=-g_ampa1/taum + z1/ms: siemens
dz1/dt = -z1/taum: siemens

# food right side (g_ampa3)
dg_ampa3/dt=-g_ampa3/taum + z3/ms: siemens
dz3/dt = -z3/taum: siemens

# mate right side (g_ampa5)
g_ampa5 = 8000*g*B5*sN5*bb5 : siemens
dsN5/dt = alphaN*(1-sN5)*Tr_pre5 - betaN*sN5 : 1
B5 = 1/(1+exp(-vN5/(16.13*mV))*(Mg/3.57)) : 1
dvN5/dt = g_ampa5*(vN5+60*mV)/(Cm*area) :  volt

Tr_pre5=10*(tanh((t/ms-tspike5/ms)/.005) -
            tanh((t/ms-(tspike5/ms +transwdth/ms))/.005)):1
tspike5:second
bb5:1

# enemy right side (g_ampa7)
g_ampa7 = (g_synmax*2000)*(sb7**n)/(sb7**n + Kd) : siemens
dr7/dt = (alphab*Tr_pre_GABA7*(1-r7) - betab*r7) : 1
dsb7/dt = (K3*r7 - K4*sb7) : 1

Tr_pre_GABA7=aa7*(tanh((t/ms-tspikeN7/ms)/(.005*adjust))-
            tanh((t/ms-(tspikeN7/ms +transwdth/
            (adjust*ms)))/(.005*adjust))):1
tspikeN7:second
aa7:1

#-----------------------------------------------------
# for I2 (left side sensors)
# poison left side (g_ampa2)
dg_ampa2/dt=-g_ampa2/taum + z2/ms: siemens
dz2/dt = -z2/taum: siemens

# food left side (g_ampa4)
dg_ampa4/dt=-g_ampa4/taum + z4/ms: siemens
```

```
dz4/dt = -z4/taum: siemens

# mate left side (g_ampa6)
g_ampa6 = 8000*g*B6*sN6*bb6 : siemens
dsN6/dt = alphaN*(1-sN6)*Tr_pre6 - betaN*sN6 : 1
B6 = 1/(1+exp(-vN6/(16.13*mV))*(Mg/3.57)) : 1
dvN6/dt = g_ampa6*(vN6+60*mV)/(Cm*area) :  volt

Tr_pre6=10*(tanh((t/ms-tspike6/ms)/.005)-
            tanh((t/ms-(tspike6/ms +transwdth/ms))/.005)):1
tspike6:second
bb6:1
g: siemens

# enemy left side (g_ampa8)
g_ampa8 = (g_synmax*2000)*(sb8**n)/(sb8**n + Kd) : siemens
dr8/dt = (alphab*Tr_pre_GABA8*(1-r8) - betab*r8) : 1
dsb8/dt = (K3*r8 - K4*sb8) : 1

Tr_pre_GABA8=aa7*(tanh((t/ms-tspikeN8/ms)/(.005*adjust))-
            tanh((t/ms-(tspikeN8/ms +transwdth/
            (adjust*ms)))/(.005*adjust))):1
tspikeN8:second
aa8:1

'''

bug = NeuronGroup(1, bug_eqs, clock=Clock(0.2*ms))
bug.angle = pi/2
bug.x = 0
bug.y = 0
bug.g = g_synmax

#-----------------------------------------------------
# mate bug equation and neuron
mate_eqs = '''

# motor neurons
motorr=  (10*g_ampa5 + 20*g_ampa8)/nS: 1
motorl = (10*g_ampa6 + 20*g_ampa7)/nS: 1

# speed
speed = (motorl + motorr)/2 + base_speed : 1
```

```
# angle and displacement calculation
dangle/dt = (motorr - motorl)*turn_rate : 1
dx/dt = speed*cos(angle)*15*Hz : 1
dy/dt = speed*sin(angle)*15*Hz : 1

# mate right side (g_ampa5)
g_ampa5 = 8000*g*B5*sN5*bb5 : siemens
dsN5/dt = alphaN*(1-sN5)*Tr_pre5 - betaN*sN5 : 1
B5 = 1/(1+exp(-vN5/(16.13*mV))*(Mg/3.57)) : 1
dvN5/dt = g_ampa5*(vN5+60*mV)/(Cm*area) : volt

Tr_pre5=10*(tanh((t/ms-tspike5/ms)/.005) -
            tanh((t/ms-(tspike5/ms +transwdth/ms))/.005)):1
tspike5:second
bb5:1

# enemy right side (g_ampa7)
g_ampa7 = (g_synmax*2000)*(sb7**n)/(sb7**n + Kd) : siemens
dr7/dt = (alphab*Tr_pre_GABA7*(1-r7) - betab*r7) : 1
dsb7/dt = (K3*r7 - K4*sb7) : 1

Tr_pre_GABA7=aa7*(tanh((t/ms-tspikeN7/ms)/(.005*adjust))-
            tanh((t/ms-(tspikeN7/ms +transwdth/
            (adjust*ms)))/(.005*adjust))):1
tspikeN7:second
aa7:1

# mate left side (g_ampa6)
g_ampa6 = 8000*g*B6*sN6*bb6 : siemens
dsN6/dt = alphaN*(1-sN6)*Tr_pre6 - betaN*sN6 : 1
B6 = 1/(1+exp(-vN6/(16.13*mV))*(Mg/3.57)) : 1
dvN6/dt = g_ampa6*(vN6+60*mV)/(Cm*area) : volt

Tr_pre6=10*(tanh((t/ms-tspike6/ms)/.005) -
            tanh((t/ms-(tspike6/ms +transwdth/ms))/.005)):1
tspike6:second
bb6:1
g:siemens

g_ampa8 = (g_synmax*2000)*(sb8**n)/(sb8**n + Kd) : siemens
dr8/dt = (alphab*Tr_pre_GABA8*(1-r8) - betab*r8) : 1
dsb8/dt = (K3*r8 - K4*sb8) : 1

Tr_pre_GABA8=aa7*(tanh((t/ms-tspikeN8/ms)/(.005*adjust))-
```

```
                tanh((t/ms-(tspikeN8/ms +transwdth/
                (adjust*ms)))/(.005*adjust))):1
tspikeN8:second
aa8:1

'''

mate = NeuronGroup(1, mate_eqs, clock=Clock(0.2*ms))
mate.angle = pi/2
mate.x = -75
mate.y = 75
mate.g = g_synmax

#-----------------------------------------------------
# enemy bug
# regard female and bug as both food
enemy_eqs = '''

# motor neurons
motorr=  10*(g_ampa1+g_ampa3)/nS: 1
motorl = 10*(g_ampa2+g_ampa4)/nS: 1

# speed
speed = (motorl + motorr)/2 + base_speed : 1

# angle and displacement calculation
dangle/dt = (motorr - motorl)*turn_rate : 1
dx/dt = speed*cos(angle)*15*Hz : 1
dy/dt = speed*sin(angle)*15*Hz : 1

#-----------------------------------------------------
# for I1 (right side sensors)

# food1 = bug right side (g_ampa3)
dg_ampa3/dt=-g_ampa3/taum + z3/ms: siemens
dz3/dt = -z3/taum: siemens

# food2 = female right side (g_ampa1)
dg_ampa1/dt=-g_ampa1/taum + z1/ms: siemens
dz1/dt = -z1/taum: siemens

#-----------------------------------------------------
# for I2 (left side sensors)
```

```
# food1 = bug left side (g_ampa4)
dg_ampa4/dt=-g_ampa4/taum + z4/ms: siemens
dz4/dt = -z4/taum: siemens

# food2 = female left side (g_ampa2)
dg_ampa2/dt=-g_ampa2/taum + z2/ms: siemens
dz2/dt = -z2/taum: siemens

'''
enemy = NeuronGroup(1, enemy_eqs, clock=Clock(0.2*ms))
enemy.angle = 3*pi/2
enemy.x = -100
enemy.y = -100

#-----------------------------------------------
# sensor (antenna) = Izhikevich
# 1. poison
sensor_eqs = '''
# Izhikevich + alpha function
dv/dt = (0.04*(v)*(v) + 5*(v) + 140- u)/ms
        + (I/mV)/(Cm*area): 1
du/dt = a*(b*v - u)/ms : 1

# stimulus from the objects to sensor
I = I0*nA/
    (sqrt((x-poisonx)**2 + (y-poisony)**2)): amp

# bug position
x : 1
y : 1
x_disp : 1
y_disp : 1

# poison coordinates
poisonx : 1
poisony : 1
'''

# 2. food
sensor_eqs2 = '''
# Izhikevich + alpha function
dv/dt = (0.04*(v)*(v) + 5*(v) + 140- u)/ms
        + (I/mV)/(Cm*area): 1
du/dt = a*(b*v - u)/ms : 1
```

```
# stimulus from the objects to sensor
I = I0*nA/
    (sqrt((x-foodx)**2 + (y-foody)**2)): amp

# bug position
x : 1
y : 1
x_disp : 1
y_disp : 1

# poison coordinates
foodx : 1
foody : 1
'''

# 3. mate
sensor_eqs3 = '''
# Izhikevich + alpha function
dv/dt = (0.04*(v)*(v) + 5*(v) + 140- u)/ms
        + (I/mV)/(Cm*area): 1
du/dt = a*(b*v - u)/ms : 1

# stimulus from the objects to sensor
I = 3*I0*nA/
    sqrt((x-matex)**2 + (y-matey)**2): amp

# bug position
x : 1
y : 1
x_disp : 1
y_disp : 1

# poison coordinates
matex : 1
matey : 1
'''

# 4. Enemy (predator)
sensor_eqs4 = '''
# Izhikevich + alpha function
dv/dt = (0.04*(v)*(v) + 5*(v) + 140- u)/ms
        + (I/mV)/(Cm*area): 1
du/dt = a*(b*v - u)/ms : 1
```

```
# stimulus from the objects to sensor
I = I0*nA/
    sqrt((x-enemyx)**2 + (y-enemyy)**2): amp

# bug position
x : 1
y : 1
x_disp : 1
y_disp : 1

# poison coordinates
enemyx : 1
enemyy : 1
'''

#-----------------------------------------------------
# for each neuron group you will also need to
# initialize parameters associated with membrane model,
# and assign thresholds and resets as needed- sveral
# neurons can be in each group

# set neuron for right side sensor
# 1. poison
sr_poison = NeuronGroup(1, sensor_eqs,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sr_poison.x_disp = 5; sr_poison.v = c;
sr_poison.y_disp = 5
sr_poison.x = sr_poison.x_disp
sr_poison.y = sr_poison.y_disp
sr_poison.poisonx = poisonx
sr_poison.poisony = poisony
# 2. food
sr_food = NeuronGroup(1, sensor_eqs2,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sr_food.x_disp = 5; sr_food.v = c;
```

```
sr_food.y_disp = 5
sr_food.x = sr_food.x_disp
sr_food.y = sr_food.y_disp
sr_food.foodx = foodx
sr_food.foody = foody
# 3. mate
sr_mate = NeuronGroup(1, sensor_eqs3,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sr_mate.x_disp = 10; sr_mate.v = c;
sr_mate.y_disp = 5
sr_mate.x = sr_mate.x_disp
sr_mate.y = sr_mate.y_disp
sr_mate.matex = mate.x
sr_mate.matey = mate.y

# 4. enemy
sr_enemy = NeuronGroup(1, sensor_eqs4,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sr_enemy.x_disp = 10; sr_enemy.v = c;
sr_enemy.y_disp = 5
sr_enemy.x = sr_enemy.x_disp
sr_enemy.y = sr_enemy.y_disp
sr_enemy.enemyx = enemy.x
sr_enemy.enemyy = enemy.y


# set neuron for left side sensor
# 1. poison
sl_poison = NeuronGroup(1, sensor_eqs,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sl_poison.x_disp = -5
sl_poison.y_disp = 5
```

```
sl_poison.x = sl_poison.x_disp; sl_poison.v = c;
sl_poison.y = sl_poison.y_disp
sl_poison.poisonx = poisonx
sl_poison.poisony = poisony
# 2. food
sl_food = NeuronGroup(1, sensor_eqs2,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sl_food.x_disp = -5
sl_food.y_disp = 5
sl_food.x = sl_food.x_disp; sl_food.v = c;
sl_food.y = sl_food.y_disp
sl_food.foodx = foodx
sl_food.foody = foody

# 3. mate
sl_mate = NeuronGroup(1, sensor_eqs3,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sl_mate.x_disp = -10
sl_mate.y_disp = 5
sl_mate.x = sl_mate.x_disp; sl_mate.v = c;
sl_mate.y = sl_mate.y_disp
sl_mate.matex = mate.x
sl_mate.matey = mate.y

# 4. enemy
sl_enemy = NeuronGroup(1, sensor_eqs4,
                      threshold='v > 0',
                      reset = 'v = c; u = u + d',
                      refractory='v > -60',
                      clock=Clock(defaultclock.dt),
                      method='euler')
sl_enemy.x_disp = -10
sl_enemy.y_disp = 5
sl_enemy.x = sl_enemy.x_disp; sl_enemy.v = c;
sl_enemy.y = sl_enemy.y_disp
sl_enemy.enemyx = enemy.x
```

```
sl_enemy.enemyy = enemy.y


#-----------------------------------------------------
# the mate bug
# only consist of enemy and mate

sr_m = NeuronGroup(1, sensor_eqs3,
                   threshold='v > 0',
                   reset = 'v = c; u = u + d',
                   refractory='v > -60',
                   clock=Clock(defaultclock.dt),
                   method='euler')
sr_m.x_disp = 10; sr_m.v = c;
sr_m.y_disp = 5
sr_m.x = sr_m.x_disp
sr_m.y = sr_m.y_disp
sr_m.matex = bug.x
sr_m.matey = bug.y

sr_m_enemy = NeuronGroup(1, sensor_eqs4,
                   threshold='v > 0',
                   reset = 'v = c; u = u + d',
                   refractory='v > -60',
                   clock=Clock(defaultclock.dt),
                   method='euler')
sr_m_enemy.x_disp = 10; sr_m_enemy.v = c;
sr_m_enemy.y_disp = 5
sr_m_enemy.x = sr_m_enemy.x_disp
sr_m_enemy.y = sr_m_enemy.y_disp
sr_m_enemy.enemyx = enemy.x
sr_m_enemy.enemyy = enemy.y

sl_m = NeuronGroup(1, sensor_eqs3,
                   threshold='v > 0',
                   reset = 'v = c; u = u + d',
                   refractory='v > -60',
                   clock=Clock(defaultclock.dt),
                   method='euler')
sl_m.x_disp = -10
sl_m.y_disp = 5
sl_m.x = sl_m.x_disp; sl_m.v = c;
sl_m.y = sl_m.y_disp
sl_m.matex = bug.x
```

```
sl_m.matey = bug.y

sl_m_enemy = NeuronGroup(1, sensor_eqs4,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sl_m_enemy.x_disp = -10
sl_m_enemy.y_disp = 5
sl_m_enemy.x = sl_m_enemy.x_disp; sl_m_enemy.v = c;
sl_m_enemy.y = sl_m_enemy.y_disp
sl_m_enemy.enemyx = enemy.x
sl_m_enemy.enemyy = enemy.y

#-----------------------------------------------------
# the enemy
# regard both bug (food1) and female (food2) as food

# right side
# food 1 = bug
sr_enemy_food1 = NeuronGroup(1, sensor_eqs2,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sr_enemy_food1.x_disp = 5; sr_enemy_food1.v = c;
sr_enemy_food1.y_disp = 5
sr_enemy_food1.x = sr_enemy_food1.x_disp
sr_enemy_food1.y = sr_enemy_food1.y_disp
sr_enemy_food1.foodx = bug.x
sr_enemy_food1.foody = bug.y
# food 2 = female
sr_enemy_food2 = NeuronGroup(1, sensor_eqs2,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sr_enemy_food2.x_disp = 5; sr_enemy_food2.v = c;
sr_enemy_food2.y_disp = 5
sr_enemy_food2.x = sr_enemy_food2.x_disp
sr_enemy_food2.y = sr_enemy_food2.y_disp
```

```
sr_enemy_food2.foodx = mate.x
sr_enemy_food2.foody = mate.y

# left side
# food 1 = bug
sl_enemy_food1 = NeuronGroup(1, sensor_eqs2,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sl_enemy_food1.x_disp = -5; sl_enemy_food1.v = c;
sl_enemy_food1.y_disp = 5
sl_enemy_food1.x = sl_enemy_food1.x_disp
sl_enemy_food1.y = sl_enemy_food1.y_disp
sl_enemy_food1.foodx = bug.x
sl_enemy_food1.foody = bug.y
# food 2 = female
sl_enemy_food2 = NeuronGroup(1, sensor_eqs2,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),
                    method='euler')
sl_enemy_food2.x_disp = -5; sl_enemy_food2.v = c;
sl_enemy_food2.y_disp = 5
sl_enemy_food2.x = sl_enemy_food2.x_disp
sl_enemy_food2.y = sl_enemy_food2.y_disp
sl_enemy_food2.foodx = mate.x
sl_enemy_food2.foody = mate.y

#-----------------------------------------------------
# Synapses (sensors communicate with bug motor)
# right side
# 1. poison
# 2. food
# 3. mate
# 4. enemy
syn_r = Synapses(sr_poison, bug,
                 clock=sr_poison.clock,model='''
                 w : siemens
                 ''',
                 pre='''
                 g_ampa2 += w
```

```
                '''')
syn_r.connect(i=[0],j=[0])
syn_r.w = g_synmax/(defaultclock.dt/ms)*10

syn_rf = Synapses(sr_food, bug,
                clock=sr_food.clock,model='''
                w : siemens
                ''',
                pre='''
                g_ampa4 += w
                '''')
syn_rf.connect(i=[0],j=[0])
syn_rf.w = g_synmax/(defaultclock.dt/ms)*10

syn_rm = Synapses(sr_mate, bug,
                clock=sr_mate.clock,
                pre='''
                tspike6 =t
                bb6 = 1
                vN6 = -65*mV
                ''')
syn_rm.connect(i=[0],j=[0])

syn_re = Synapses(sr_enemy, bug, clock=sr_enemy.clock,
                pre='''
                tspikeN = t
                aa8 = 200
                ''')
syn_re.connect(i=[0],j=[0])


# left side
# 1. poison
# 2. food
# 3. mate
syn_l = Synapses(sl_poison, bug,
                clock=sl_poison.clock,model='''
                w : siemens
                ''',
                pre='''
                g_ampa1 += w
                ''')
syn_l.connect(i=[0],j=[0])
syn_l.w = g_synmax/(defaultclock.dt/ms)*10
```

```
syn_lf = Synapses(sl_food, bug,
                  clock=sr_food.clock,model='''
                  w : siemens
                  ''',
                  pre='''
                  g_ampa3 += w
                  ''')
syn_lf.connect(i=[0],j=[0])
syn_lf.w = g_synmax/(defaultclock.dt/ms)*10

syn_lm = Synapses(sl_mate, bug,
                  clock=sr_mate.clock,
                  pre='''
                  tspike5 =t
                  bb5 = 1
                  vN5 = -65*mV

                  ''')
syn_lm.connect(i=[0],j=[0])

syn_le = Synapses(sl_enemy, bug, clock=sl_enemy.clock,
                  pre='''
                  tspikeN = t
                  aa7 = 200
                  ''')
syn_le.connect(i=[0],j=[0])

#-----------------------------------------------------------------------
# Mate bug synapse
syn_mate_rm = Synapses(sr_m, mate,
                  clock=sr_mate.clock,
                  pre='''
                  tspike6 =t
                  bb6 = 1
                  vN6 = -65*mV
                  ''')
syn_mate_rm.connect(i=[0],j=[0])

syn_re = Synapses(sr_enemy, mate, clock=sr_enemy.clock,
                  pre='''
                  tspikeN = t
                  aa8 = 200
                  ''')
```

```
syn_re.connect(i=[0],j=[0])

syn_mate_lm = Synapses(sl_m, mate,
                clock=sr_mate.clock,
                pre='''
                tspike5 =t
                bb5 = 1
                vN5 = -65*mV
                ''')
syn_mate_lm.connect(i=[0],j=[0])

syn_mate_le = Synapses(sl_enemy, mate,
                clock=sl_enemy.clock,
                pre='''
                tspikeN = t
                aa7 = 200
                ''')
syn_mate_le.connect(i=[0],j=[0])

#-----------------------------------------------------------------
# enemy bug synapse
syn_enemy_rf1 = Synapses(sr_enemy_food1, enemy,
                clock=sr_enemy_food1.clock,model='''
                w : siemens
                ''',
                pre='''
                g_ampa4 += w
                ''')
syn_enemy_rf1.connect(i=[0],j=[0])
syn_enemy_rf1.w = g_synmax/(defaultclock.dt/ms)*8

syn_enemy_rf2 = Synapses(sr_enemy_food2, enemy,
                clock=sr_enemy_food2.clock,model='''
                w : siemens
                ''',
                pre='''
                g_ampa2 += w
                ''')
syn_enemy_rf2.connect(i=[0],j=[0])
syn_enemy_rf2.w = g_synmax/(defaultclock.dt/ms)*5

syn_enemy_lf1 = Synapses(sl_enemy_food1, enemy,
                clock=sl_enemy_food1.clock,model='''
                w : siemens
```

```
                    ''',
                    pre='''
                    g_ampa3 += w
                    ''')
syn_enemy_lf1.connect(i=[0],j=[0])
syn_enemy_lf1.w = g_synmax/(defaultclock.dt/ms)*8

syn_enemy_lf2 = Synapses(sl_enemy_food2, enemy,
                    clock=sl_enemy_food2.clock,model='''
                    w : siemens
                    ''',
                    pre='''
                    g_ampa1 += w
                    ''')
syn_enemy_lf2.connect(i=[0],j=[0])
syn_enemy_lf2.w = g_synmax/(defaultclock.dt/ms)*5

#-----------------------------------------------------------------------
# plot

f = figure(1)
bug_plot = plot(bug.x, bug.y, 'ko')
poison_plot = plot(poisonx, poisony, 'b^')
food_plot = plot(foodx, foody, 'r^')
mate_plot = plot(mate.x, mate.y, 'yo')
enemy_plot = plot(enemy.x, enemy.y, 'ro')

sr_plot = plot([0], [0], 'w')   # Just leaving it blank for now
sl_plot = plot([0], [0], 'w')
sr_mate_plot = plot([0], [0], 'w')
sl_mate_plot = plot([0], [0], 'w')
sr_enemy_plot = plot([0], [0], 'w')
sl_enemy_plot = plot([0], [0], 'w')

# array of x and y
x_plot = []; y_plot = []
x_mate_plot = []; y_mate_plot = []
x_enemy_plot = []; y_enemy_plot = []

# This block of code updates the position of the bug and
# poison and makes the appropriate rotations
@network_operation()
def update_positions():
    global poisonx, poisony, poison_count
```

```
    global foodx, foody, food_count
    global matex, matey, mate_count
    global death_count, enemy_count


# <bug command>
# 1. poison position
    sr_poison.x = bug.x + sr_poison.x_disp*cos(
                  bug.angle-pi/2
                  ) - sr_poison.y_disp*sin(
                  bug.angle-pi/2)
    sr_poison.y = bug.y + sr_poison.x_disp*sin(
                  bug.angle-pi/2
                  ) + sr_poison.y_disp*cos(
                  bug.angle-pi/2)

    sl_poison.x = bug.x + sl_poison.x_disp*cos(
                  bug.angle-pi/2
                  ) - sl_poison.y_disp*sin(
                  bug.angle-pi/2)
    sl_poison.y = bug.y + sl_poison.x_disp*sin(
                  bug.angle-pi/2
                  ) + sl_poison.y_disp*cos(
                  bug.angle-pi/2)

# 2. food position
    sr_food.x = bug.x + sr_food.x_disp*cos(
                  bug.angle-pi/2
                  ) - sr_food.y_disp*sin(
                  bug.angle-pi/2)
    sr_food.y = bug.y + sr_food.x_disp*sin(
                  bug.angle-pi/2
                  ) + sr_food.y_disp*cos(
                  bug.angle-pi/2)

    sl_food.x = bug.x + sl_food.x_disp*cos(
                  bug.angle-pi/2
                  ) - sl_food.y_disp*sin(
                  bug.angle-pi/2)
    sl_food.y = bug.y + sl_food.x_disp*sin(
                  bug.angle-pi/2
                  ) + sl_food.y_disp*cos(
                  bug.angle-pi/2)
```

```
# 3. mate sensor position
    sr_mate.x = bug.x + sr_mate.x_disp*cos(
                    bug.angle-pi/2
                    ) - sr_mate.y_disp*sin(
                    bug.angle-pi/2)
    sr_mate.y = bug.y + sr_mate.x_disp*sin(
                    bug.angle-pi/2
                    ) + sr_mate.y_disp*cos(
                    bug.angle-pi/2)

    sl_mate.x = bug.x + sl_mate.x_disp*cos(
                    bug.angle-pi/2
                    ) - sl_mate.y_disp*sin(
                    bug.angle-pi/2)
    sl_mate.y = bug.y + sl_mate.x_disp*sin(
                    bug.angle-pi/2
                    ) + sl_mate.y_disp*cos(
                    bug.angle-pi/2)

# 4. enemy sensor position
    sr_enemy.x = bug.x + sr_enemy.x_disp*cos(
                    bug.angle-pi/2
                    ) - sr_enemy.y_disp*sin(
                    bug.angle-pi/2)
    sr_enemy.y = bug.y + sr_enemy.x_disp*sin(
                    bug.angle-pi/2
                    ) + sr_enemy.y_disp*cos(
                    bug.angle-pi/2)

    sl_enemy.x = bug.x + sl_enemy.x_disp*cos(
                    bug.angle-pi/2
                    ) - sl_enemy.y_disp*sin(
                    bug.angle-pi/2)
    sl_enemy.y = bug.y + sl_enemy.x_disp*sin(
                    bug.angle-pi/2
                    ) + sl_enemy.y_disp*cos(
                    bug.angle-pi/2)


# < mate command>
# 1. Mate position
    sr_m.x = mate.x + sr_m.x_disp*cos(
                    mate.angle-pi/2
                    ) - sr_m.y_disp*sin(
```

```
                    mate.angle-pi/2)
    sr_m.y = mate.y + sr_m.x_disp*sin(
                    mate.angle-pi/2
                    ) + sr_m.y_disp*cos(
                    mate.angle-pi/2)

    sl_m.x = mate.x + sl_m.x_disp*cos(
                    mate.angle-pi/2
                    ) - sl_m.y_disp*sin(
                    mate.angle-pi/2)
    sl_m.y = mate.y + sl_m.x_disp*sin(
                    mate.angle-pi/2
                    ) + sl_m.y_disp*cos(
                    mate.angle-pi/2)

# 2. enemy position
    sr_m_enemy.x = mate.x + sr_m_enemy.x_disp*cos(
                    mate.angle-pi/2
                    ) - sr_m_enemy.y_disp*sin(
                    mate.angle-pi/2)
    sr_m_enemy.y = mate.y + sr_m_enemy.x_disp*sin(
                    mate.angle-pi/2
                    ) + sr_m_enemy.y_disp*cos(
                    mate.angle-pi/2)

    sl_m_enemy.x = mate.x + sl_m_enemy.x_disp*cos(
                    mate.angle-pi/2
                    ) - sl_m_enemy.y_disp*sin(
                    mate.angle-pi/2)
    sl_m_enemy.y = mate.y + sl_m_enemy.x_disp*sin(
                    mate.angle-pi/2
                    ) + sl_m_enemy.y_disp*cos(
                    mate.angle-pi/2)

# <enemy command>
    sr_enemy_food1.x = enemy.x + sr_enemy_food1.x_disp*cos(
                    enemy.angle-pi/2
                    ) - sr_enemy_food1.y_disp*sin(
                    enemy.angle-pi/2)
    sr_enemy_food1.y = enemy.y + sr_enemy_food1.x_disp*sin(
                    enemy.angle-pi/2
                    ) + sr_enemy_food1.y_disp*cos(
                    enemy.angle-pi/2)
```

```
    sl_enemy_food1.x = enemy.x + sl_enemy_food1.x_disp*cos(
                    enemy.angle-pi/2
                    ) - sl_enemy_food1.y_disp*sin(
                    enemy.angle-pi/2)
    sl_enemy_food1.y = enemy.y + sl_enemy_food1.x_disp*sin(
                    enemy.angle-pi/2
                    ) + sl_enemy_food1.y_disp*cos(
                    enemy.angle-pi/2)

    sr_enemy_food2.x = enemy.x + sr_enemy_food2.x_disp*cos(
                    enemy.angle-pi/2
                    ) - sr_enemy_food2.y_disp*sin(
                    enemy.angle-pi/2)
    sr_enemy_food2.y = enemy.y + sr_enemy_food2.x_disp*sin(
                    enemy.angle-pi/2
                    ) + sr_enemy_food2.y_disp*cos(
                    enemy.angle-pi/2)

    sl_enemy_food2.x = enemy.x + sl_enemy_food2.x_disp*cos(
                    enemy.angle-pi/2
                    ) - sl_enemy_food2.y_disp*sin(
                    enemy.angle-pi/2)
    sl_enemy_food2.y = enemy.y + sl_enemy_food2.x_disp*sin(
                    enemy.angle-pi/2
                    ) + sl_enemy_food2.y_disp*cos(
                    enemy.angle-pi/2)

# re-locate objects when encountered

    if ((bug.x-poisonx)**2+(bug.y-poisony)**2) < 10:
        print 'The bug ate poison and died'
        # refresh NMDA learning behavior
        bug.g = g_synmax

        # count death and poison
        death_count += 1
        poison_count += 1

        # relocate poison
        relocatex = True
        relocatey = True
        while relocatex:
        # relocate poison
            poisonx_candidate = randint(-map_size+10, map_size-10)
```

```
            if (abs(poisonx_candidate-foodx) > 30 and
               abs(poisonx_candidate-bug.x)>30):
               poisonx = poisonx_candidate
               relocatex = False
               print 'relocated poison in x-direction'
        while relocatey:
            poisony_candidate = randint(-map_size+10, map_size-10)
            if (abs(poisony_candidate-foody) > 30 and
               abs(poisony_candidate-bug.y)>30):
               poisony = poisony_candidate
               relocatey = False
               print 'relocated poison in y-direction'

    if ((mate.x-enemy.x)**2+(mate.y-enemy.y)**2) < 10:
        print 'The predator ate the mate'
        # refresh NMDA learning behavior
        mate.g = g_synmax

        # count enemy catch number
        enemy_count += 1

        # relocate mate
        relocatex = True
        relocatey = True
        while relocatex:
            matex_candidate = randint(-map_size+10, map_size-10)
            if (abs(matex_candidate-poisonx) > 30 and
                abs(matex_candidate-enemy.x)>30):
                mate.x = matex_candidate
                relocatex = False
                print 'relocated mate in x-direction'


        while relocatey:
            matey_candidate = randint(-map_size+10, map_size-10)
            if (abs(matey_candidate-poisony) > 30 and
               abs(matey_candidate-enemy.y)>30):
                mate.y = matey_candidate
                relocatey = False
                print 'relocated mate in y-direction'

    if ((bug.x-enemy.x)**2+(bug.y-enemy.y)**2) < 10:
        print 'The predator ate the bug'
        # refresh NMDA learning behavior
```

```
        bug.g += g_synmax*0.1 #LTP learning
        mate.g += g_synmax*0.1 #LTP learning

        # count death and enemy catch number
        death_count += 1
        enemy_count += 1

        # relocate mate
        relocatex = True
        relocatey = True
        while relocatex:
            bugx_candidate = randint(-map_size+10, map_size-10)
            if (abs(bugx_candidate-poisonx) > 30 and
               abs(bugx_candidate-enemy.x) > 30):
               bug.x = bugx_candidate
               relocatex = False
               print 'relocated bug in x-direction'

        while relocatey:
            bugy_candidate = randint(-map_size+10, map_size-10)
            if (abs(bugy_candidate-poisony) > 30 and
               abs(bugy_candidate-enemy.y) > 30):
               bug.y = bugy_candidate
               relocatey = False
               print 'relocated bug in y-direction'

    if ((bug.x-foodx)**2+(bug.y-foody)**2) < 10:
        food_count += 1

        relocatex = True
        relocatey = True
        while relocatex:
            foodx_candidate = randint(-map_size+10, map_size-10)
            if (abs(foodx_candidate-poisonx) > 30):
                foodx = foodx_candidate
                relocatex = False

        while relocatey:
            foody_candidate = randint(-map_size+10, map_size-10)
            if (abs(foody_candidate-poisony) > 30):
                foody = foody_candidate
                relocatey = False

    if ((bug.x-mate.x)**2+(bug.y-mate.y)**2) < 10:
```

```
            mate_count += 1
            relocatex = True
            relocatey = True
            bug.g +=
            while relocatex:
                matex_candidate = randint(-map_size+10, map_size-10)
                if (abs(matex_candidate-enemy.x) > 30):
                    mate.x = matex_candidate
                    relocatex = False

            while relocatey:
                matey_candidate = randint(-map_size+10, map_size-10)
                if (abs(matey_candidate-enemy.y) > 30):
                    mate.y = matey_candidate
                    relocatey = False

    # rebound when hit margin
        if (bug.x < -map_size):
            bug.x = -map_size
            bug.angle = pi - bug.angle
        if (bug.x > map_size):
            bug.x = map_size
            bug.angle = pi - bug.angle
        if (bug.y < -map_size):
            bug.y = -map_size
            bug.angle = -bug.angle
        if (bug.y > map_size):
            bug.y = map_size
            bug.angle = -bug.angle

        if (mate.x < -map_size):
            mate.x = -map_size
            mate.angle = pi - mate.angle
        if (mate.x > map_size):
            mate.x = map_size
            mate.angle = pi - mate.angle
        if (mate.y < -map_size):
            mate.y = -map_size
            mate.angle = -mate.angle
        if (mate.y > map_size):
            mate.y = map_size
            mate.angle = -mate.angle

        if (enemy.x < -map_size):
```

```
        enemy.x = -map_size
        enemy.angle = pi - enemy.angle
    if (enemy.x > map_size):
        enemy.x = map_size
        enemy.angle = pi - enemy.angle
    if (enemy.y < -map_size):
        enemy.y = -map_size
        enemy.angle = -enemy.angle
    if (enemy.y > map_size):
        enemy.y = map_size
        enemy.angle = -enemy.angle

    sr_poison.poisonx = poisonx
    sr_poison.poisony = poisony
    sl_poison.poisonx = poisonx
    sl_poison.poisony = poisony
    sr_food.foodx = foodx
    sr_food.foody = foody
    sl_food.foodx = foodx
    sl_food.foody = foody
    sr_mate.matex = mate.x
    sr_mate.matey = mate.y
    sl_mate.matex = mate.x
    sl_mate.matey = mate.y
    sr_enemy.enemyx = enemy.x
    sr_enemy.enemyy = enemy.y
    sl_enemy.enemyx = enemy.x
    sl_enemy.enemyy = enemy.y

    sr_m.matex = bug.x
    sr_m.matey = bug.y
    sl_m.matex = bug.x
    sl_m.matey = bug.y
    sr_m_enemy.enemyx = enemy.x
    sr_m_enemy.enemyy = enemy.y
    sl_m_enemy.enemyx = enemy.x
    sl_m_enemy.enemyy = enemy.y

    sr_enemy_food1.foodx = bug.x
    sr_enemy_food1.foody = bug.y
    sl_enemy_food1.foodx = bug.x
    sl_enemy_food1.foody = bug.y
    sr_enemy_food2.foodx = mate.x
    sr_enemy_food2.foody = mate.y
```

```
        sl_enemy_food2.foodx = mate.x
        sl_enemy_food2.foody = mate.y

    # trace the path of enemy, bug, mate
        x_plot.append(float(bug.x/1.0))
        y_plot.append(float(bug.y/1.0))

        x_mate_plot.append(float(mate.x/1.0))
        y_mate_plot.append(float(mate.y/1.0))

        x_enemy_plot.append(float(enemy.x/1.0))
        y_enemy_plot.append(float(enemy.y/1.0))

    # this block of code updates the plots so
    # you can see the bug and poison move
    @network_operation(dt=2*ms)
    def update_plot():
        global poisonx, poisony, poison_plot
        global bug_plot, sr_plot, sl_plot
        global mate_plot, sr_mate_plot, sl_mate_plot
        global enemy_plot, sr_enemy_plot, sl_enemy_plot
        global food_plot, foodx, foody

        bug_plot[0].remove()
        poison_plot[0].remove()
        food_plot[0].remove()
        mate_plot[0].remove()
        enemy_plot[0].remove()
        sr_plot[0].remove()
        sl_plot[0].remove()
        sr_mate_plot[0].remove()
        sl_mate_plot[0].remove()
        sr_enemy_plot[0].remove()
        sl_enemy_plot[0].remove()

        bug_x_coords = [bug.x, bug.x-
                        2*cos(bug.angle),
                        bug.x-4*cos(bug.angle)]
                        # ant-like body
        bug_y_coords = [bug.y, bug.y-
                        2*sin(bug.angle),
                        bug.y-4*sin(bug.angle)]

        mate_x_coords = [mate.x, mate.x-
```

```
                    2*cos(mate.angle),
                    mate.x-4*cos(mate.angle)]
                    # ant-like body
    mate_y_coords = [mate.y, mate.y-
                    2*sin(mate.angle),
                    mate.y-4*sin(mate.angle)]

    enemy_x_coords = [enemy.x, enemy.x-
                    2*cos(enemy.angle),
                    enemy.x-4*cos(enemy.angle)]
                    # ant-like body
    enemy_y_coords = [enemy.y, enemy.y-
                    2*sin(enemy.angle),
                    enemy.y-4*sin(enemy.angle)]

    # Plot the bug's current position
    bug_plot = plot(bug_x_coords, bug_y_coords, 'ko')
    mate_plot = plot(mate_x_coords, mate_y_coords, 'yo')
    enemy_plot = plot(enemy_x_coords, enemy_y_coords, 'ro')

    # draws sensors. Since no need to draw all four sensors
    # this part remains unchanged
    sr_plot = plot([bug.x, sr_poison.x], [bug.y, sr_poison.y], 'b')
    sl_plot = plot([bug.x, sl_poison.x], [bug.y, sl_poison.y], 'r')

    sr_mate_plot = plot([mate.x, sr_m.x], [mate.y, sr_m.y], 'b')
    sl_mate_plot = plot([mate.x, sl_m.x], [mate.y, sl_m.y], 'r')

    sr_enemy_plot = plot([enemy.x, sr_enemy_food1.x],
                         [enemy.y, sr_enemy_food1.y], 'b')
    sl_enemy_plot = plot([enemy.x, sl_enemy_food1.x],
                         [enemy.y, sl_enemy_food1.y], 'r')


    poison_plot = plot(poisonx, poisony, 'b^')
    food_plot = plot(foodx, foody, 'r^')

    axis([-100,100,-100,100])
    draw()
    #print "."
    pause(0.01)

run(3000*ms,report='text')
```

```
plt.clf()
plt.plot(x_plot, y_plot, 'k--')
plt.plot(x_mate_plot,y_mate_plot, 'b--')
plt.plot(poisonx, poisony, 'b^')
plt.plot(foodx, foody, 'b^')

axis([-100,100,-100,100])
title('Path')
show()
```

```
from brian2 import *
import numpy as np
from matplotlib import gridspec

# set variables
duration = 1000*ms # change to 1000ms for long GABAb, default = 100*ms
res = 0.01*ms # resolution of time for TimedArray function
G = 0*arange(0, duration/res)
G[3500] = 1.0 # time when event takes place (t = 3500*.01ms = 35ms)
G[7000] = 1.0 # time when event takes place (t = 3500*.01ms = 70ms)
I = TimedArray(G, dt = res) # stimulus current

v_th = 1.*mV # threshold voltage
Cm = 200.0*pfarad # Membrane capacitance
bgcurrent = 200*pA # External current
tau_m = 2*ms # time constant of membrane which is not crucial for this
             # problem
tau_ampa = 2*ms
tau_alpha = 2*ms # for single time-constant model. Default = tau_ampa
                 # use 100 ms for comparison with NMDA/GABAb

g_synpk = 0.3*nsiemens # peak synaptic conductance
transwdth = 1.0*ms
g_synmax = g_synpk/(tau_ampa/ms*exp(-1)) # maximum synaptic conductance

# define Synaptic neurotransmitter variables

VAMPA = 0.0*mV

# NMDA neurotransmitter variables
Mg = 2 # unit mM. For ease of calculation, we do not give unit for this
       # found in Chapter 8 of "Foundations of Neuroscience" p. 163
betaN = 0.0066/ms
alphaN = 0.072/ms
VNMDA = 0.0*mV # equilibrium potential for NMDA
Cm_NMDA = 1*ufarad*cm**-2 # HH model Cm for simulation of NMDA
area = 20000*umetre**2

# GABA_b synaptic neuotransimtters
K3 = 0.18/ms
K4 = 0.034/ms
Kd = 100 # found in Chapter 8 of "Foundations of Neuroscience" in p.164
```

```
betab = 0.0012/ms
alphab = 0.09/ms
n = 4 # power of GABAb s
adjust = 2 # adjustment of stimulus width
interval = 1*ms # interval between stimulus



# model for
# 1) simple exponential decay model
# 2) alpha function model with single time constant
eqs = '''
dv/dt=-v/tau_m + I(t)*mag: volt # do not touch this part
                                # this works as a switch to turn on 1m
                                # train stimulus input

# alpha function model
dg_alpha/dt = (-g_alpha/tau_ampa + z/ms) : siemens
dz/dt = -z/tau_ampa + g_synmax*(Tr_pre/0.5)/ms : siemens

mag: volt/second
'''

#
eqs_NMDA = '''
gNMDA = g_synmax*B*sN : siemens
dsN/dt = alphaN*(Tr_pre*30)*(1-sN) - betaN*sN : 1
B = 1/(1+exp(-vN/(16.13*mV))*(Mg/3.57)) : 1
dvN/dt = -v/(4*tau_m) + I(t)*mag+ gNMDA*(VNMDA-vN)/Cm :  volt


Tr_pre=.25*(tanh((t/ms-tspike/ms)/.005) -
            tanh((t/ms-(tspike/ms +transwdth/ms))/.005)):1
tspike:second

'''
eqs_GABAb = '''
# scale = 20
gGABAb = (g_synmax*20)*(sb**n)/(sb**n + Kd) : siemens
dr/dt = (alphab*Tr_pre_GABA*(1-r) - betab*r) : 1
dsb/dt = (K3*r - K4*sb) : 1

# for 8 spikes
Tr_pre_GABA=.25*(tanh((t/ms-tspike/(ms))/(.005*adjust))-
```

```
        tanh((t/ms-(tspike/ms +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+2*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+2*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+3*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+3*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+4*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+4*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+5*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+5*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+6*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+6*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust)))+
            .25*(tanh(((t)/ms-(tspike+7*interval)/(ms))/(.005*adjust))-
            tanh(((t)/ms-((tspike+7*interval)/ms
            +transwdth/(adjust*ms)))/(.005*adjust))):1
'''

# plot the Alpha function
eqs += (eqs_NMDA+eqs_GABAb)
neurons = NeuronGroup(1, model=eqs,
                      threshold='v > v_th',
                      reset='v = 0*mV; g = g_synpk; tspike=t',
                      refractory='0.3*ms')

neurons.mag=150.0*mV/ms
neurons.v=0.0*mV
neurons.tspike=-100.0*ms  #needed so a spike does not happen at time 0
# Comment these two lines out to see what happens without Synapses


M = StateMonitor(neurons,
                 ('g_alpha','gNMDA','gGABAb'),
                 record=True)
sm = SpikeMonitor(neurons)

run(duration)
```

```
# result of simulation

# plot of alpha function with 2 time constant
# use different peak time, tau_decay, tau_rise for each
# neurotransmitters
figure(1)
#plot(M.t/ms, M.g_alpha[0]/nsiemens, '-g', lw=2, label = 'alpha functio
#plot(M.t/ms, M.gNMDA[0]/nsiemens, '-r', lw=2, label = 'gNMDA')
plot(M.t/ms, M.gGABAb[0]/nsiemens, '-m', lw=2, label = 'gGABAb')
xlabel('time (msec)')
ylabel('conductance (msimens)')
legend(loc = 'best')
xlim(0,duration/ms)
show()
```

```
from brian2 import *
import numpy as np
from matplotlib import gridspec
import time
t0 = time.clock()

defaultclock.dt =0.01*ms
div=defaultclock.dt

El=-10*mV
gl=0.01*msiemens/cm**2 # start from 0.01 to 0.8
Cm=1.0*ufarad/cm**2
tau_ampa=5*ms # synaptic time constant = 5ms
area=20000*umetre**2

# Parameters of Izhikevich model
# LTS (0.02, 0.25, -65, 2)
a = 0.02; b = 0.25;
c = -65.0; d = 2.0;
g_synpk = 0.12*nsiemens # peak synaptic conductance
# maximum synaptic conductance
g_synmax = g_synpk/(tau_ampa/ms*exp(-1))

# NMDA neurotransmitter variables
Mg = 2 # unit mM. For ease of calculation, we do not give unit for this
       # found in Chapter 8 of "Foundations of Neuroscience" p. 163
betaN = 0.0066/ms
alphaN = 0.072/ms

v_th = 0*mV # reversal potential
transwdth = 1.0*ms

# GABA_b synaptic neuotransimtters
K3 = 0.18/ms
K4 = 0.034/ms
Kd = 100 # found in Chapter 8 of "Foundations of Neuroscience" in p.164
betab = 0.0012/ms
alphab = 0.09/ms
n = 4 # power of GABAb s
adjust = 100
```

```
eqs_Antenna = '''
# Izhikevich + alpha function
dv/dt = (0.04*(v)*(v) + 5*(v) + 140- u)/ms
                  + (I/area/mV)/Cm : 1
du/dt = a*(b*v - u)/ms : 1
I : amp
'''

eqs_motor = '''
# ANTENNA 1 (food agressor)
dg_ampaO/dt=-g_ampaO/tau_ampa + zO/ms: siemens
dzO/dt = -zO/tau_ampa: siemens

# ANTENNA 2 (positon coward)
dg_ampaT/dt=-g_ampaT/tau_ampa + zT/ms: siemens
dzT/dt = -zT/tau_ampa: siemens

# ANTENNA 3 (Female jitter)
gNMDA = g_synmax*B*sN*30000*bb : siemens
dsN/dt = alphaN*(1-sN)*Tr_pre - betaN*sN : 1
B = 1/(1+exp(-vN/(16.13*mV))*(Mg/3.57)) : 1
dvN/dt = gNMDA*(60*mV+vN)/(Cm*area) :  volt

Tr_pre=100*(tanh((t/ms-tspike/ms)/.005) -
             tanh((t/ms-(tspike/ms +transwdth/ms))/.005)):1

# ANTENNA 4 (Enemy GABAb excitatory)
gGABAb = (g_synmax*1000)*(sb**n)/(sb**n + Kd) : siemens
dr/dt = (alphab*Tr_pre_GABA*(1-r) - betab*r) : 1
dsb/dt = (K3*r - K4*sb) : 1

Tr_pre_GABA=aa*(tanh((t/ms-tspikeN/ms)/(.005*adjust))-
             tanh((t/ms-(tspikeN/ms +transwdth/
             (adjust*ms)))/(.005*adjust))):1

I: amp
tspike:second
tspikeN:second
aa:1
bb:1
'''

# Motor neurons
Motor1 = NeuronGroup(1,  eqs_motor,
```

```
                    clock=Clock(defaultclock.dt),method='euler')

Motor2 = NeuronGroup(1,  eqs_motor,
                    clock=Clock(defaultclock.dt),method='euler')


# Antenna neurons
Antenna1T = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

Antenna2T = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

Antenna3T = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')
Antenna4T = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

Antenna1B = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

Antenna2B = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

Antenna3B = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
```

```
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')
Antenna4B = NeuronGroup(1,  eqs_Antenna,
                    threshold='v > 0',
                    reset = 'v = c; u = u + d',
                    refractory='v > -60',
                    clock=Clock(defaultclock.dt),method='euler')

# initialize voltage/conductance
# motor neurons voltage
# motor neurons conductance
Motor1.g_ampaO= 0*nS;Motor1.g_ampaT= 0*nS;
Motor1.sN= 0;Motor1.r= 0;Motor1.sb= 0;
Motor1.aa = 0; Motor1.bb = 0;Motor1.vN = 0

Motor2.g_ampaO= 0*nS;Motor2.g_ampaT= 0*nS;
Motor2.sN= 0;Motor2.r= 0;Motor2.sb= 0;
Motor2.aa = 0; Motor2.bb = 0;Motor2.vN = 0

# antenna neurons
Antenna1T.v= c;Antenna2T.v= c;Antenna3T.v= c;Antenna4T.v= c;
Antenna1B.v= c;Antenna2B.v= c;Antenna3B.v= c;Antenna4B.v= c;

# connect neurons

# Antenna1T, Antenna2T, Antenna3B, Antenna4B to Motor1
# Antenna1T -> Motor 1
SrM1A1 = Synapses(Antenna1T, Motor1, clock=Antenna1T.clock,
                  model='''
                  w : siemens
                  ''',
                  pre='''
                  g_ampaO += w
                  ''')
SrM1A1.connect(i=[0],j=[0])
SrM1A1.w = g_synmax/(defaultclock.dt/ms)*3
SrM1A1.delay=5*ms
# Antenna2B -> Motor 1
SrM1A2 = Synapses(Antenna2B, Motor1, clock=Antenna2B.clock,
                  model='''
                  w : siemens
                  ''',
                  pre='''
```

```
                g_ampaT += w
                ''')
SrM1A2.connect(i=[0],j=[0])
SrM1A2.w = g_synmax/(defaultclock.dt/ms)*3
SrM1A2.delay=5*ms
# Antenna3T -> Motor 1
SrM1A3 = Synapses(Antenna3T, Motor1, clock=Antenna3T.clock,
                pre='''
                tspike =t
                bb = 1
                vN = -65*mV
                ''')
SrM1A3.connect(i=[0],j=[0])
SrM1A3.delay=5*ms
# Antenna4B -> Motor 1
SrM1A4 = Synapses(Antenna4B, Motor1, clock=Antenna4B.clock,
                pre='''
                tspikeN = t
                aa = 200
                bb = 0
                ''')
SrM1A4.connect(i=[0],j=[0])
SrM1A4.delay=5*ms

# Antenna1B, Antenna2B, Antenna3T, Antenna4T to Motor2
# Antenna1B -> Motor 2
SrM2A1 = Synapses(Antenna1B, Motor2, clock=Antenna1B.clock,
                model='''
                w : siemens
                ''',
                pre='''
                g_ampaO += w
                ''')
SrM2A1.connect(i=[0],j=[0])
SrM2A1.w = g_synmax/(defaultclock.dt/ms)*3
SrM2A1.delay=5*ms
# Antenna2T -> Motor 2
SrM2A2 = Synapses(Antenna2T, Motor2, clock=Antenna2T.clock,
                model='''
                w : siemens
                ''',
                pre='''
                g_ampaT += w
                ''')
```

```
SrM2A2.connect(i=[0],j=[0])
SrM2A2.w = g_synmax/(defaultclock.dt/ms)*3
SrM2A2.delay=5*ms
# Antenna3B -> Motor 2
SrM2A3 = Synapses(Antenna3B, Motor2, clock=Antenna3B.clock,
                  pre='''
                  tspike = t
                  bb = 1
                  vN = -65*mV
                  ''')
SrM2A3.connect(i=[0],j=[0])
SrM2A3.delay=5*ms
# Antenna4T -> Motor 2
SrM2A4 = Synapses(Antenna4T, Motor2, clock=Antenna4T.clock,
                  pre='''
                  tspikeN = t
                  aa = 200
                  bb = 0
                  ''')
SrM2A4.connect(i=[0],j=[0])
SrM2A4.delay=5*ms

# setup monitors
# motor neuron (voltage and conductance)
M1 = StateMonitor(Motor1,('g_ampaO','Tr_pre_GABA',
                  'g_ampaT', 'gNMDA','gGABAb'),record=True)
M2 = StateMonitor(Motor2,('g_ampaO', 'Tr_pre_GABA',
                  'g_ampaT', 'gNMDA','gGABAb'),record=True)

# Antenna on top (only voltage)
A1T = StateMonitor(Antenna1T, ('I'), record = True)
A2T = StateMonitor(Antenna2T, ('I'), record = True)
A3T = StateMonitor(Antenna3T, ('I'), record = True)
A4T = StateMonitor(Antenna4T, ('I'), record = True)

# Antenna on Bottom (only voltage)
A1B = StateMonitor(Antenna1B, ('I'), record = True)
A2B = StateMonitor(Antenna2B, ('I'), record = True)
A3B = StateMonitor(Antenna3B, ('I'), record = True)
A4B = StateMonitor(Antenna4B, ('I'), record = True)

# give stimulus to antenna accordingly to the
# scenairo
# run 200 msec for stablizing
```

```
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(200*ms,report='text')

# encounter 1st food
Antenna1T.I=10*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=10*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(60*ms,report='text')

# encounter 2nd food
Antenna1T.I=10*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=10*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(60*ms,report='text')

# encounter 1st poison
Antenna1T.I=0*nA; Antenna2T.I=10*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=3*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
```

```
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(60*ms,report='text')

# encounter 2nd poison
Antenna1T.I=0*nA; Antenna2T.I=3*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=10*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(60*ms,report='text')

# encounter female
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=10*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=10*nA;
Antenna4B.I=0*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
run(60*ms,report='text')

# encounter enemy
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=2*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=10*nA;
run(20*ms,report='text')

# interval
Antenna1T.I=0*nA; Antenna2T.I=0*nA; Antenna3T.I=0*nA;
Antenna4T.I=0*nA;
Antenna1B.I=0*nA; Antenna2B.I=0*nA; Antenna3B.I=0*nA;
Antenna4B.I=0*nA;
```

```
run(60*ms,report='text')

figure(1)
gs = gridspec.GridSpec(2,2, height_ratios = [2,1])
# conductance of motor 1
subplot(gs[0,0])
plot(M1.t/ms, M1.g_ampaO[0]/nS, '-g', lw=1,
            label = 'g_Food Motor Neuron 0')
plot(M1.t/ms, M1.g_ampaT[0]/nS, '-c', lw=1,
            label = 'g_Poison Motor Neuron 0')
plot(M1.t/ms, M1.gNMDA[0]/nS, '-r', lw=1,
            label = 'g_Female Motor Neuron 0')
plot(M1.t/ms, M1.gGABAb[0]/nS, '-m', lw=1,
            label = 'g_Enemy Motor Neuron 0')
xlabel('time (msec)', fontsize = 10)
ylabel('conductance (mS)', fontsize = 10)
xlim(150, 700); legend(loc = 0, prop={'size':10});
title('A', fontsize = 10)

# stimulus to antennas (top)
subplot(gs[1,0])
plot(A1T.t/ms, A1T.I[0]/nA, '-g', lw=1,
            label = 'Stim to N1 (Top)')
plot(A2T.t/ms, A2T.I[0]/nA, '-c', lw=1,
            label = 'Stim to N2 (Top)')
plot(A3T.t/ms, A3T.I[0]/nA, '-r', lw=1,
            label = 'Stim to N3 (Top)')
plot(A4T.t/ms, A4T.I[0]/nA, '-m', lw=1,
            label = 'Stim to N4 (Top)')
xlabel('time (msec)', fontsize = 10)
ylabel('Stimulus (nA)', fontsize = 10)
xlim(150, 700); ylim(0,11)
legend(loc = 0, prop={'size':10})
title('C', fontsize = 10)

# conductance of motor 2
subplot(gs[0,1])
plot(M2.t/ms, M2.g_ampaO[0]/nS, '-g', lw=1,
            label = 'g_Food Motor Neuron 1')
plot(M2.t/ms, M2.g_ampaT[0]/nS, '-c', lw=1,
            label = 'g_Poison Motor Neuron 1')
plot(M2.t/ms, M2.gNMDA[0]/nS, '-r', lw=1,
            label = 'g_Female Motor Neuron 1')
plot(M2.t/ms, M2.gGABAb[0]/nS, '-m', lw=1,
```

```
            label = 'g_Enemy Motor Neuron 1')
xlabel('time (msec)', fontsize = 10)
ylabel('conductance (mS)', fontsize = 10)
xlim(150, 700); legend(loc = 0, prop={'size':10});
title('B', fontsize = 10)

# stimulus to antennas (Bottom)
subplot(gs[1,1])
plot(A1B.t/ms, A1B.I[0]/nA, '-g', lw=1,
            label = 'Stim to N1 (Bottom)')
plot(A2B.t/ms, A2B.I[0]/nA, '-c', lw=1,
            label = 'Stim to N2 (Bottom)')
plot(A3B.t/ms, A3B.I[0]/nA, '-r', lw=1,
            label = 'Stim to N3 (Bottom)')
plot(A4B.t/ms, A4B.I[0]/nA, '-m', lw=1,
            label = 'Stim to N4 (Bottom)')
xlabel('time (msec)', fontsize = 10)
ylabel('Stimulus (nA)', fontsize = 10)
xlim(150, 700); ylim(0,11)
legend(loc = 0, prop={'size':10})
title('D', fontsize = 10)

print time.clock() - t0, "seconds process time"
show()
```