Braitenbugs

Mary Zhang

Introduction

The study of Neurorobotics started in the late 1980s when a group of scientists built a series of robotic devices to test how the cerebellum adapts movements. Since then, the field has grown to include brain-inspired algorithms, computational models of biological neural networks and actual biological systems (e.g. in vivo and in vitro neural nets). These models and algorithms are then being implemented in robots, prosthetics, micro-machines, furniture and infrastructures among many others products to enhance their performance and functions.

History and implementation aside, one of the core parts of neurorobotics comes down to studying how simple nervous systems can create complex behaviors and adapt and interact with a changing environment, as all living beings do. This study requires looking at simple organisms - like in this case, a humble bug - and how to create artificial models to recreate their biological mechanisms resulting in mimicking their behaviors in real life.

A simple fruit fly has 100,000 neurons in their brain. A bumblebee has 1 million. The connection and signaling combinations of these neurons is what gives rise the vast range of behaviors a bug can exhibit in feeding, moving and mating while moving through their constantly changing environments. This project explores two variations of a bug model with four neurons and a changing food source.

Valentino Braitenberg's book "Vehicles" discussing the psychology around how humans can interpret "behaviors" in non living models by their motions. In this he proposes the law of uphill analysis and downhill invention which states "when we analyze a mechanism, we tend to overestimate its complexity". This leads to our ability of interpreting the simple actions of a simple bug model to hold complex human qualities. He goes on to call different models "The Agressor," "The Lover," "The Knower," and even "The Thinker" based on how they interact with stimuli. The range of behaviors my two bugs possess will explore the two types of interaction with a food source by modifying the parameters of their neurons. One is modeled to have the sole purpose of eating (the classic Aggressor), while the other seems to possess a little more hesitation and deliberation around the act of eating (The Indecisive Lover).

Methods

To build my bug I used the Izhikevich model of to simulate the neuronal spiking of the sensor neurons. This model is commonly used to model a range of bursting and spiking patterns of cortical neurons. The equation for the Izhikevich model is shown below in equations 1 and 2 with threshold and reset parameters shown in equation 3 and 4.

$$dv/dt = (0.04)(v^2) + (5v) + 140 - u + I) + g(E_{syn} - v) \tag{1}$$

$$du/dt = a((bv)-u) \tag{2}$$

$$v = c \tag{3}$$

$$u = u + d \tag{4}$$

The current/stimulus intensity to drive the change in voltage of the sensor neurons is dependent on the distance of sensor neurons from the food source as shown in the following equation.

$$I = I0 / sqrt((x - foodxx)^2 + (y - foodyy)^2) \tag{5}$$

After creating the spiking neurons for two front sensors and two back neurons, I connected the front sensor with the contralateral back neuron with the Synapse function block. Both front sensor neurons are excitatory, so the Nernst potential (E_{syn}) was set to 40 mV. When a presynaptic spike arrives at the synapse the conductance is triggered to a max conductance value (g_synmaxval) of 0.4. To mimic the effects of neurotransmitters to impact spiking, I used an alpha wave to modulate the increase of conductance, g. The increase in conductance to cause depolarization triggers the neuron to depolarize and spike, driving the motor velocity of the bug through the back neuron. The back neurons (sbr and sbl in the code) are then synapsed with one bug neuron that function as the motors to output motor velocity and position of the bug.

$$dg/dt = (-g/tau_{syn}) + z \tag{6}$$

$$dz/dt = -z/tau_{syn} \tag{7}$$

The set of equations that drive the movement of the bug are as follows: motorl and motorr are the individual speeds of the right and left motors. Vel gives the total velocity of the bug by combining the speeds of both motors. The angle of the bug is the direction the bug is facing with its sensor neurons. The angle is used to calculate x and y, the position of the bug in x and y coordinates.

$$dmotorl/dt = - motorl/taum \tag{8}$$

$$dmotorr/dt = - motorr/taum \tag{9}$$

$$vel = base\_speed + (motorl+motorr)/2 \tag{10}$$

$$dangle/dt = (motorr-motorl)turn\_rate \tag{11}$$

$$dx/dt = 0.02(vel * cos(angle)) \tag{12}$$

$$dy/dt = 0.02*(vel * sin(angle)) \tag{13}$$

Tau_m, base_speed and turn_rate are parameters for motor speed spike time constant, a baseline speed to allow the bug to fluctuate above and below for velocity, and the turning speed, respectively. These equations ultimately result in a plot on an x-y plane for where the bug and its food are located.

One model I did not explore in this bug model but is present in neuronal firing is spike-timing dependent plasticity (STDP). This modulation of change in synaptic strengths can be modeled by

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W(t_{ni} - t_{fj}) \tag{14}$$

And will be further discussed in the following section.

Results

The Izhikevich model of neurons is used to reproduce spiking and bursting behavior of a range of cortical neurons. (1) The model is a combination of the biological accuracy of the Hodgkin-Huxley and computational efficiency of integrate and fire neurons. A range of biological spiking patterns can be modeled with Izhikevich by changing the a, b, c and d constants in Equation 1. Figure 1 below is a plot of combinations of these constants and which models patterns they create. Figures 2 - 5 show 4 examples of different spiking patterns. These spiking patterns represent the signalling patterns of different structures in the brain. For example, the hippocampus fires low threshold spiking (LTS) pattern while the cortical layer 3 is found to fire in the chattering (CH) pattern. (2)

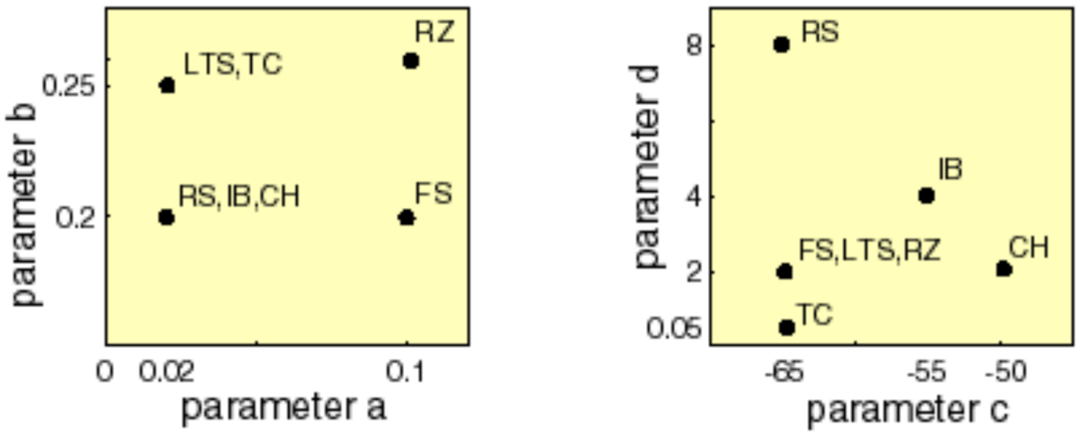

Figure 1: Combination plot of constants a, b, c and d.

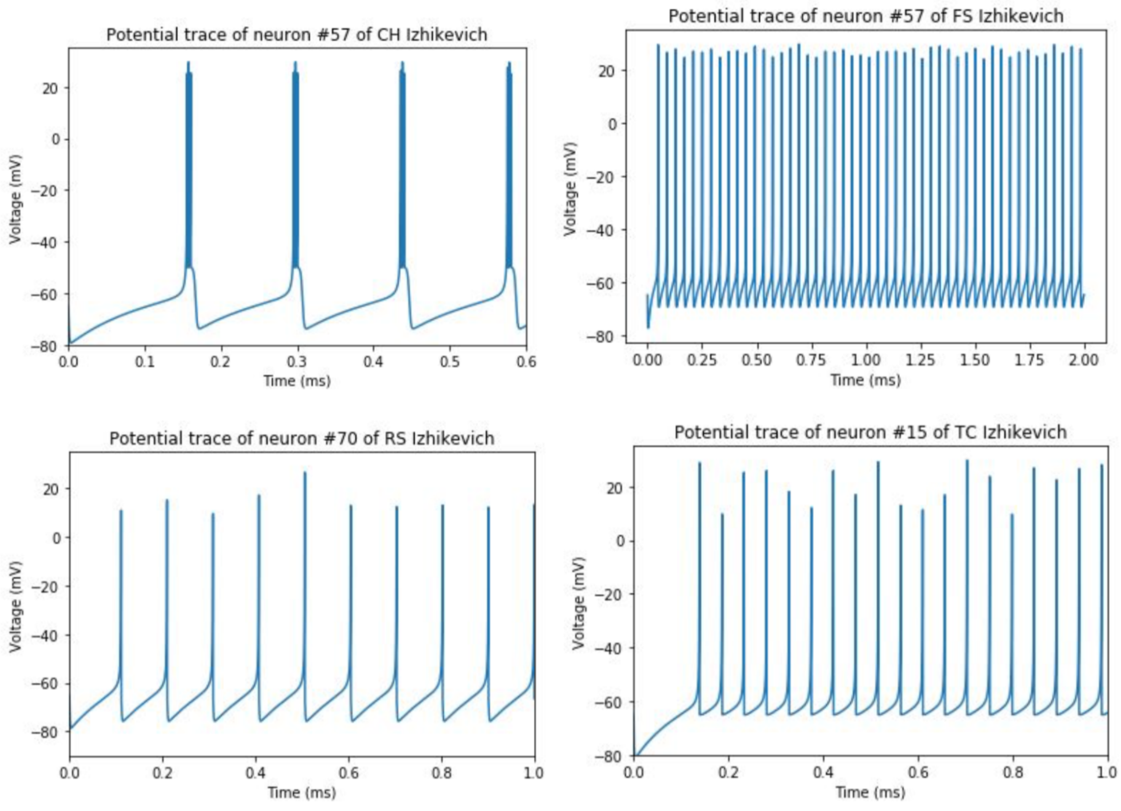

Figure 2: 4 bursting patterns of Izhikevich model: chatter, fast-spiking, regular spiking and thalamo-cortico.

After modeling individual neurons to produce spiking patterns, the next step is to create synapses between them. Synapses are chemical junctions between two neurons that allow information/signals to be transmitted. I explore two signalling effects in this bug modeling -- inhibitory signaling and excitatory signaling. Excitatory signaling creates a more more easily excitable neuron, one that has a higher probability of firing. This is done by increasing the Nernst potential (E_{syn}) closer to the activation threshold (30 mV), such as 40 mV. Doing so raises the membrane potential closer to threshold, decreasing the amount of additional positive ions needed to flow in to bring the membrane potential above 30 mV. Inhibitory signalling produces the opposite effect, it drives the membrane potential away from threshold by setting E_{syn} at a value below threshold, such as -80 mV. The effects of inhibition and excitation are shown in Figure 3 and 4.

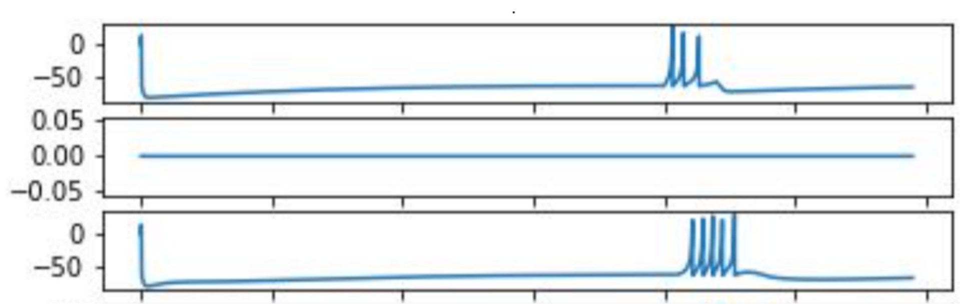

Figure 3: Excitatory signalling from the presynaptic neuron (1st graph) to the postsynaptic neuron (3rd graph.)

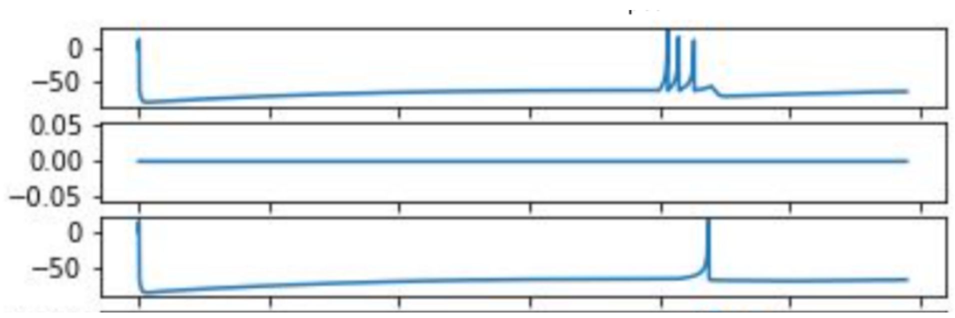

Figure 4: Inhibitory signalling from the presynaptic neuron (1st graph) to the postsynaptic neuron (3rd graph).

The signals to cause excitatory or inhibitory effects are carried on neurotransmitters. The signalling occurs in the following steps:

"The action potential travels down the axon and terminates at many presynaptic sites invading regions called synaptic terminals. These terminals contain calcium channels which when depolarized, cause (a) release of calcium; (b) calcium activates a calcium binding protein which promotes transmitter release by binding to vesicles containing the transmitter; (c) these "docked" vesicles release their transmitter into the synaptic cleft; (d) the transmitter diffuses through the cleft where it binds to various receptors on the postsynaptic neuron (often on protuberances on the dendrites called spines); (e) these receptors open channels which either depolarize or hyperpolarize the neuron depending on the nature of the transmitter." (3)

The occurences of these transmissions are modeled by a change in conductance, g(t), of the synaptic current, with E_{syn} signalling excitatory or inhibitory. At peak conductance, the maximum amount of ions are permeable to the neuron -- to either depolarize or hyperpolarize the membrane.

$$I_{syn} = g(t)(E_{syn} - V_m) \tag{15}$$

There are four main neurotransmitters we've explored to date. NMDA and AMPA are excitatory neurotransmitters and GABAa and GABAb are inhibitory. NMDA and GABAa are fast acting neurotransmitters, meaning that within milliseconds of binding to the receptor their excitatory/inhibitory effects occur. NMDA and GABAb are slow acting, so their effects do no occur immediately. In this bug model I use an alpha wave to model the conductance. The rise and decay times of the alpha wave can be modified to mimic the speed of the effects of the neurotransmitters as shown in the green (second row) plots in the figures below.

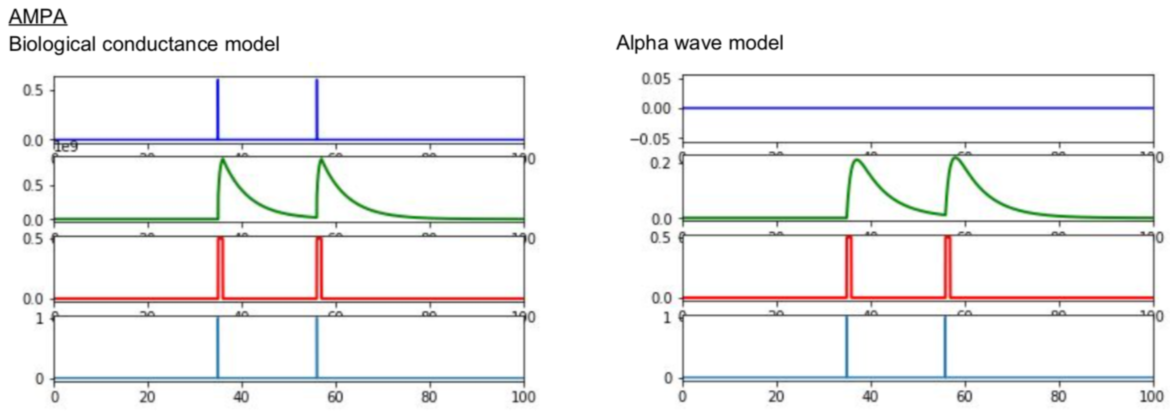

Figure 5: Biological model vs. alpha wave model of fast acting transmitter.

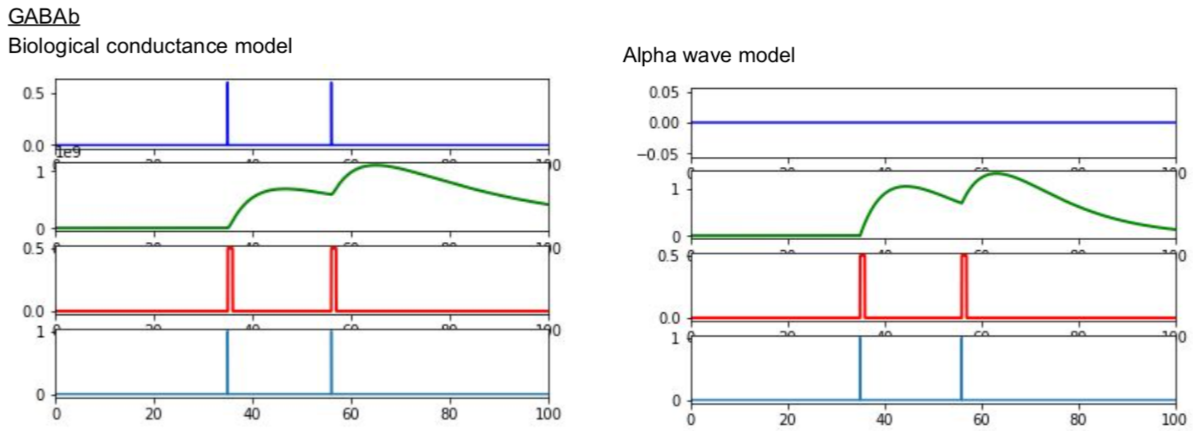

Figure 6: Biological model vs. alpha wave model of slow acting transmitter.

One feature of synaptic transmission that was not explored in this bug was spike-timing dependent plasticity (STDP). With STDP, recurring presynaptic signalling coupled with postsynaptic spiking occuring milliseconds afterwards leads to long term potentiation (LTP) of the synapses. This leads to an increase in synaptic strength, as the cells undergoing Hebbian learning of "firing together and wiring together". The strength of the synapses increase between neurons that fire in the correct order and with the shorter amount of delay between the two. The opposite also occurs when the presynaptic signal arrives after the postsynaptic spiking. This is long term depression (LTD). LTP and LTD are both methods of how synapse strengths (weights) can change depending on activation timing, as shown in the figure below.

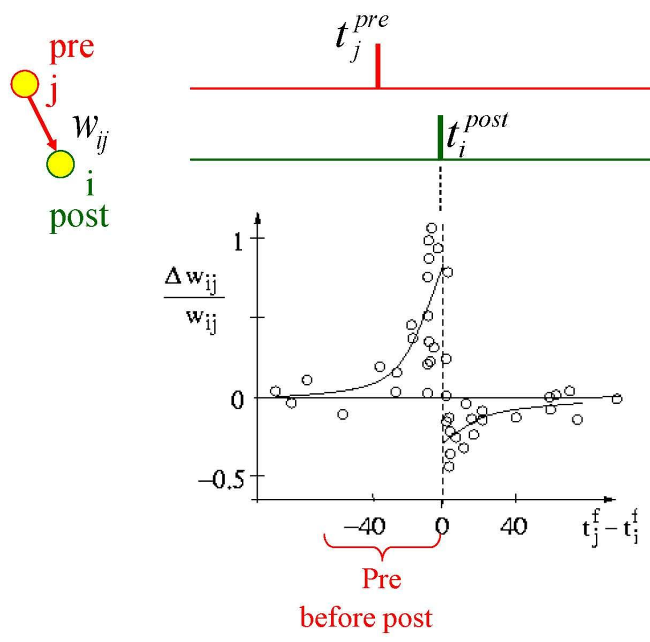

Figure 7: Plot of STDP with weights (synaptic strength) on y axis and timing between pre and post synpatic neuron on x axis.

Discussion

The Aggressor model currently only responds to food stimuli. To further develop the model to replicate a bug's actual behavior, I could increase the types of stimuli to include light, oxygen and temperature and include additional sensor neurons to to detect these stimuli and drive the motors. This would turn the Aggressor model into The Knower, a bug that can knows what the stimuli are and how it should respond to them in order to "survive". Avoidance of high temperature would require crossed inhibitory sensing, and if the bug were a moth it would require a circling motion around a light source. In building the simple Aggressor I learned the significance of proper sensor to motor neuron connection paths that dictate movement behavior. An excitatory contralateral crossing results in a movement towards the direction of the sensor. The combination of inhibition/excitation with the connection path would have to be mapped out in the building of The Knower because a bug would interact with the four stimuli in different ways: circling, preferring eating and avoiding, respectively. "Vehicles" mapped out an example of a Knower network in the following figure.

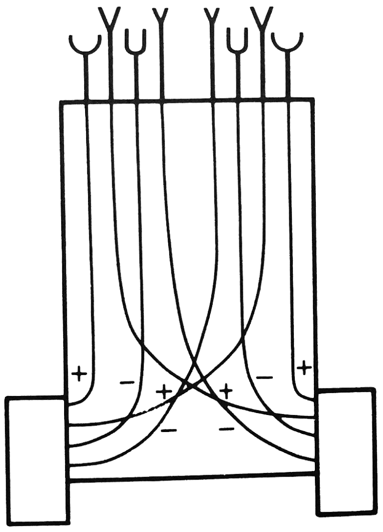

Figure 8: The Knower with multi sensory inputs.

A bug can also be modeled to demonstrate “preferences” and patterns towards inputs as shown in Figure 9.

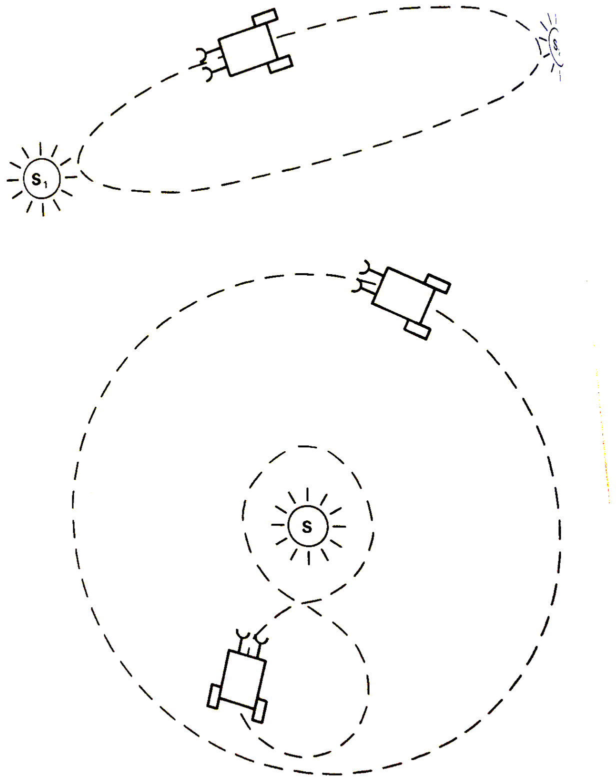

Figure 9: Patterns of behavior.

This idea of "pattern of behavior" led to my variation of the Indecisive Lover. I initially created the original Lover with inhibitory neurons and then decreased the motor time constant and increased the turn rate to create a lag in reaction when it approached the food source. This caused the bug to slow down as it approached the food and then swing around it and run away from it, before realizing that it had made a mistake and circle back around to approach the food again. This behavior of "realizing it made a mistake" comes from a simple in the motor reaction. The increase in turn rate would induce a rapid 360 degree turn around, coming across as a sudden realization of a mistake. It was also necessary to increase the base speed since the neurons are inhibitory and reduce motor drive.

From my two variations I learned that combining the structure of the network of sensors to motors, the number of stimuli and neuron parameters all create significant behavioral effects on a model of the bug.

References

1. https://www.izhikevich.org/publications/spikes.htm
2. http://www.scholarpedia.org/article/Bursting
3. Chapter 8 of "Foundations of Mathematical Neuroscience" by Bard Ermentrout and David Terman
4. http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity#STDP_and_Hebbian_learning_rules
5. “Vehicles” by Valentino Braitenberg

Appendix

Code for Aggressor bug

```
from brian2 import *
import matplotlib.pyplot as plt
map_size = 100
global foodx, foody, food_count, bug_plot, food_plot, sr_plot,
sl_plot,outbugx,outbugy,outbugang,outfoodx,outfoody,outsrx,outsry,outslx,outsly


food_count = 0
foodx=50
foody=50
duration=1000
outbugx=np.zeros(int(duration/2))
outbugy=np.zeros(int(duration/2))
outbugang=np.zeros(int(duration/2))
outfoodx=np.zeros(int(duration/2))
outfoody=np.zeros(int(duration/2))
outsrx=np.zeros(int(duration/2))
outsry=np.zeros(int(duration/2))
outslx=np.zeros(int(duration/2))
outsly=np.zeros(int(duration/2))

# Sensor neurons
a = 0.02
b = 0.2
c = -65
d = 0.5

I0 = 1250
E_syn = 40
tau_syn=1
g_synpk=0.4
g_synmaxval=(g_synpk/(tau_syn*exp(-1)))

sensor_eqs = '''

x : 1
y : 1
x_disp : 1
y_disp : 1
foodxx : 1
foodyy : 1
mag :1
I = I0 / sqrt(((x-foodxx)**2+(y-foodyy)**2)): 1
dv/dt = ((((0.04)*(v**2)) + (5*v) + 140 - u + I) + g*(E_syn-v) ) / ms : 1
du/dt = (a*((b*v)-u)) / ms  : 1

dg/dt = ((-g/tau_syn) + z) / ms : 1
dz/dt = ((-z/tau_syn)) / ms  : 1


'''

sensor_reset = '''
v = c
u = u + d
'''
```

```
sr = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sr.v = c
sr.u = c*b
sr.x_disp = 5
sr.y_disp = 5
sr.x = sr.x_disp
sr.y = sr.y_disp
sr.foodxx = foodx
sr.foodyy = foody
sr.mag=1

sl = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sl.v = c
sl.u = c*b
sl.x_disp = -5
sl.y_disp = 5
sl.x = sl.x_disp
sl.y = sl.y_disp
sl.foodxx = foodx
sl.foodyy = foody
sl.mag=1

sbr = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sbr.v = c
sbr.u = c*b
sbr.foodxx = foodx
sbr.foodyy = foody
sbr.mag=0

sbl = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sbl.v = c
sbl.u = c*b
sbl.foodxx = foodx
sbl.foodyy = foody
sbl.mag=0


# The virtual bug

taum = 4
base_speed = 30
turn_rate = 20*Hz

bug_eqs = '''
#equations for movement here
dmotorl/dt = -motorl/taum / ms : 1
dmotorr/dt = -motorr/taum / ms : 1
vel = base_speed+((motorl+motorr)/2) : 1
dangle/dt = (motorr-motorl)*turn_rate  : 1
dx/dt = 0.02*(vel*cos(angle)) / ms : 1
dy/dt = 0.02*(vel*sin(angle)) / ms : 1


'''

bug = NeuronGroup(1, bug_eqs, clock=Clock(0.2*ms),method='euler')
bug.motorl = 0
bug.motorr = 0
bug.angle = pi/3
bug.x = 0
bug.y = 0
```

```
# Synapses (sensors communicate with bug motor)
w = 10
syn_rr=Synapses(sr, sbl, clock=Clock(0.2*ms), model='''
                g_synmax:1
                ''',
                on_pre='''
                z+= g_synmax
                ''')




syn_rr.connect(i=[0],j=[0])
syn_rr.g_synmax=g_synmaxval

syn_ll=Synapses(sl, sbr, clock=Clock(0.2*ms), model='''
                g_synmax:1
                ''',
                on_pre='''
                z+= g_synmax
                ''')

syn_ll.connect(i=[0],j=[0])
syn_ll.g_synmax=g_synmaxval

syn_r = Synapses(sbr, bug, clock=Clock(0.2*ms), on_pre='motorr += w')
syn_r.connect(i=[0],j=[0])
syn_l = Synapses(sbl, bug, clock=Clock(0.2*ms), on_pre='motorl += w')
syn_l.connect(i=[0],j=[0])

f = figure(1)
bug_plot = plot(bug.x, bug.y, 'ko')
food_plot = plot(foodx, foody, 'b*')
sr_plot = plot([0], [0], 'w')   # Just leaving it blank for now
sl_plot = plot([0], [0], 'w')
# Additional update rules (not covered/possible in above eqns)

@network_operation()
def update_positions():
    global foodx, foody, food_count
    sr.x = bug.x + sr.x_disp*sin(bug.angle)+ sr.y_disp*cos(bug.angle)
    sr.y = bug.y + - sr.x_disp*cos(bug.angle) + sr.y_disp*sin(bug.angle)

    sl.x = bug.x +  sl.x_disp*sin(bug.angle)+sl.y_disp*cos(bug.angle)
    sl.y = bug.y  - sl.x_disp*cos(bug.angle)+sl.y_disp*sin(bug.angle)




    if ((bug.x-foodx)**2+(bug.y-foody)**2) < 16:
        food_count += 1
        foodx = randint(-map_size+10, map_size-10)
        foody = randint(-map_size+10, map_size-10)

    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

    sr.foodxx = foodx
    sr.foodyy = foody
    sl.foodxx = foodx
    sl.foodyy = foody

@network_operation(dt=2*ms)
def update_plot(t):
    global foodx, foody, bug_plot, food_plot, sr_plot, 
sl_plot,outbugx,outbugy,outbugang,outfoodx,outfoody,outsrx,outsry,outslx,outsly
    indx=int(.5*t/ms+1)
    bug_plot[0].remove()
    food_plot[0].remove()
    sr_plot[0].remove()
    sl_plot[0].remove()
    bug_x_coords = [bug.x, bug.x-4*cos(bug.angle), bug.x-8*cos(bug.angle)]    # ant-like body
    bug_y_coords = [bug.y, bug.y-4*sin(bug.angle), bug.y-8*sin(bug.angle)]
    outbugx[indx-1]=bug.x[0]
    outbugy[indx-1]=bug.y[0]
    outbugang[indx-1]=bug.angle[0]
    outfoodx[indx-1]=foodx
    outfoody[indx-1]=foody
    outsrx[indx-1]=sr.x[0]
    outsry[indx-1]=sr.y[0]
    outslx[indx-1]=sl.x[0]
    outsly[indx-1]=sl.y[0]
    bug_plot = plot(bug_x_coords, bug_y_coords, 'ko')     # Plot the bug's current position
    sr_plot = plot([bug.x, sr.x], [bug.y, sr.y], 'b')
    sl_plot = plot([bug.x, sl.x], [bug.y, sl.y], 'r')
    food_plot = plot(foodx, foody, 'b*')
    axis([-100,100,-100,100])
    draw()
    #print "."
    pause(0.01)

#ML = StateMonitor(sl, ('v', 'I'), record=True)
#MR = StateMonitor(sr, ('v', 'I'), record=True)
#MB = StateMonitor(bug, ('motorl', 'motorr', 'speed', 'angle', 'x', 'y'), record = True)
run(duration*ms,report='text')
np.save('outbugx',outbugx)
np.save('outbugy',outbugy)
np.save('outbugang',outbugang)
np.save('outfoodx',outfoodx)
np.save('outfoody',outfoody)
np.save('outsrx',outsrx)
np.save('outsry',outsry)
np.save('outslx',outslx)
np.save('outsly',outsly)
```

Code for Indecisive Lover bug

```
from brian2 import *
import matplotlib.pyplot as plt
map_size = 100
global foodx, foody, food_count, bug_plot, food_plot, sr_plot,
sl_plot,outbugx,outbugy,outbugang,outfoodx,outfoody,outsrx,outsry,outslx,outsly


food_count = 0
foodx=50
foody=50
duration=1000
outbugx=np.zeros(int(duration/2))
outbugy=np.zeros(int(duration/2))
outbugang=np.zeros(int(duration/2))
outfoodx=np.zeros(int(duration/2))
outfoody=np.zeros(int(duration/2))
outsrx=np.zeros(int(duration/2))
outsry=np.zeros(int(duration/2))
outslx=np.zeros(int(duration/2))
outsly=np.zeros(int(duration/2))

# Sensor neurons
a = 0.02
b = 0.2
c = -65
d = 0.5

I0 = 1250
E_syn = -80
tau_syn=1
g_synpk=0.7
g_synmaxval=(g_synpk/(tau_syn*exp(-1)))

sensor_eqs = '''

x : 1
y : 1
x_disp : 1
y_disp : 1
foodxx : 1
foodyy : 1
mag :1
I = I0 / sqrt(((x-foodxx)**2+(y-foodyy)**2)): 1
dv/dt = ((((0.04)*(v**2)) + (5*v) + 140 - u + I) + g*(E_syn-v) ) / ms : 1
du/dt = (a*((b*v)-u)) / ms  : 1

dg/dt = ((-g/tau_syn) + z) / ms : 1
dz/dt = ((-z/tau_syn)) / ms  : 1


'''

sensor_reset = '''
v = c
u = u + d
'''

sr = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sr.v = c
sr.u = c*b
sr.x_disp = 5
sr.y_disp = 5
sr.x = sr.x_disp
sr.y = sr.y_disp
sr.foodxx = foodx
sr.foodyy = foody
sr.mag=1

sl = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sl.v = c
sl.u = c*b
sl.x_disp = -5
sl.y_disp = 5
sl.x = sl.x_disp
sl.y = sl.y_disp
sl.foodxx = foodx
sl.foodyy = foody
sl.mag=1

sbr = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sbr.v = c
sbr.u = c*b
sbr.foodxx = foodx
sbr.foodyy = foody
sbr.mag=0

sbl = NeuronGroup(1, sensor_eqs, clock=Clock(0.2*ms), threshold = "v>=30", reset =
sensor_reset,method='euler')
sbl.v = c
sbl.u = c*b
sbl.foodxx = foodx
sbl.foodyy = foody
sbl.mag=0


# The virtual bug

taum = 5
base_speed = 50
turn_rate = 30*Hz

bug_eqs = '''
#equations for movement here

dmotorl/dt = -motorl/taum / ms : 1
dmotorr/dt = -motorr/taum / ms : 1
vel = base_speed+((motorl+motorr)/2) : 1
dangle/dt = (motorr-motorl)*turn_rate  : 1
dx/dt = 0.02*(vel*cos(angle)) / ms : 1
dy/dt = 0.02*(vel*sin(angle)) / ms : 1


'''

bug = NeuronGroup(1, bug_eqs, clock=Clock(0.2*ms),method='euler')
bug.motorl = 0
bug.motorr = 0
bug.angle = pi/3
bug.x = 0
bug.y = 0



# Synapses (sensors communicate with bug motor)
w = 10
```

```
syn_rr=Synapses(sr, sbr, clock=Clock(0.2*ms), model='''
                g_synmax:1
                ''',
                on_pre='''
                z+= g_synmax
                ''')




syn_rr.connect(i=[0],j=[0])
syn_rr.g_synmax=g_synmaxval

syn_ll=Synapses(sl, sbl, clock=Clock(0.2*ms), model='''
                g_synmax:1
                ''',
                on_pre='''
                z+= g_synmax
                ''')

syn_ll.connect(i=[0],j=[0])
syn_ll.g_synmax=g_synmaxval

syn_r = Synapses(sbr, bug, clock=Clock(0.2*ms), on_pre='motorr += w')
syn_r.connect(i=[0],j=[0])
syn_l = Synapses(sbl, bug, clock=Clock(0.2*ms), on_pre='motorl += w')
syn_l.connect(i=[0],j=[0])

f = figure(1)
bug_plot = plot(bug.x, bug.y, 'ko')
food_plot = plot(foodx, foody, 'b*')
sr_plot = plot([0], [0], 'w')   # Just leaving it blank for now
sl_plot = plot([0], [0], 'w')
# Additional update rules (not covered/possible in above eqns)

@network_operation()
def update_positions():
    global foodx, foody, food_count
    sr.x = bug.x + sr.x_disp*sin(bug.angle)+ sr.y_disp*cos(bug.angle)
    sr.y = bug.y + - sr.x_disp*cos(bug.angle) + sr.y_disp*sin(bug.angle)

    sl.x = bug.x +  sl.x_disp*sin(bug.angle)+sl.y_disp*cos(bug.angle)
    sl.y = bug.y  - sl.x_disp*cos(bug.angle)+sl.y_disp*sin(bug.angle)




    if ((bug.x-foodx)**2+(bug.y-foody)**2) < 16:
        food_count += 1
        foodx = randint(-map_size+10, map_size-10)
        foody = randint(-map_size+10, map_size-10)

    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

    sr.foodxx = foodx
    sr.foodyy = foody
    sl.foodxx = foodx
    sl.foodyy = foody

@network_operation(dt=2*ms)
def update_plot(t):
    global foodx, foody, bug_plot, food_plot, sr_plot,
sl_plot,outbugx,outbugy,outbugang,outfoodx,outfoody,outsrx,outsry,outslx,outsly
    indx=int(.5*t/ms+1)
    bug_plot[0].remove()
    food_plot[0].remove()
    sr_plot[0].remove()
    sl_plot[0].remove()
    bug_x_coords = [bug.x, bug.x-4*cos(bug.angle), bug.x-8*cos(bug.angle)]    # ant-like body
    bug_y_coords = [bug.y, bug.y-4*sin(bug.angle), bug.y-8*sin(bug.angle)]
    outbugx[indx-1]=bug.x[0]
    outbugy[indx-1]=bug.y[0]
    outbugang[indx-1]=bug.angle[0]
    outfoodx[indx-1]=foodx
    outfoody[indx-1]=foody
    outsrx[indx-1]=sr.x[0]
    outsry[indx-1]=sr.y[0]
    outslx[indx-1]=sl.x[0]
    outsly[indx-1]=sl.y[0]
    bug_plot = plot(bug_x_coords, bug_y_coords, 'ko')     # Plot the bug's current position
    sr_plot = plot([bug.x, sr.x], [bug.y, sr.y], 'b')
    sl_plot = plot([bug.x, sl.x], [bug.y, sl.y], 'r')
    food_plot = plot(foodx, foody, 'b*')
    axis([-100,100,-100,100])
    draw()
    #print "."
    pause(0.01)

#ML = StateMonitor(sl, ('v', 'I'), record=True)
#MR = StateMonitor(sr, ('v', 'I'), record=True)
#MB = StateMonitor(bug, ('motorl', 'motorr', 'speed', 'angle', 'x', 'y'), record = True)
run(duration*ms,report='text')
np.save('outbugx',outbugx)
np.save('outbugy',outbugy)
np.save('outbugang',outbugang)
np.save('outfoodx',outfoodx)
np.save('outfoody',outfoody)
np.save('outsrx',outsrx)
np.save('outsry',outsry)
np.save('outslx',outslx)
np.save('outsly',outsly)
```