Summer 2014

Neuromodulation Based Control of Autonomous Robots on a Cloud Computing Platform

Cameron Muhammad

NEUROMODULATION BASED CONTROL OF AUTONOMOUS ROBOTS ON A CLOUD COMPUTING PLATFORM

By

CAMERON MUHAMMAD

(Under the Direction of Biswanath Samanta)

ABSTRACT

In recent years, the advancement of neurobiologically plausible models and computer networking has resulted in new ways of implementing control systems on robotic platforms. The work presents a control approach based on vertebrate neuromodulation and its implementation on autonomous robots in the open-source, open-access environment of robot operating system (ROS). A spiking neural network (SNN) is used to model the neuromodulatory function for generating context based behavioral responses of the robots to sensory input signals. The neural network incorporates three types of neurons- cholinergic and noradrenergic (ACh/NE) neurons for attention focusing and action selection, dopaminergic (DA) neurons for rewards- and curiosity-seeking, and serotonergic (5-HT) neurons for risk aversion behaviors. This model depicts neuron activity that is biologically realistic but computationally efficient to allow for large-scale simulation of thousands of neurons. The model is implemented using graphics processing units (GPUs) for parallel computing in real-time using the ROS environment. The model is implemented to study the risk-taking, risk-aversive, and distracted behaviors of the neuromodulated robots in single- and multi-robot configurations. The entire process is implemented in a cloud computing environment using ROS where the robots communicate wirelessly with the computing nodes through the on-board laptops. However, unlike the traditional neural networks, the neuromodulatory models do not need any pre-training. Instead, the robots learn from the sensory inputs and follow the behavioral facets of living organisms. The details of algorithm development, the experimental setup and implementation results under different conditions, in both single- and multi-robot configurations, are presented along with a discussion on the scope of further work.

INDEX WORDS: Artificial neural networks, Cloud computing, Cloud Robotics, CUDA, GPU, Izhikevich Spiking Neuron, Neuromodulation, Neurorobotics, Parallel computing, Robot Operating System, ROS, Parallel processing, Spiking neural networks.

NEUROMODULATION BASED CONTROL OF AUTONOMOUS ROBOTS ON A CLOUD COMPUTING PLATFORM

By

CAMERON MUHAMMAD

B.S., Georgia Southern University, 2011

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial

Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

2014

NEUROMODULATION BASED CONTROL OF AUTONOMOUS ROBOTS ON A CLOUD COMPUTING PLATFORM

By

CAMERON MUHAMMAD

DEDICATION

To Mom, Ronelka, and Kayleigh for all of their support.

To Dad for always keeping my spirits up and showing what work ethic I needed to succeed in life. Rest in Peace.

ACKNOWLEDGEMENTS

I would like to thank Dr. Biswanath Samanta for his guidance and support over the past two years on this project.

I would also like to thank all of my family and friends at Georgia Southern University.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

5-HT – Serotonin

ACh – Acetylcholine

ANN – artificial neural network

CPU – central processing unit

CUDA – Compute Unified Device Architecture

DA - Dopamine

EPSP – excitatory post-synaptic potential

GPU – graphics processing unit

IPSP – inhibitory post-synaptic potential

MLP – multilayer perceptron

NE - Noradrenergic

SNN – spiking neural network

TLU – threshold logic unit

CHAPTER 1 . INTRODUCTION

1.1 Effects of Neuromodulation

As any type of organism moves through its surrounding environment, it is constantly taking in and evaluating external stimuli. Based on the present condition of the organism, its knowledge of the environment, and the type of stimuli encountered, an actual physical action may need to take place. A key part of this reaction is the transmission of electrical signals and chemicals across various neuronal systems. The existence of neurotransmitters allows for the focusing, filtering, and assessment of an organism's current situation and decision-making for action selection.

Within the cerebral cortex, acetylcholine (ACh) is used to increase focus and responsiveness to certain stimuli, almost like a form of tunnel vision. Without this neurotransmitter, the organism loses the ability to parse through and prioritize multiple incoming stimuli. When multiple iterations of the same stimuli occur over a short period of time, the neurotransmitter norepinephrine (NE) can suppress the effect of the stimuli in order to prioritize new and novel events happening within the organism's environment (J. L. Krichmar 2008). Serotonin (5-HT) based systems modulate how much risk-aversive actions are taken by the organism (Cox and Krichmar 2009). Dopamine (DA) based systems are the opposite and tend toward reward-seeking actions which lead to risk-taking behavior by the organism (J. L. Krichmar 2012).

1.2 Scope of Present Work

While previous work has mainly focused simulating the effects of vertebrate neuromodulation on robotic platforms using simple artificial neurons (Prince 2013; Prince and Samanta 2013; J. L. Krichmar 2012) there has not been any in-depth research on the effects of using modern biologically realistic neural networks based on spiking neural network (SNN) for autonomous control of multiple robots in a cloud computing environment. Actual vertebrate brains consist of millions of neurons working in tandem (Kistler and Gerstner 2002) to create a responsive network. The main hypothesis of this research is whether a modern spiking neural network-cloud computing model is an effective control system based on vertebrate neuromodulation compared to a simple artificial neural network for an autonomous robot. The implementation of the SNN model can be facilitated with the current trend of parallel computing algorithms based on graphics processing units (GPU). In both cases, the networks in question would show action selection mechanisms for the robots in association with the activity levels of the neuromodulatory systems that would be influenced by the sensory inputs from the environment.

To test the hypothesis, the overall goal of this work was to model a behavioral control system based on the principles of vertebrate neuromodulation using SNN and implement it on autonomous mobile robotic platforms in a cloud computing environment making use of open-source, open-access software platform of robot operating system (ROS). The distinct objectives of this work were:

- To develop SNN model suitable for GPU based implementation,
- To implement the SNN model in a ROS-cloud computing environment on autonomous robotic platforms,
- To study the behaviors of the neuromodulated robots in single- and multi-robot configurations under different modes,
- To compare the results of neuromodulation models based on the simple neuron model and the SNN.

In this study, the performance of a simple untrained neural network model implemented on a robotic platform was compared to the performance of a spiking neural network (SNN) model implemented on the same robotic platform. First the simple neuron model was created using Robot Operating System (ROS)-compatible C++ code. The simple neuron model consisted of an input layer of "event" neurons, a second layer of neurotransmitters, and an output layer of "states". Next the code was run on a Turtlebot robotic base. Turtlebot is an open-source robotic platform consisting of an iClebo Kobuki base, a Microsoft Kinect sensor, mounting plates, and an on-board laptop running the Ubuntu distribution of Linux with the ROS platform already installed. In ROS environment, it was possible to reuse multiple open-source libraries of codes developed for general purpose robotics on the Turtlebot platform.

After running several simulations with the simple neural network in multi-robot environments, the spiking neural network (SNN) model implemented on a GPU became the focal point of the study. The open-access SNN simulator developed for off-the-shelf hardware was applied to two PCs using NVIDIA GPUs for the parallel processing of computational neuron models. The SNN simulator used neuron models based off the Izhikevich neurons, it was meant to be biologically realistic (four different parameters were used to define the spiking patterns of the neuron) yet relatively easy to compute for large number of neurons (Izikevich 2003).

Using this SNN simulator integrated with the ROS structure used before, the behavior of the robotic bases used in this study were evaluated in single-robot and multi-robot environments. From there, the amount of spiking in each group of neurons (ranging anywhere from 100 to 1,000 neurons in one group) was recorded and used for autonomous action selection as the robots moved around the environment.

1.3 Organization of Thesis

The rest of the thesis is organized in the following order:

Chapter 2 is the literature review covering three different areas. First the background of many different artificial neural network types is explained to track the progression of ANNs from their beginnings to the current neuron models. Then the individual components of the spiking neural network simulation are explained from biologically realistic nature of the neuron models to their implementation on parallel computing platforms. Finally the systems that handle the integration of the neural network models onto a cloud computing platform are analyzed and explained in full detail.

Chapter 3 covers the research methodologies used in this study. The discussion of research methodology was divided into several sections. The first section deals with the functional description of the robotic platform used to implement the neural control schemes of this study. The second section lays out the actual code configuration of the ROS networking platform. The third section presents the simple neural network model of the neuromodulation based control algorithm used in this study. The fourth section presents the evolution of the neural network model into something more biologically realistic leading to the SNN based model. The final section gives the technical details of the cloud computing configuration used in this study.

Chapter 4 presents the details of the implementation results along with discussions of results. For the first set of results the simple neural network was implemented to obtain experimental results that demonstrated the neuronal responses and the behavioral states of the robot for three modes, namely, (i) risk aversive, (ii) risk taking, and (iii) distracted modes. Results were recorded for both single- and multi-robot configurations. For the next set of results, the same steps as the first were applied again but with a spiking neural network (SNN) model. Results detailing the action selection and decision making of the single- and multi-robot configurations, under different modes are presented. Results of comparisons between the simple neuron model and the SNN model are discussed as well.

The final chapter provides a summary of the present work and the scope of future work. Each of the objectives of the work is addressed. The results are summarized to show that each of the objectives was met. Finally, ideas for possible future work are discussed.

Appendices A through C provide additional information related to the work. Appendix A presents the software implementation of the neuromodulation based neural network. Appendix B shows the snapshots of robot motion in three different modes. Publications resulting from the present work are listed in Appendix C.

CHAPTER 2 . LITERATURE REVIEW

2.1. Artificial Neural Networks

Before one delves into the field of spiking neural networks, there needs to be an introduction to artificial neural networks (ANN) in general. The field of artificial neural network is one that consists of vast and numerous types and applications that can be confusing to a newcomer from a primarily engineering background. Using a biological model based on the layout of actual neurons, it is possible to create entire networks that allow control over a certain function or object, adapting to a specific situation.

The typical basic neuron is made up of three key components: the soma, axon, and dendrites. The input of the neuron begins with the dendrites receiving electronic pulses from the outputs of different neurons. These pulses are sent from the dendrites to the soma, which collects all of these pulses. Once the total number sum of all the inputs sent to the soma reach above a certain threshold, the neuron sends a pulse through its output, called the axon. A typical neuron has one axon but multiple dendrites. These pulses (usually of about 100 mV and 1-2 milliseconds in duration) are usually called spikes. There is no physical connection between the dendrites and the axons of different neurons. Instead there is a small gap called the synaptic cleft where chemical reactions activated by the axons allow ions from the surrounding fluid to pass through to the dendrites. These chemicals are called neurotransmitters. On the axonal side of the synapse (pre-synaptic connection), the neurotransmitter molecules go across the cleft to fit into receptors on the dendrites (post-synaptic connection). This acts as a system to permit the ion flow to happen. Post-synaptic signals can be positive or negative in value. They are commonly referred to an excitatory post-synaptic potentials (EPSP) or inhibitory post-synaptic potentials (IPSP). Figure 2.1 on the next page shows a typical neuron. Figure 2.2 on the next page shows the location of a synapse (the synapse is the name of the actual location of the axon/dendrite chemical connection; the cleft is the actual gap). Figure 2.3 shows a typical form of an EPSP and IPSP.

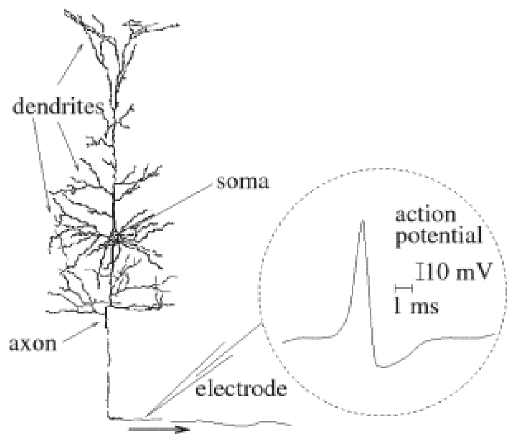

Figure 2.1. A typical neuron. *Sources*: (Ramòn y Cajal 1909) and (Kistler and Gerstner 2002, 3)

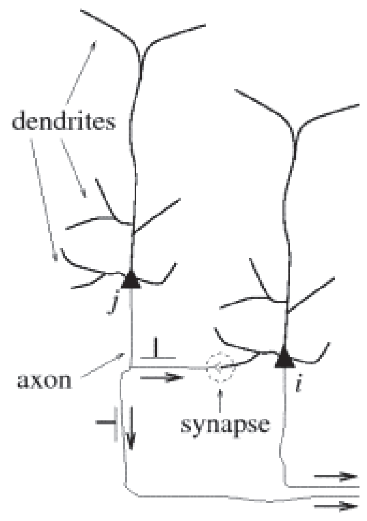

Figure 2.2. A typical synaptic connection. *Sources:* (Ramòn y Cajal 1909) and (Kistler and Gerstner 2002, 3)

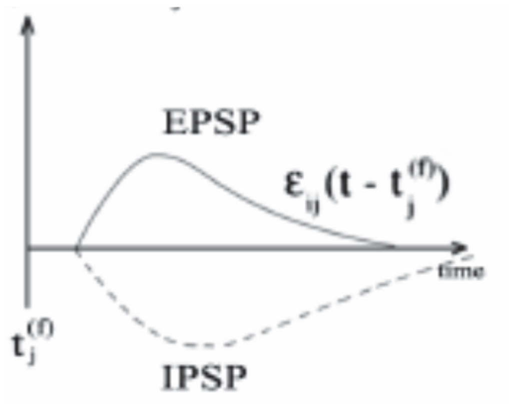

Figure 2.3. A typical form of an EPSP and IPSP. *Sources*: (Gerstner 1999)

2.2 The First Artificial Neural Network

ANNs are based upon this layout which was first popularized by Walter Pitts and Warren McCulloch in 1943. Called the Threshold Logic Unit (TLU) (McCulloch and Pitts 1943), weighed inputs are connected to a soma-like summation function. The output of the summation function is then sent to a threshold function. When the value sent to the threshold is above a certain pre-set value, the threshold function outputs a logical 1 or true signal. When the value is below the pre-set threshold the value sent by the threshold function is a logical 0 or false signal. Figure 2.4 on the next page shows a block diagram representation of the TLU.

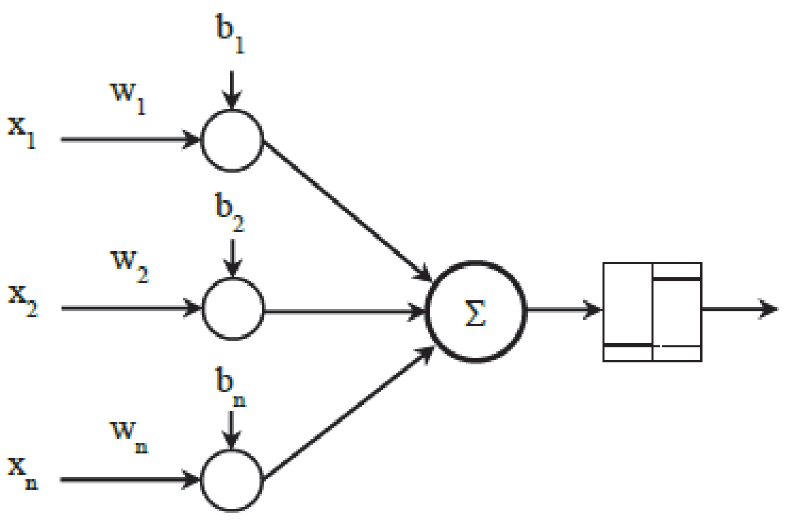

Figure 2.4. A block diagram representation of the TLU. *Source*: (Turner 2012, 10)

In Figure 2.4, the x are the inputs, and the variables w and b are the weights and biases applied to the inputs. There is no theoretical limit to the amount of inputs a single TLU can have. The values sent to the summation function are weighed and have applied biases to replicate how certain values are more meaningful than others in different applications.

For example, a neuron can be set up to replicate the thought processes of a child wanting to play with a small red toy. The inputs to a TLU are logical values (true or false) describing certain objects such as roundness, the color red, and lightness in weight. The weights and biases are set higher on input values signifying roundness and the color red. A heavy black book would have no chance of inducing a true value for the artificial neuron. A heavy purple rubber square toy would have a better chance than the book but the neuron still would not have a true output in this case. A lightweight, rubber square toy with a red color would have a higher chance than the heavier purple toy due to having the desired input color but would still be filtered out by the neuron due to the emphasis also placed on desired output roundness. A small red ball would reach all three requirements of the child's desires and would result in the neuron outputting a true value.

2.3 Second Generation Artificial Neural Networks

This landmark research by McCulloch and Pitts is credited for beginning the entire field of artificial neural networks. As the ANN field grew, there was concern that the McCulloch/Pitts model was too simple a model to follow. Thus began the second generation of neural network innovation, with changes made to the activation functions of neural networks. With traditional threshold functions, it was impossible to use them with analog signals. However, with continuous threshold functions this was an option. Following the trend of complexity replacing simplicity, this new generation of neural networks was able to perform digital computations with a smaller amount of neurons than the first generation along with their analog signal handling capabilities (Vreeken 2003).

2.3.1 The Perceptron

Further revisions were made to the TLU design by Frank Rosenblatt at the Cornell Aeronautical Laboratory in 1957. In his published research Rosenblatt presented the perceptron (Rosenblatt 1958), a TLU with a learning algorithm. According to the perceptron learning algorithm, the basic artificial neuron would be fed “training” data with the desired outputs for that data. Through multiple iterations of calculations, the weights for each input were adjusted until they converged on one set of weights to give the desired output for each input. The perceptron “learned” how to adapt to the given data and came up with a relationship that would always lead to the desired output.

To go back to the previous example with the child and the toys, a perceptron would constantly be fed the inputs and desired outputs of the different objects (small red ball – true, heavy black book, purple ball, red square – false). The learning algorithm would constantly increase or decrease the weight put on each input until the weights could no longer change. The state reached here would be the final value for a specific given “learning rate”. Not only could the perceptron self-learn but also the speed at which it did could be adjustable. Also, similar to TLU, the number of possible inputs to the perceptron was theoretically infinite.

With this innovation, it was thought possible to invent self-taught logical gates but it was quickly discovered that there were patterns that the perceptron could not be trained to recognize. In particular, it was found impossible to create the equivalent of an XOR logical gate using a perceptron. This revelation led to a massive amount of disinterest in ANN research that lasted until the 1980s.

2.3.2 The Multilayer Perceptron

In the 1980s, interest rose in the field again after it was discovered that grouping interconnected networks of perceptrons could overcome the logical limitations of only one perceptron. This configuration was also a closer representation of how the actual biological neuron was used in the brain. By adding layers of neurons (groups of neurons that are all interconnected with one another) to the input and the introduction of a new weight changing method called backpropagation (using differential equations to find the change in weights instead of the simple adding and subtracting seen in the single perceptron model) it was possible to create networks that could be trained to find relationships for more sophisticated applications. These are commonly called multilayer perceptron (MLP) artificial neural networks. Figure 2.5 below shows a simplified model of an MLP ANN.

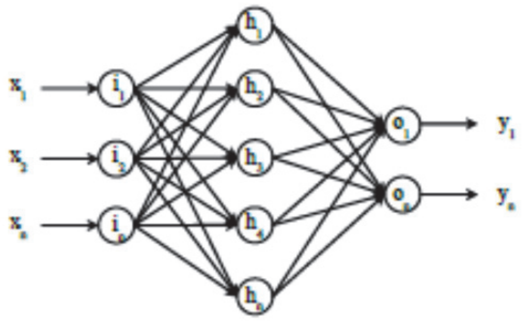

Figure 2.5. A model of an MLP ANN. *Source*: (Turner 2012, 13)

With new sparked interest in the ANN field, new artificial neuron models appeared to take advantage of not only the new methods but the increasingly available processing and computing power found in new technology that was growing more powerful and complex as the years went on.

2.4 Realism and the Newest Generation of ANN

The third generation of ANNs seeks to replicate the biological functions of neurons even further by analyzing the timing of the signals sent out by the artificial neurons. In previous generations of ANNs, it was discovered that a higher input in a network resulted in a faster frequency of pulses sent out. The average frequency of these pulses could be calculated and associated with a specific input value. This method of encoding value is called rate coding. In spiking neural networking (SNN) the focus is spent on when these pulses are sent instead of how many. This is called spike coding. The unique nature of the spiking neural network means that multiple forms of information can be sent in one pulse train. For instance, a sequence of pulses could contain information describing a signal such as amplitude and frequency. Having multiple data streams in one message mirrors how the brain works in real-life. When looking around in the world multiple signals such as light, color, and sound come to all of us at the same time. We are able to process these mixed signals naturally without being confused about which signal our brain is sending us.

When a neuron fires an output, a negative potential follows the quick positive rise that has previously been discussed. It is from this negative after-pulse that the neuron begins to slowly return to normal. The duration of this negative after-pulse is called the refractory period. During this time, the neuron cannot fire another output spike. This is seen below in Figure 2.6.

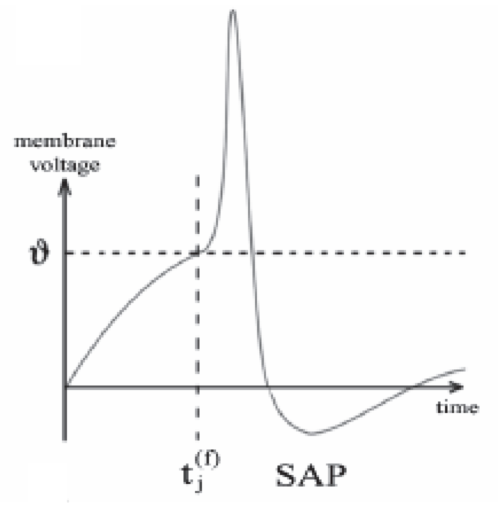

Figure 2.6. Spike after-potential. *Source*: (Turner 2012, 13)

The time taken in this refractory period can be modeled mathematically in order to create an equation to model the current state of a neuron. This can be seen in equation 2.1 below.

$$u_i(t) = \sum_{t_i^{(f)} \in F_i} n_i \left(t - t_i^{(f)}\right) + h(t) \qquad (2.1)$$

In this equation $u_i(t)$ is the current membrane potential of a neuron i, h(t) is whatever external influence that may affect the membrane (number of ion channels activated by neurotransmitters for example), n_i is the scaling factor of the membrane potential and $t^{(f)}$ represents all of times that neuron i fired. The unique nature of the spiking neural network requires that values fed into one must be in the form of spikes or an external influence must be placed upon the membrane potential. Equation 2.1 shows the latter.

2.5 The Izhikevich Model

Two spiking neural network models commonly used are the integrate-and-fire model and the Hodgkin-Huxley model. In the integrate-and-fire model, the neuron is the equivalent to a capacitor in parallel with a resistor. An impulse is sent through a low pass filter and the output is compared to a threshold. If the output value is at or above the threshold the circuit outputs a pulse. This is seen in Figure 2.7.

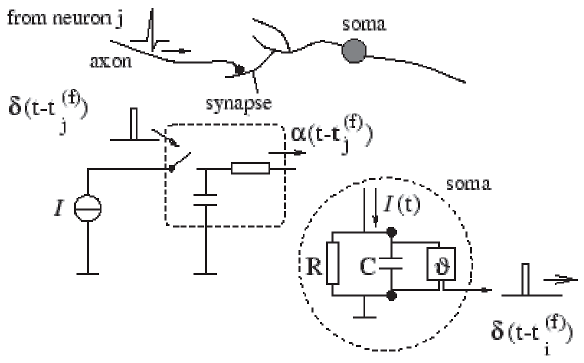

Figure 2.7. Integrate-and-fire model. *Source*: (Kistler and Gerstner 2002, 94)

In the Hodgkin-Huxley model, which was first studied on the vastly larger neurons of squids, the cell membrane is seen as a capacitor and the sodium and potassium ion channels present in the surrounding liquid are seen as resistors. When a current is injected into the cell membrane current could either charge the capacitor or go through the different ion channels. This can be seen in Figure 2.8 below. From this model, mathematical models are created with several parameters that can accurately describe the nature of a spiking neuron. However, solving for all these parameters can be mathematically intensive.

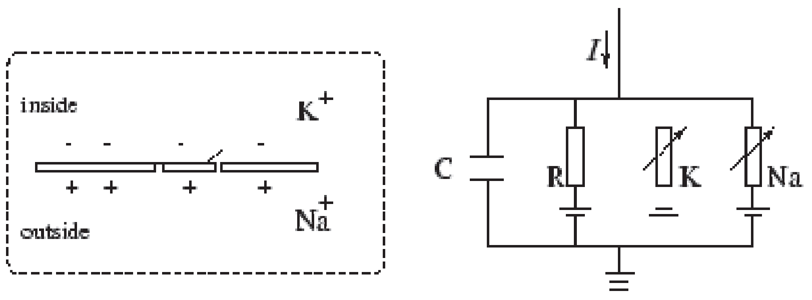

Figure 2.8. Hodgkin-Huxley model. *Source*: (Kistler and Gerstner 2002, 34)

For the purposes of this study, the model used for simulating a large number of spiking neural networks (SNN) is the Izhikevich model (Izhikevich 2003). This model is seen as a compromise between the computational simplicity of the integrate-and-fire neurons and the biological accuracy of the Hodgkin-Huxley neuron. In the Izhikevich model, the neuron is represented by the following equations.

$$v' = 0.04v^2 + 5v + 140 - u + I \qquad (2.2)$$

$$u' = a(bv - u) \qquad (2.3)$$

$$if\ v \geq 30\ mV, then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$

In these equations, *v* is the memory potential of the neuron and *u* is the membrane recovery variable. The parameter a is the time scale of the recovery variable. The parameter b is the sensitivity of the recovery variable to the fluctuations of the membrane potential. The parameter *c* is the after-spike reset value of the membrane potential. The parameter *d* is the after-spike reset value of the recovery variable and *I* is the thalamic current input. Changing the values of *I*, *a*, *b*, *c*, and *d* changes the dynamics of the spiking neuron being simulated.

2.6 Parallel Programming and GPU based Computing

To simulate SNN models, a new method of mass computation and simulation was developed on a previously unused hardware. In the PC gaming industry, expensive graphics cards have been used to give gamers virtual worlds with life-like graphics and effects for years. These powerful number-crunching processors have to constantly calculate and update graphics, lighting, modeling, and player input data for the latest games. In recent years, movements to harness the computing power of graphics cards to power neural network simulations (Nageswaran , Dutt, et al., Efficient simulation of large-scale spiking neural networks using CUDA graphics processors 2009) have caught steam. This technique also matches how the thought processes in the brain work since both graphics cards and the brain process in parallel. In the brain, this allows animals to process more than one sense at a time and to multitask between actions. In computers, this allows the graphics cards to off-load math-intensive processes from the CPU and run multiple iterations of a single simulation at once. For instance, in a simulation that tracks the number of spikes in a SNN simulation, the CPU most go through multiple iterations one-by-one while storing and writing the processes that happen. In a properly setup GPU (graphics processing unit) simulation, a single "thread" could handle one iteration of the program, while another thread would compute the next iteration at the same time, while the next thread would handle the next computation, etc. This leads to a highly efficient way of simulating a large number of spiking neurons.

2.7 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source software framework designed for the development of customized robotic applications (ROS.org | Powering the world's robots 2013). Aimed at both robotic hobbyists and researchers, many codes for hardware platforms are managed in "packages" which are uploaded online for anyone to use on ROS-compatible systems. This also allows the constant reuse of flexible codes. ROS uses data communication in the form of "topics" and "messages" among program nodes. Messages have a simple data structure, comprising typed fields and supporting standard primitive types (integer, floating point, Boolean, etc.). Topics are named buses over which nodes exchange messages. Hardware adapted to an ROS system can tap several data streams (topics) at one time. By subscribing to the ROS nodes that are publishing the necessary data (messages), the robot sensor data can be used as necessary. For instance, a topic named "color" could carry a hexadecimal message describing a certain shade of blue between a camera and a laptop node. All node connections are peer-to-peer connections governed by one central node (the "roscore") that makes sure data handshakes and data types throughout the network are all valid. Figure 2.9 and figure 2.10 show the layouts of ROS nodes and master nodes.

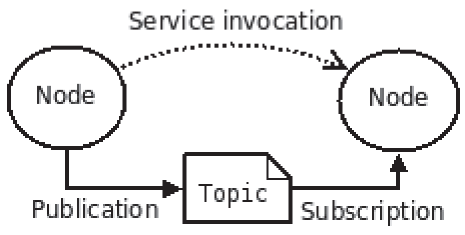

Figure 2.9 ROS node layout

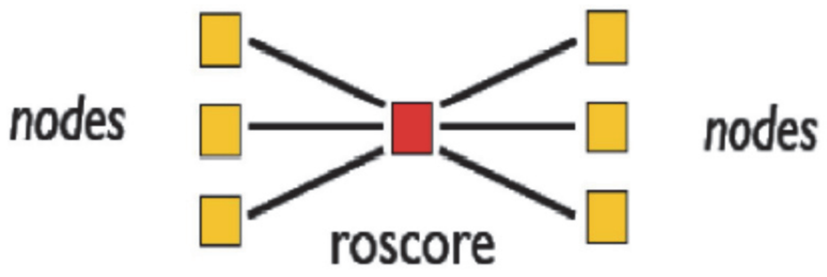

Figure 2.10 ROS master node layout

2.8 Cloud Computing

Cloud computing is simply the method of running an application or some other program on a server instead of a local machine. The spiking neural networks (SNN) involved in this study require large volumes of computation due to (1) a more biologically realistic neural model of the Izhikevich neuron and (2) the large number of neurons involved in the simulation. Therefore the laptops involved in the on-board processing of the robots only transmit sensory data to a cloud computing solution where the neuromodulatory code and GPU parallel processors are located.

To facilitate this, the on-board laptops and cloud servers are connected by Wi-Fi using a commercial off-the-shelf Linksys E2500 router. Figure 2.11 shows a schematic of a cloud computing network with robots, the configuration is, in general, termed cloud robotics.

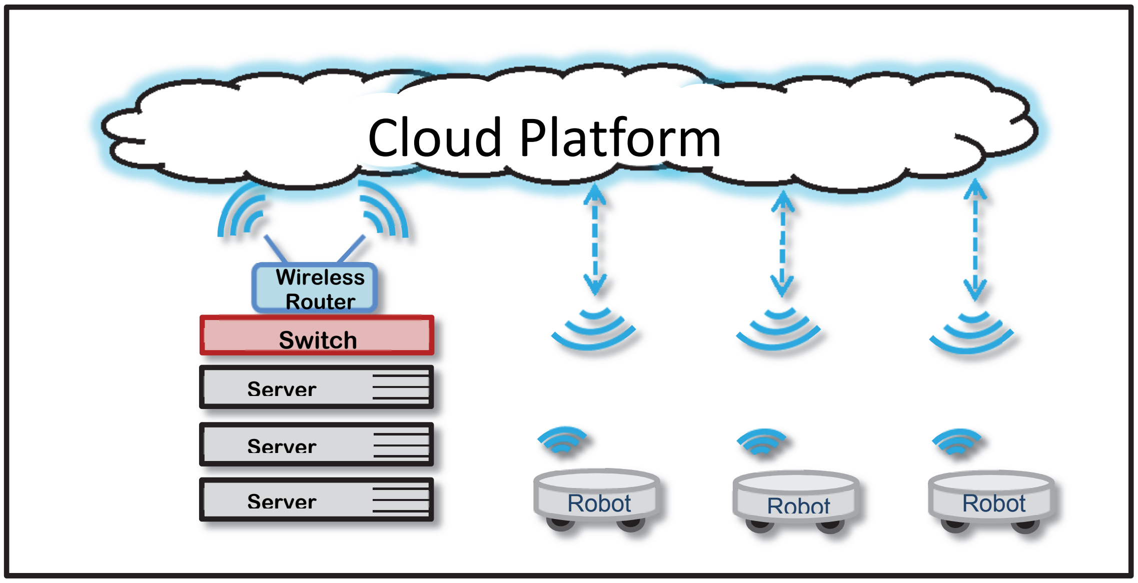

Figure 2.11 The layout of a Cloud Computing network involving robots

CHAPTER 3 . RESEARCH METHODOLOGY

3.1 Robotic Platform

Turtlebots (TurtleBot 2013) were used as the mobile robotic platforms in the present study (Fig. 1). Each Turtlebot (Figure 3.1) comes with the following components to create an integrated package for robot development: (i) an iClebo Kobuki robotic base containing infrared sensors, an internal gyroscope, and other actuators for moving the Turtlebot through its environment; (ii) a Microsoft Kinect for motion sensing and object recognition to parse visual inputs for the neural network, and (iii) an Asus X201E laptop with an Intel Celeron 847 CPU running the 64-bit Ubuntu distribution of the Linux operating system. All of these components are connected to one another through USB. ROS uses the concepts of "nodes" and "topics" to allow for the selection of data streams (such as depth image data, positional point-clouds, movement speed, etc.) to be read and have commands written to over a Wi-Fi network. While the Turtlebot moved around its environment, several onboard sensors were used as inputs to on-board ROS-compatible C++ code that was stored on the Asus laptop. The purpose of the laptop and code was to pass on these sensory data to the cloud network through the ROS framework and to receive neural information from the cloud network. A front "bump" sensor recorded whether or not the front of the robot hit an object. The internal battery was constantly monitored for voltage levels and if dropped to a critical level activated a simulated red flag within the network. The Microsoft Kinect was used as a rangefinder through the use of depth images sent to the laptop at a rate of 30Hz that were used to generate a 2-D laser scan of the environment in front of it. This allowed the Kinect to have an effective range of 10 meters to 0.45 meters in front of the camera. There is a docking station used for not only charging the robotic base but also influencing the robot to "go home" and search for the base, by interacting with three IR sensors on the Turtlebot's base.

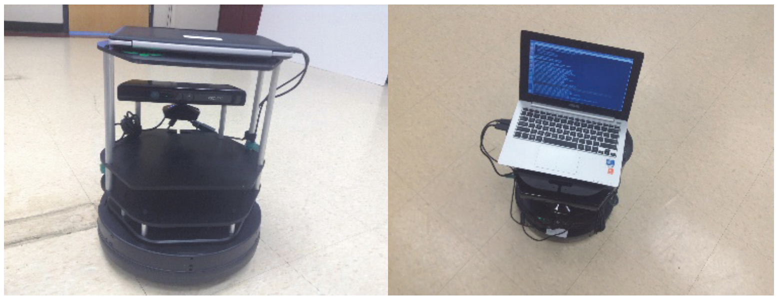

Figure 3.1 The Turtlebot with Kinect and Laptop

3.2 ROS Code Platform

The Robot Operating System (ROS) code structure in this study was done in C++ and made the Turtlebot's on-board laptop, Microsoft Kinect sensor, and Kobuki base as the key nodes within the different network configurations (with the on-board simple neural network or the off-board GPU based SNN). The main program flow in ROS (Figure 3.2) followed that the Microsoft Kinect node sent sensory data and the Kobuki node sent important diagnostic data to the on-board laptop to be used as inputs for the neural network. If the neural network used was within the on-board laptop, the on-board calculations were made and movement data containing the appropriate action was sent to the Kobuki base. If the neural network used was on the cloud server, the sensory data received from the Kinect was sent to the cloud network through the laptop and the robot waited for a response back from the cloud node. Once the cloud node returned with the correct response, this was sent to the Kobuki base through the on-board laptop which sent the appropriate action commands to the mobile robotic base.

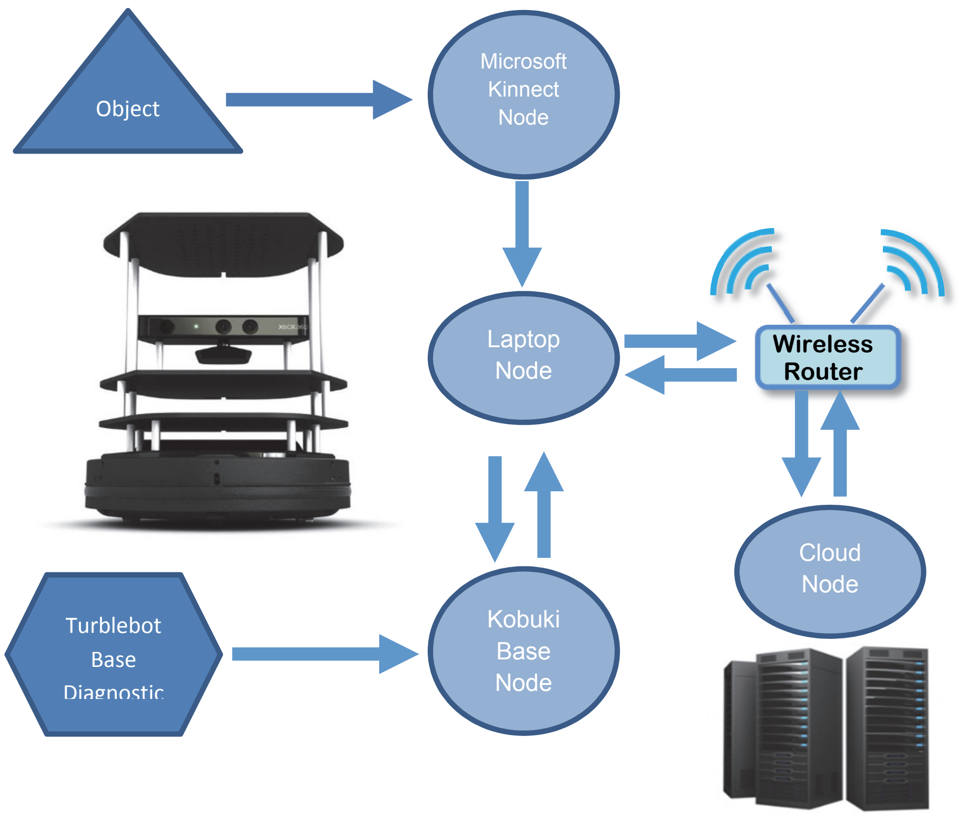

Figure 3.2 ROS code structure layout

3.3 Simple Neuromodulation Model and States

The 3-layer neural network structure of Figure 3.3 was used to study how efficiently it would perform to control an autonomous robot's behavior, similar to (Prince and Samanta 2013). Previous work with this model has been reported by (J. L. Krichmar 2012) in the field of autonomous robotics. The model consisted of three groups of neurons - event neurons from sensory signals, neuromodulatory neurons and behavior state neurons. The first layer of neurons indicated the incidents happening in the real world environment for the robot. The entire experiment was run to test how the robot would respond if any of events on the first layer took place. This network structure was designed in such a way that it would be capable of accommodating several events taking place simultaneously in the real world environment.

For this work, four events neurons were used. These event neurons wee simple activated meaning if there was an event, it would be set as 1, and reset to 0 otherwise. These four events were – OBJECT, BATTERY, BUMP, and BEAM. The robot swept in a 180 degree arc to read the distances of objects in its environment. Event Object occurred if any of the distance parameters were less than 0.52 m. Event Battery got triggered when the battery level of the robot dropped below certain percentage (while running the experiment) since its last charge. Event Bump neuron was initiated and triggered by the built-in bump sensor of the Kobuki Turtlebot base when it physically bumped against any object. The Bump event was also activated if distance measured by any of the parameters was less than 0.72 m. Event Beam was triggered when one or more of the infrared emitters of the robot's docking "home" base were detected.

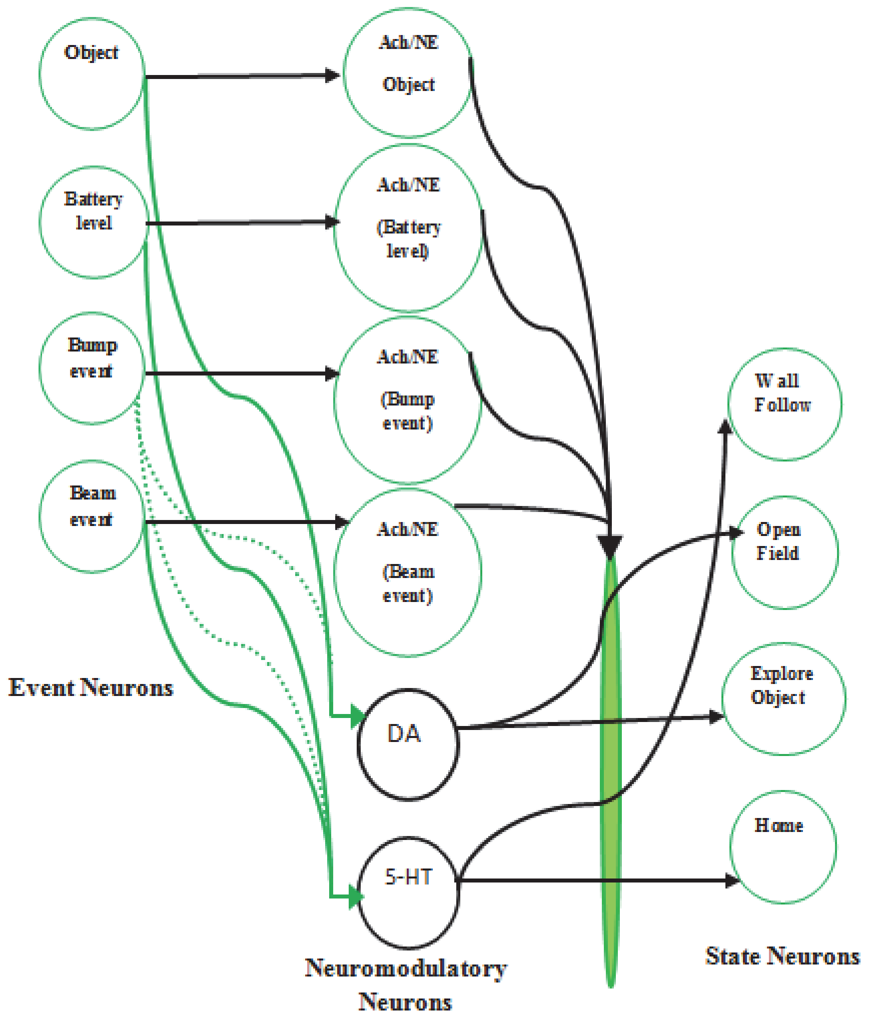

Figure 3.3 The structure of the simple neural simulation model (J. L. Krichmar 2012).

The neuromodulatory layer consisted of four ACh/NE neurons, one for each event, one dopaminergic neuron (DA), and one serotonergic neuron (5-HT). The last layer of four neurons indicated the four different behavioral outputs as states. The four behavior states were: 1) Wall-Following, 2) Open-field, 3) Explore Object, and 4) Find Home. There were also two sub-states of Find Home called At Home and Leave Home when the actual docking station was found. The simulation cycle time was about 14 s which accounted for the time to read sensor data, update neural simulation, and send a command to the robot motors. The robot would stay on one of the states at the end of each simulation cycle but kept switching in between those states based on neuromodulatory response during the entire run period.

The connection weights between event neurons and the state neurons were 1 and these weights did not get updated. The connection weights between the event neurons to the ACh/NE neurons were initialized at 1. The connections between event neurons and ACh/NE neurons were kept depressive and both DA and 5-HT neurons were kept facilitative. The connection weights between event neurons and neuromodulatory neurons go updated after the end of the every simulation cycle especially when the robot was running in both risk-taking and risk-aversive modes. In distracted behavior mode, the connection weights between the event neurons and the neuromodulatory neurons were kept at 1 throughout the run period. The state neurons were connected internally all-to-all. The connections indicated the intrinsic inhibitory weight connections with value of -1 and intrinsic excitatory connections with a value of 0.5.

The relationships for neuronal activity, neuronal inputs and weight updating are presented briefly here for completeness. The activation function for all neuromodulatory and state neurons was governed by sigmoid function:

$$n(k) = \frac{1}{1+e^{-gI(k)}} \tag{3.1}$$

where *I* was the input to the neuron, *g* was the gain of the function, and *k* denoted the simulation cycle index. Since the activation function for all neurons were governed by the sigmoid function, the activity values of these neurons remained within 0 to 1.

The input to the neurons for all the neuromodulatory neurons and the state neurons was given as:

$$I_j(k) = b + \sum_i c(k) n_i(k) w_{ij}(k) + p n_j(k-1) + nm(k) \tag{3.2}$$

where *b* was the baseline input set to -1.0 for DA and 5-HT neurons, -0.5 for ACh/NE neurons, and -1.0+rand (0.0,0.5) for state neurons, *c(k)* was set to the sum of DA and 5-HT neuronal activity for inhibitory connections, otherwise $c(k)$ was set to 1.0 [3]. p was the persistence set to 0.25 for all neurons and nm(k) is the neuromodulatory input into last layer of state neurons:

$$nm_i(k) = \sum_j \sum_l n_l(k) w_{li}(k) AChNE_j(k) e_j(k) w_{ji}(k) \tag{3.3}$$

where $nm_i(k)$ was the neuromodulatory input into state neuron i, $n_l(k)$, was the activity of either the DA or 5-HT neuron, $w_{li}(k)$ was the weight from neuromodulatory neuron *l* to state neuron *i*, $AChNE_j(k)$ and $e_j(k)$ were the activities of ACh/NE and event neurons corresponding to event *j*, and $w_{ji}(k)$ was the weight from event neuron *j* to state neuron *i*.

The updating of the connection weights was based on both the occurred events and the synaptic plasticity which was given by the following equation:

$$w_{ij}(k) = \begin{cases} p w_{ij}(t-1) & if \quad e_i = 1 \\ w_{ij}(k-1) + \frac{1 - w_{ij}(k-1)}{\tau} & otherwise \end{cases} \tag{3.4}$$

where *i* was the index of the event neuron, *j* was the index of the 5-HT, DA, or ACh/NE neuron, *p* was the amount of change in response to an event, and τ, which was set to 50, was a time constant that governed the rate at which weights returned to their original value.

3.4 Spiking Neuromodulation Model and States

To integrate the biologically realistic Izhikevich artificial neuron model in the control system two computers using NVIDIA graphics processing units (GPUs) with CUDA architecture are used for parallel computing. Each computer used the same copy of a SNN simulator platform that was optimized to take advantage of the parallel processing capability of the GPUs. The SNN simulator used in this paper is a publicly available self-contained resource used in previous work (Richert, et al. 2011). The simulator also uses a code interface based on neural group construction. For every type of input into the neural network there would be a group of neurons defined as "spike generators" to inject a start to the network. These generators would replicate the stimuli a vertebrate's brain would encounter when using "spike coding" (Verken 2003) as a way of processing information. Then each of the neuromodulators used in the neural network would be defined as individual groups of neurons in the spiking neuromodulation simulator with the four Izhikevich parameters defining the spiking behavior of each group. . For this simulation, all groups used the Regular Spiking Izhikevich model (Izhikevich 2003, 2) with parameters of $a = 0.02$, $b = 0.2$, $c = -65.0$, and $d = 8.0$.

The actual synaptic connections between each neurons group were defined as either inhibitory or excitatory in nature. That is, whether or not an incoming spike from a pre-synaptic neuron would increase or decrease the ability of the post-synaptic neuron to activate. The data flow for the simulation was as follows: once the robot received its network inputs during its quarter-sweep, the data would be transferred to a cloud computing platform waiting on said inputs. The simulation created a "spike generator" group for each of the present stimuli. The simulated spikes from the sensory inputs is sent to through the various pre-defined synaptic connections. After the simulation was done for its one second run, the time and number of each event, neuromodulator, and state neuron spike was recorded in a data file located on the cloud PC. A MATLAB script was then run in the background to calculate the number of spikes of each type. ROS, which handled the automation of the simulation process also handled transmitting the number of resulting spikes back to the robot. Reading the number of spikes and taking in consideration its surroundings, the robot took the appropriate (most active) action.

The synaptic connections between all groups of neurons were set as excitatory except for the connection between the dopamine and serotonin groups. This synaptic connection was set to be inhibitory, mirroring the opposing effect that each neuromodulator had on each other (J. L. Krichmar 2013). Every excitatory connection had a weight of +1.0 and every inhibitory connection had a weight of -1.0. Synaptic plasticity (Alexander and Sporns 2002) was included to duplicate the phenomenon of a synaptic connection becoming stronger as it received constant spikes between two neurons. The dopamine and serotonin groups totaled 1,000 neurons each. The state neuron groups totaled 100 neurons each. The spike generating event inputs also totaled 100 neurons each. The ACh/NE group totaled 4 neurons (to facilitate as many spikes as possible to induce the "tunnel vision" effect). Altogether there were up to 2,804 neurons simulated at any time during the runs. The number of neurons used to represent each group came from the increasing tradeoff between neuron complexities for stability. With neuron counts over 5,000 in the simulation, output states and neuromodulator levels stopped correlating and the results became more random. Simulation time of the network could be defined down to the millisecond but for this study they were set to one second. The structure of the SNN model is shown in Figure 3.4.

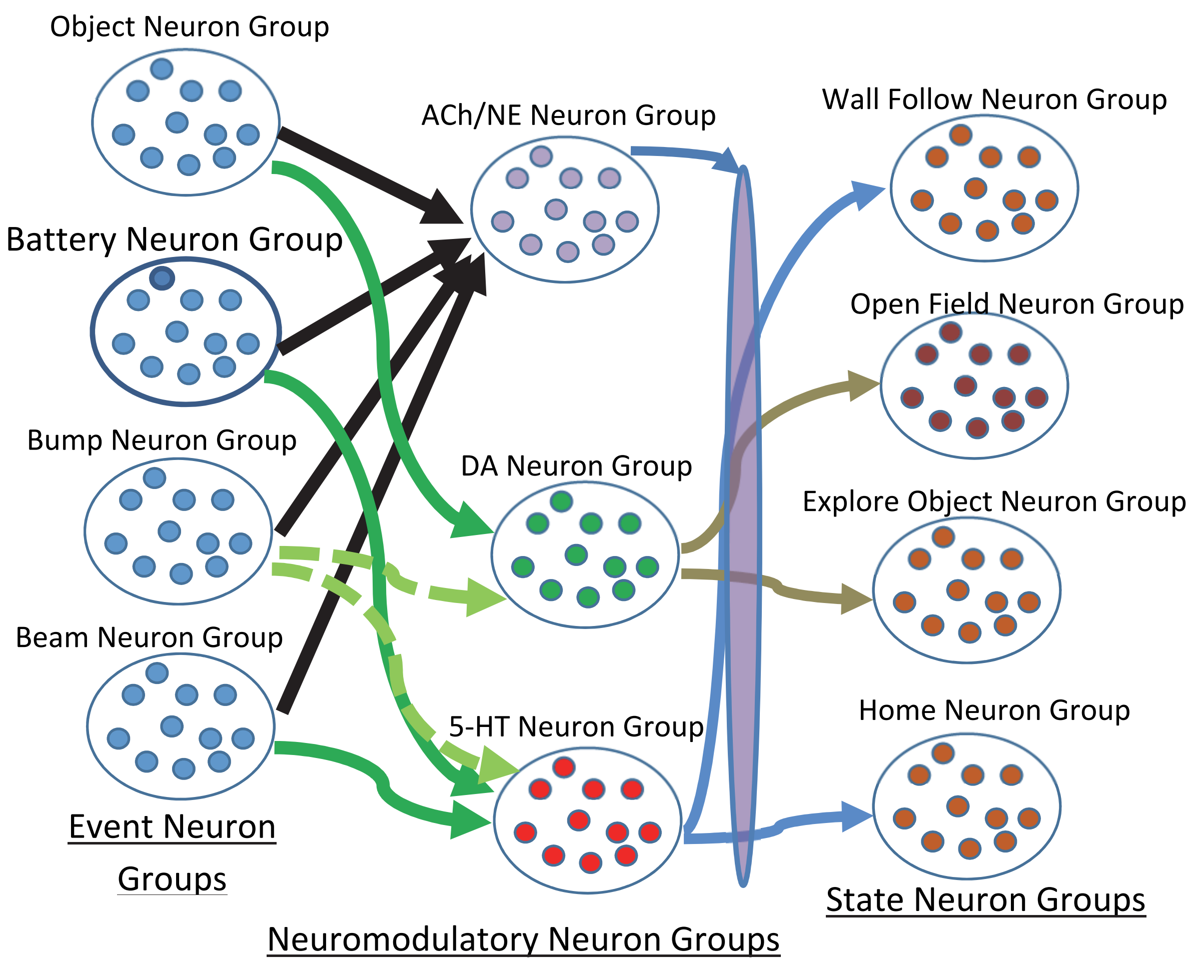

Figure 3.4 The structure of the spiking neural simulation model

3.5 Cloud Computing Configuration

To facilitate the necessary amount of computing power for the Spiking Neural Network cases, two different PCs were used as cloud servers for the study. One server used an NVIDIA Tesla K20c GPU as the parallel processing unit along with 32 GB of RAM and a six-core Intel E2620 CPU running a 64-bit partition of Ubuntu Linux. In the other server were two NVIDIA Tesla C2075 GPUs within a PC using 16 GB of RAM and a quad-core Intel E5620 CPU also running the 64-bit edition of Ubuntu Linux. Used to facilitate the data traffic between Turtlebot and server was a Linksys E2500 router. Each server ran the Groovy Galapagos (sixth edition) distribution of ROS.

CHAPTER 4 . EXPERIMENTAL RESULTS AND DISCUSSIONS

4.1 Simple Neuron Model (Single Robot)

A series of experiments were run in a lab studio environment where many tables, chairs and other solid objects were kept. Initially, 5 minutes of experiments were carried out to see the robot's behavioral performance under three different running conditions. 1) Risk aversive behavior: Bumps were treated as potentially harmful by connecting bump event neurons to the 5-HT neuron. 2) Risk taking behavior: Bumps were treated as novel and interesting by connecting bump event neurons to the DA neuron. 3) Distracted behavior: The second condition was repeated with the ACh/NE neurons kept always active (activity value =1).

4.1.1 Risk-Aversive Robot

During this mode of operation, in the neural network, the bump event neuron was connected to the 5-HT neuron. Results of robot run in this mode are shown in Figure 4.1 through Figure 4.5. As can be seen in Figure 4.1, the robot started off near an object and roamed until it was close to a wall, resulting in two bump events. In Figure 4.2, the robot was near a wall and the home base resulting in an increase of ACh/NE neural activity to focus on those events. And while both the bump and beam events occurred again near the end of the run, the ACh/NE neural activity was not as high since these events were not as novel as before. In Figure 4.3, while there were spikes of dopamine throughout the run, there was a consistent value of serotonin through the middle of the run. The neural activity as depicted in Figure 4.4, gives insight on how the states in Figure 4.5 were selected. The Wall Follow state is the default state and it was not until an object came into view that another state was selected. A larger value of neural activity by the Explore Object neuron over the Open-Field neuron resulted in the first state switch into briefly the Explore Object state followed by detecting the home base. After constantly detecting the home base by being in its field but not being able to find the charging portion of the base, an internal timer ended the Find Home state as the robot left the search for home and continued to follow walls as a way of avoiding danger. The neural activity in this run successfully mimicked a small animal staying out of danger to find home, only breaking a pattern for a short while to explore a new object for a very brief amount of time.

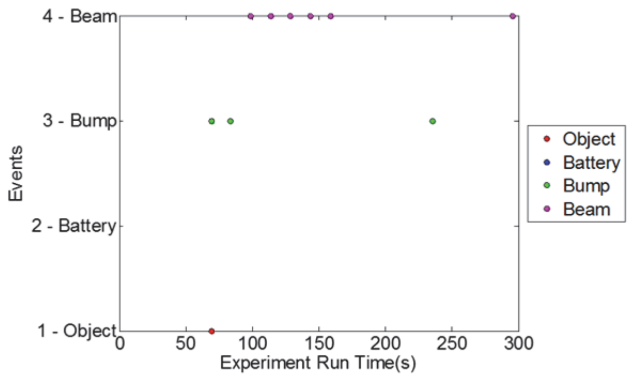

Figure 4.1 Events during the robot motion (Single Robot, Risk Aversive)

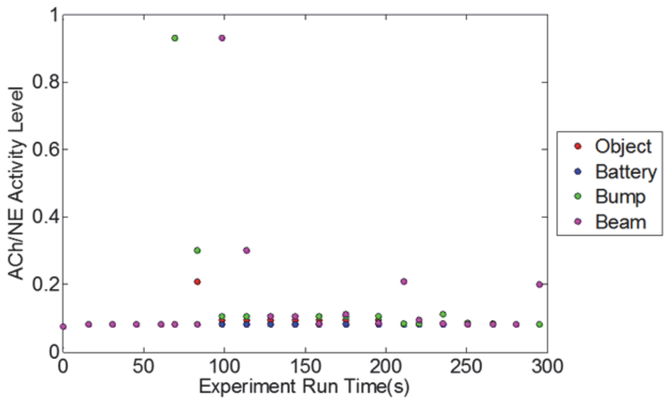

Figure 4.2 Activity of ACh/NE during the robot motion (Single Robot, Risk Aversive)

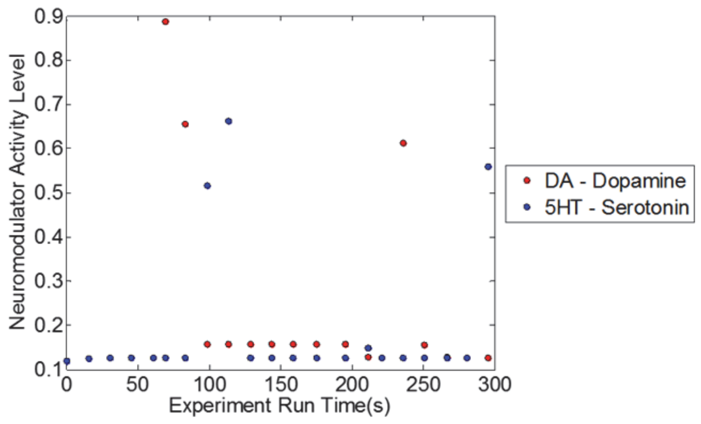

Figure 4.3 The activity of DA and 5-HT neurons during the robot motion (Single Robot, Risk Aversive)

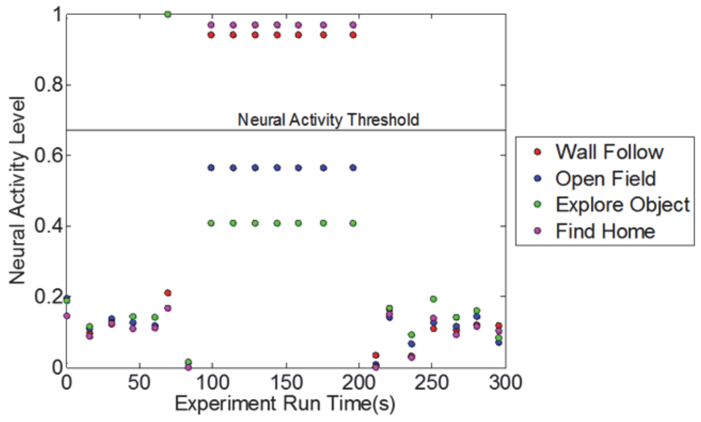

Figure 4.4 Activity of State Neurons greater than threshold (0.67) (Single Robot, Risk Aversive)

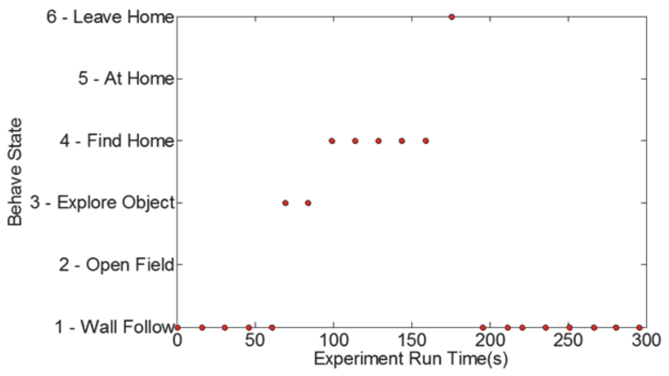

Figure 4.5 State transition during the robot motion (Single Robot, Risk Aversive)

4.1.2 Risk-Taking Robot

During this behavior mode of the robot, in the neural network the bump event neuron was connected to the DA neuron making the event interesting and worth exploring. The results for this mode are presented in Figure 4.6 through Figure 4.10.

As seen in Figure 4.6, in the risk-taking mode the robot had enough room to move around while not running into many objects or walls, partly due to the fact that being in a more adventurous mode kept it away from walls. There were two bump events halfway through the run and meeting with an object near the home base towards the end of the run. In Figure 4.7, the ACh/NE neural activity was focused first on the wall bump, then the home beam and then finally the object near the home beam. In risk-taking mode, the bump event was linked to dopamine, resulting in the spikes in dopamine activity. In Figure 4.8, when the home base was detected this caused a spike in serotonin activity, but then this was drowned out by a larger spike in dopamine activity when an object was detected near the end of the run. In Figure 4.9, the Open Field and Explore Object neurons were most active in the beginning of the run with a spike in neural activity of the Find Home neuron (explained the by the robot's proximity to the base) and Wall Follow neuron with a final spike of the Explore Object neuron to end the simulation. As seen in Figure 4.10, once the transition from the default state of Wall Following was made most of the time was spent exploring objects in the robot's field of vision or roaming in the Open Field state. Throughout the run there was not a single transition to any of the "home" states due to being linked to serotonin based actions. The difference between the risk-adverse and risk-taking modes can be illustrated by the two behavior switching figures (Figure 4.5 and Figure 4.10)

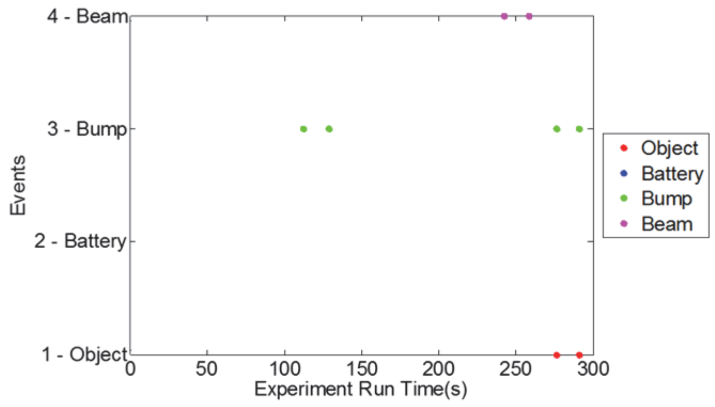

Figure 4.6 Events during the robot motion (Single Robot, Risk Taking)

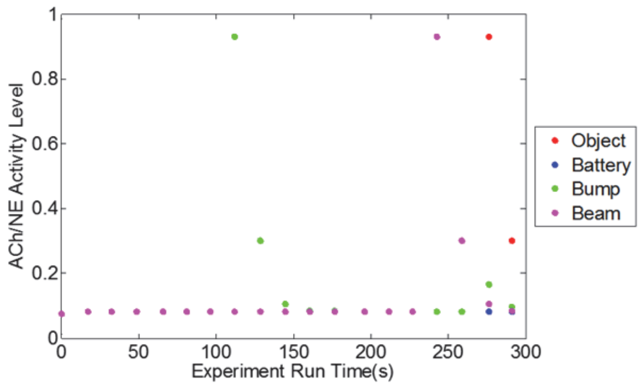

Figure 4.7 Activity of Ach/NE during the robot motion (Single Robot, Risk Taking)

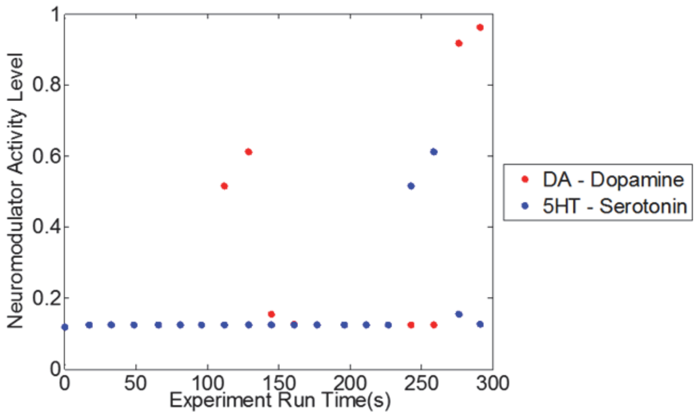

Figure 4.8 The activity of 5-HT and DA neurons during the robot motion (Single Robot, Risk Taking)

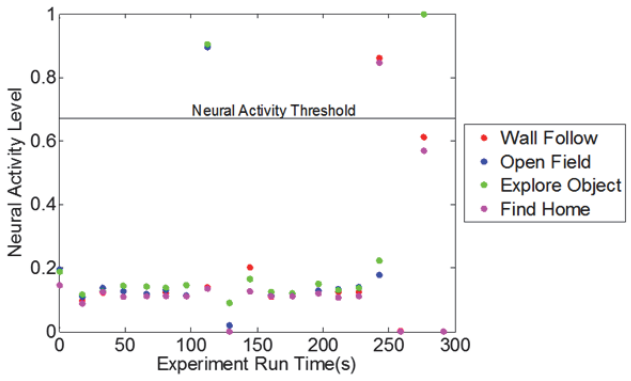

Figure 4.9 Activity of state neurons greater than threshold (0.67) (Single Robot, Risk Taking)

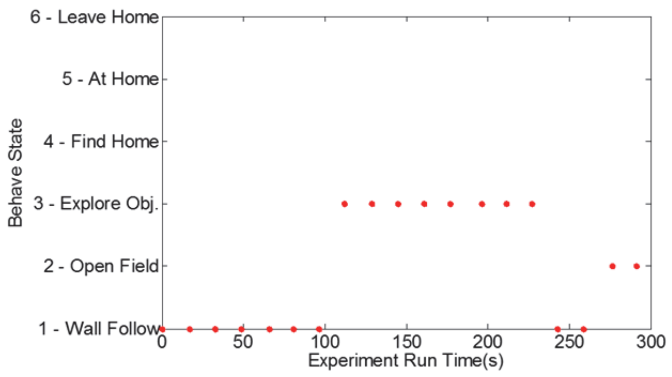

Figure 4.10 State transition during the robot motion (Single Robot, Risk Taking)

4.1.3 Distracted Robot

The third condition was experimented to see how the robot behaved as its attention system was marred. The third mode was a risk-taking subset to see how the robot behaved as its attention system was always active. This had the effect of not allowing the robot to focus on a single event. In Figure 4.11, the robot began the run around the home base and then had repeated events around walls and objects. In Figure 4.12, the neural activity of the ACh/NE neurons was at their maximum. As the distracted mode was a subset of the risk-taking mode, bumps were linked to dopamine resulting in the large amount of dopamine-related neuromodulator activity in Figure 4.13. Due to the maximally active ACh/NE, the robot tried to respond to every frequent event and was unable to ignore any unimportant events. This was why the robot was more prone to switching between the states quicker than the risk-taking and risk-averse modes. Figure 4.11 through Figure 4.15 show the importance of the ACh/NE neurons in focusing attention for the robot to respond to novel events as interesting and ignore the recurrent ones as uninteresting.

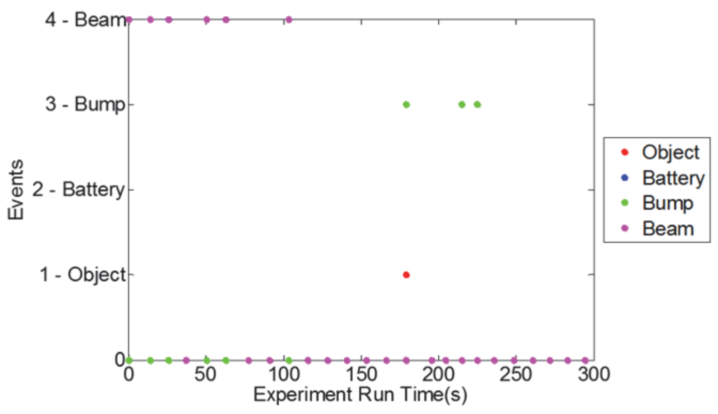

Figure 4.11 Events during the robot motion (Single Robot, Distracted)

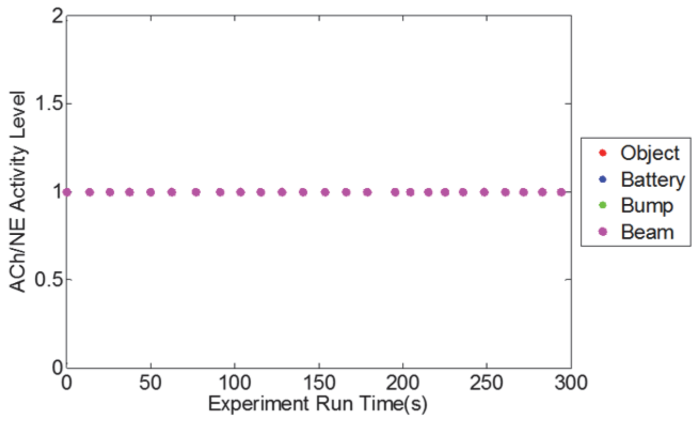

Figure 4.12 Activity of Ach/NE during the robot motion (Single Robot, Distracted)

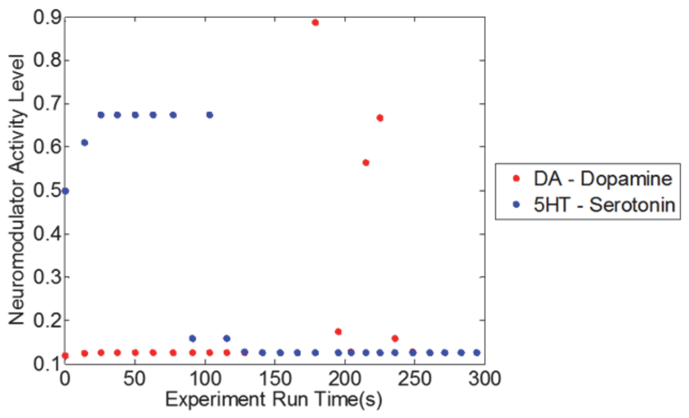

Figure 4.13 The activity of 5-HT and DA neurons during the robot motion (Single Robot, Distracted)

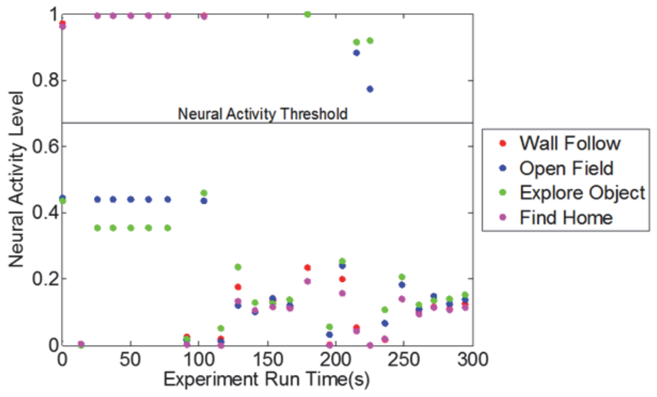

Figure 4.14 Activity of state neurons greater than threshold (0.67) (Single Robot, Distracted)

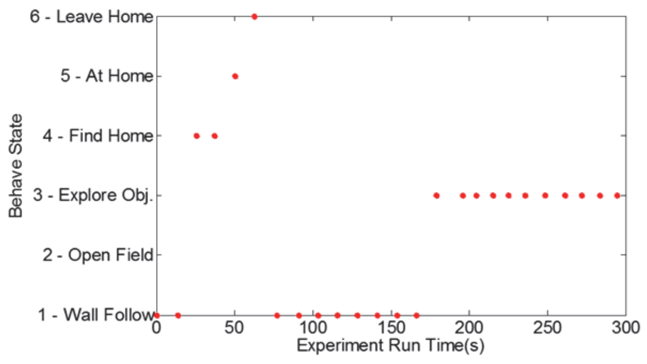

Figure 4.15 State transition during the robot motion (Single Robot, Distracted)

4.2 Simple Neuron Model (Two Robots)

The procedure of the single-robot experiments was repeated but with a second Turtlebot added in the room near the first one. Each Turtlebot used the same neuromodulation code but in combination of various modes. Each combination of modes is listed below. Under each heading Robot A was run in the first mode listed, while Robot B was run in the second mode. So for instance, for Risk Aversive/Distracted, Robot A was in Risk Aversive mode and Robot B was in Distracted mode.

4.2.1 Risk Aversive Robot with Risk Taking Robot

The results of this run demonstrated the predatory actions of Robot B (risk-taking) towards Robot A (risk-aversive). Both robots experienced the same type of events. Figure 4.16 and 4.17 show the robots were near both the home base and multiple objects in their way but the actions they took were very different. In Figure 4.20, Robot B saw Robot A and began to follow it in its Explore Object state. On the other hand, Robot A only wanted to follow the wall to look for its way back to the home base. Figure 4.18 and Figure 4.19 show the representative neural activity leading to these actions. As Robot B was in a risk-taking mode, there was more dopamine-related neural activity than serotonin-related neural activity during its run. The opposite was true for the risk-aversive Robot A.

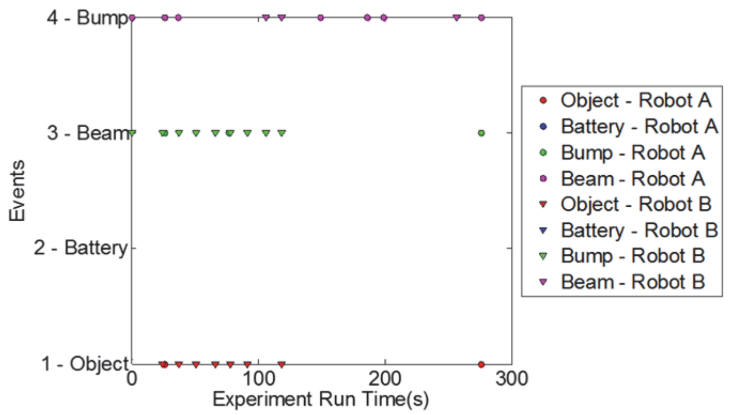

Figure 4.16 Events during the robot motion (Two Robots, Risk-Aversive, Risk-Taking)

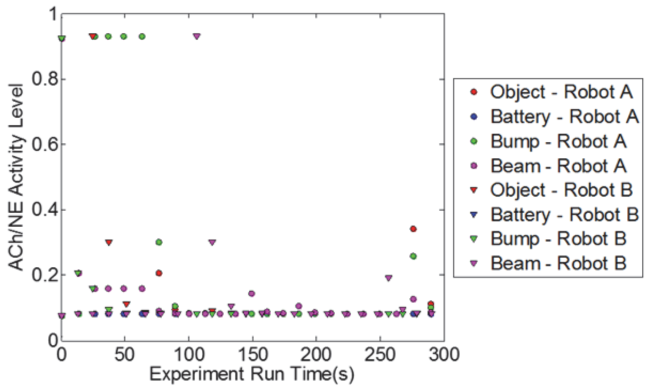

Figure 4.17 Activity of ACh/NE during the robot motion (Two Robots, Risk-Aversive, Risk-Taking)

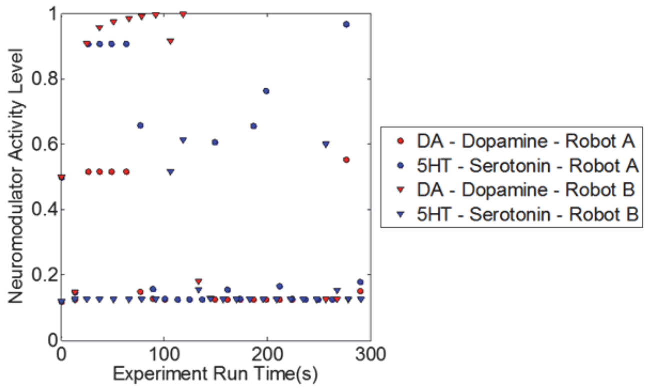

Figure 4.18 The activity of DA and 5-HT neurons during the robot motion (Two Robots, Risk-Aversive, Risk-Taking)

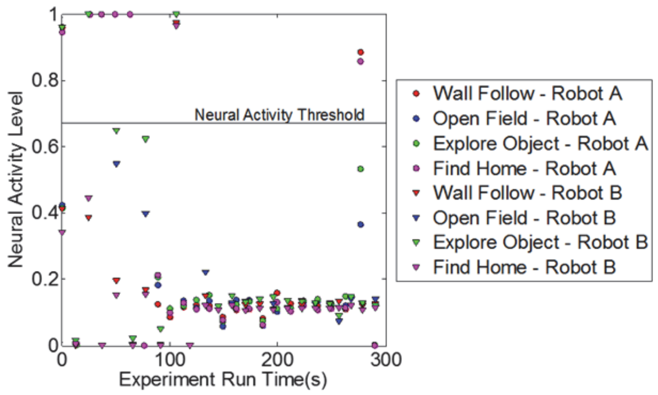

Figure 4.19 Activity of state neurons greater than threshold (0.67) (Two Robots, Risk-Aversive, Risk-Taking)

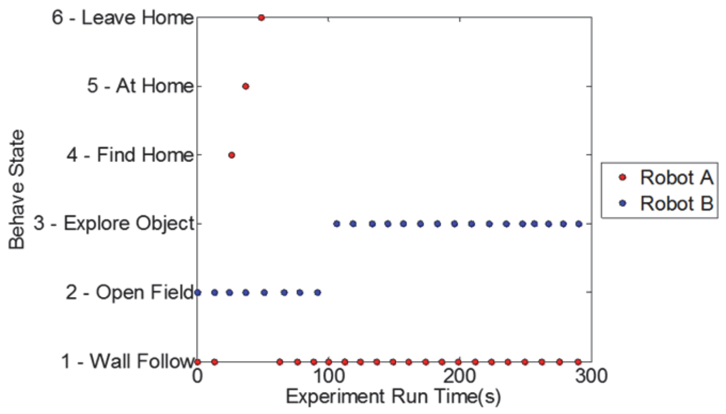

Figure 4.20 State transition during the robot motion (Two Robots, Risk-Aversive, Risk-Taking)

4.2.2 Risk Aversive Robot with Risk Aversive Robot

With both Turtlebots in risk-aversive mode, they were placed close to one another near the beginning of the run. During the run both robots avoided the other and instead followed walls in order to avoid interaction with other objects. As seen in Figure 4.25, both robots briefly transitioned to exploratory states (Open Field for Robot B and Explore Object for Robot A) but most of the time was spent wall following with the occasional run to the home base. Both robots constantly ran close to objects (Figure 4.21) but the ACh/NE neuron focus only lasted for a quick spike (Figure 4.22) demonstrating its ability to filter out non-novel event inputs into the network. As both were in risk-aversive mode, both robots had a high amount of serotonin levels throughout this run (Figure 4.23).

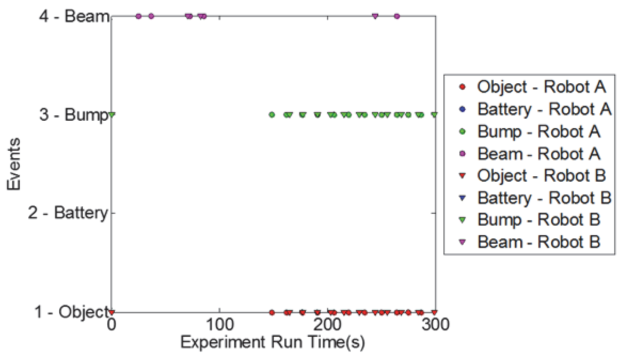

Figure 4.21 Events during the robot motion (Two Robots, Risk-Aversive, Risk-Aversive)

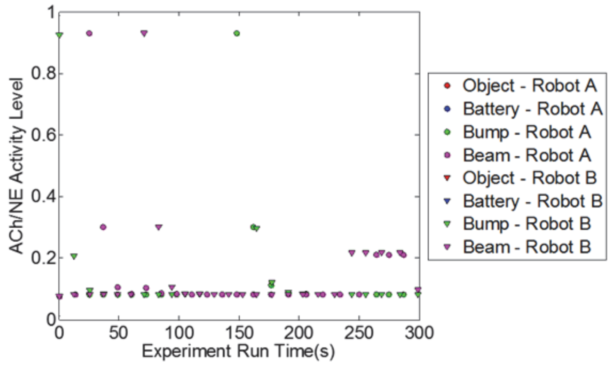

Figure 4.22 Activity of ACh/NE during the robot motion (Two Robots, Risk-Aversive, Risk-Aversive)

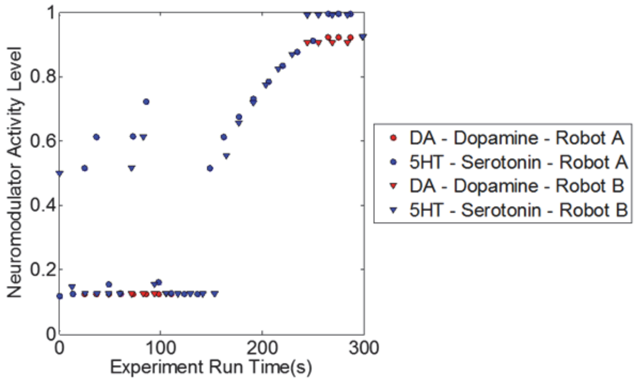

Figure 4.23 Activity of ACh/NE during the robot motion (Two Robots, Risk-Aversive, Risk-Aversive)

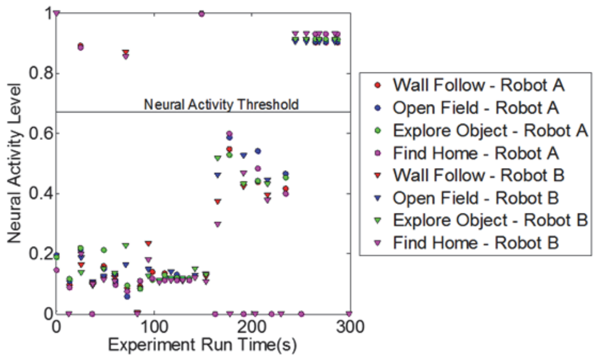

Figure 4.24 Activity of state neurons greater than threshold (0.67) (Two Robots, Risk-Aversive, Risk-Aversive)

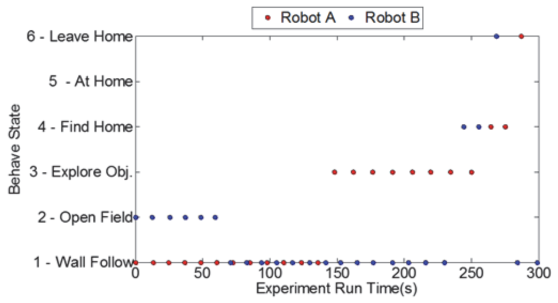

Figure 4.25 State transition during the robot motion (Two Robots, Risk-Aversive, Risk-Aversive)

4.2.3 Risk Aversive Robot with Distracted Robot

With one Turtlebot in risk-averse mode (Robot A) and one robot in distracted mode (Robot B), one can see the lack of focus of the ACh/NE neurons for Robot B in Figure 4.27. Once again while both robots detected the nearly the same events (Figure 4.26) the actions taken were different (Figure 4.30). The risk-aversive robot stayeds in the Wall Follow mode for the entire run. The distracted robot started in the default Wall Follow mode before spotting the risk-aversive robot (the first Wall Follow to Explore Object state transition for Robot B in Figure 4.30) and following it for a few seconds before switching to Open Field and then spotting another object in the distance. In Figure 4.28, the serotonin-related neural activity in Robot A and the dopamine-related neural activity in Robot B were most active during the run as expected. In Figure 4.29, the lack of focus for Robot B was demonstrated, explaining the quick transition of states by Robot B from following Robot A to switching to Open Field to following another object in the distance.

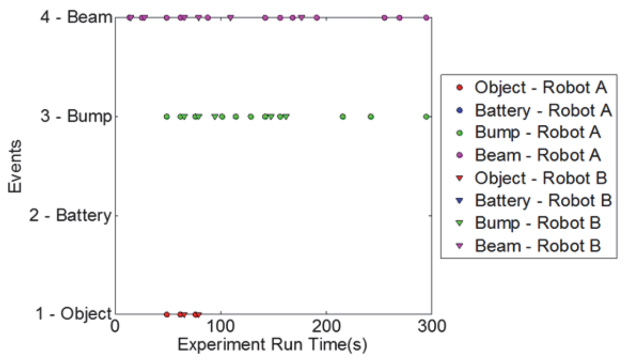

Figure 4.26 Events during the robot motion (Two Robots, Risk-Aversive and Distracted)

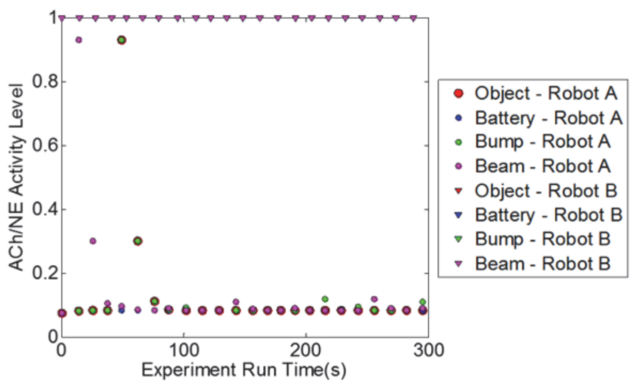

Figure 4.27 Activity of ACh/NE during the robot motion (Two Robots, Risk-Aversive and Distracted)

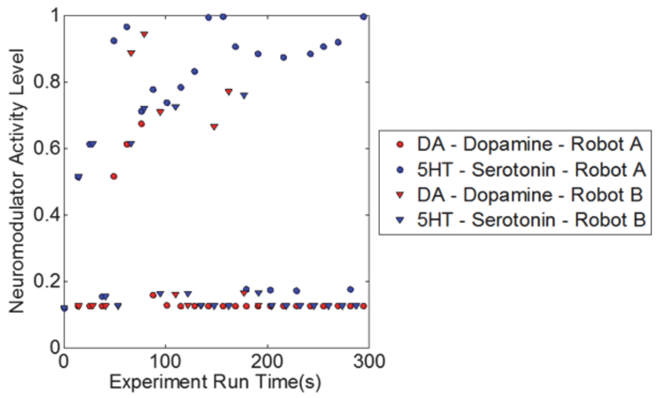

Figure 4.28 The activity of DA and 5-HT neurons during the robot motion (Two Robots, Risk-Aversive and Distracted)

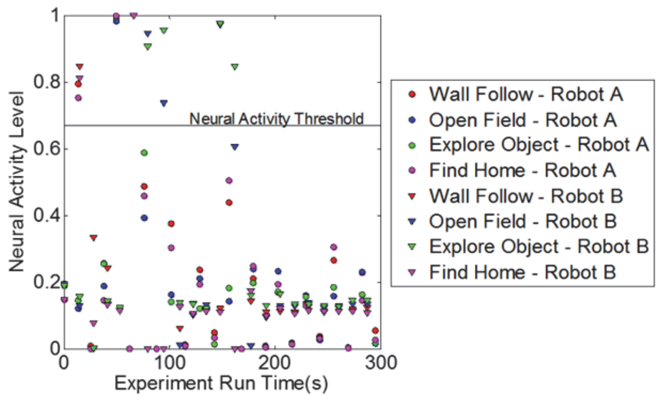

Figure 4.29 Activity of state neurons greater than threshold (0.67) (Two Robots, Risk-Aversive and Distracted)

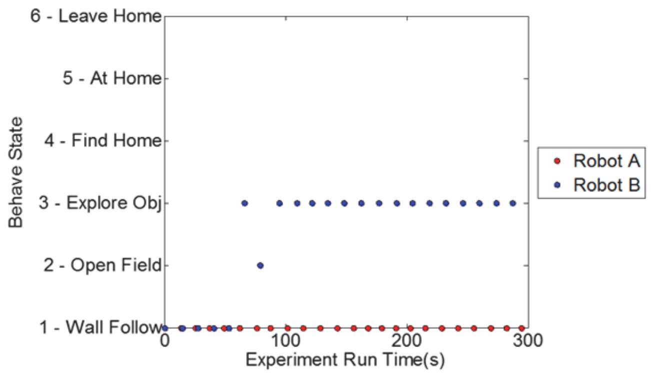

Figure 4.30 State transition during the robot motion (Two Robots, Risk-Aversive and Distracted)

4.2.4 Risk Taking Robot with Risk Taking Robot

Results for two robots, both in risk-taking mode, are presented in Figure 4.31 through Figure 4.35. With both Turtlebots in the risk-taking mode there were moments when one Turtlebot would notice the other and followed it for a short amount of time. In Figure 4.35, where the object Robot A wanted to explore was actually the Robot B. During this run Robot A saw a human being during its course and went towards it. The human then walked away leaving Robot A to its own devices. The infrared home base was also nearby during this run. Since both were in a risk-taking operation, there was an abundance of dopamine activity on both robots.

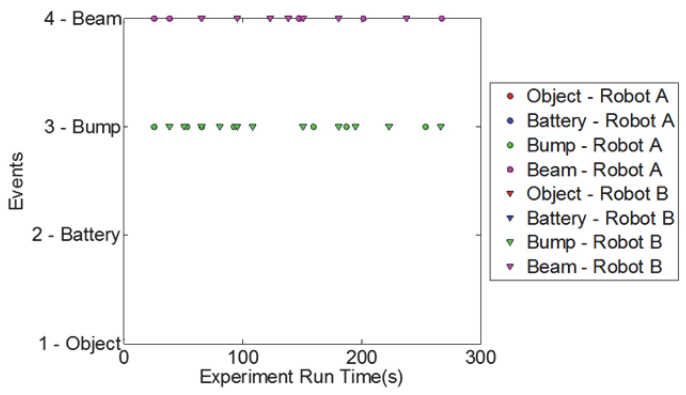

Figure 4.31 Events during the robot motion (Two Robots, Risk-Taking and Risk-Taking)

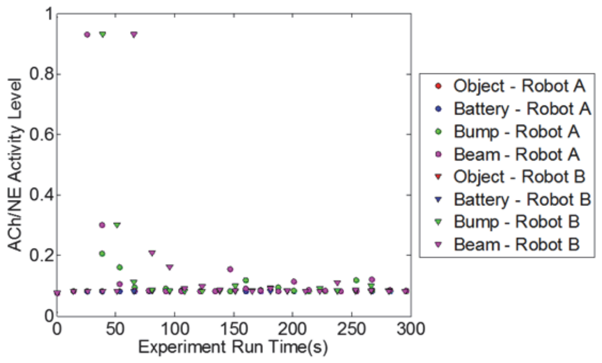

Figure 4.32 Activity of ACh/NE during the robot motion (Two Robots, Risk-Taking and Risk-Taking)

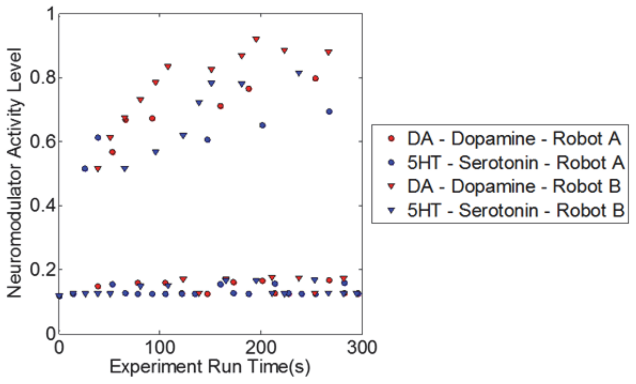

Figure 4.33 The activity of DA and 5-HT neurons during the robot motion (Two Robots, Risk-Taking and Risk-Taking)

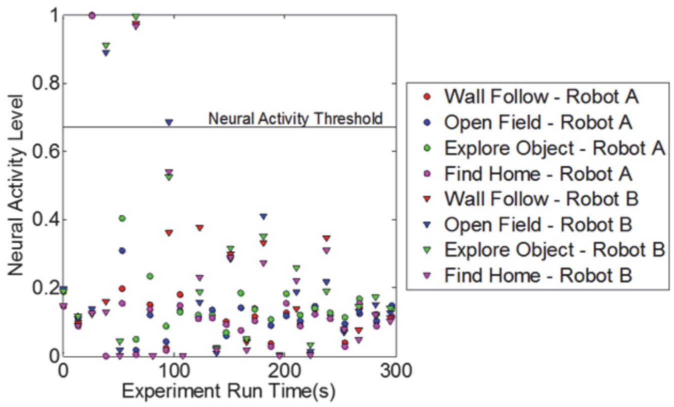

Figure 4.34 Activity of state neurons greater than threshold (0.67) (Two Robots, Risk-Taking and Risk-Taking)

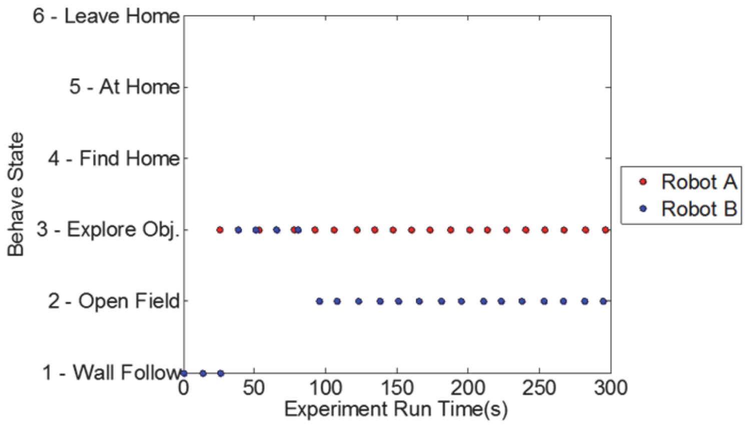

Figure 4.35 State transition during the robot motion (Two Robots, Risk-Taking and Risk-Taking)

4.2.5 Risk Taking Robot with Distracted Robot

Results of one robot in risk-taking mode and the other in distracted mode are presented in Figure 4.36 through Figure 4.40. With one Turtlebot in risk-taking mode (Robot A) and one robot in distracted mode (Robot B), the neural simulation played out with Robot A occasionally following Robot B. As seen in Figure 4.40, both robots went into Explore Object with Robot B switching quicker. The focusing on objects with higher activity levels for the ACh/NE neurons of Robot A was evident as demonstrated in Figure 4.37.

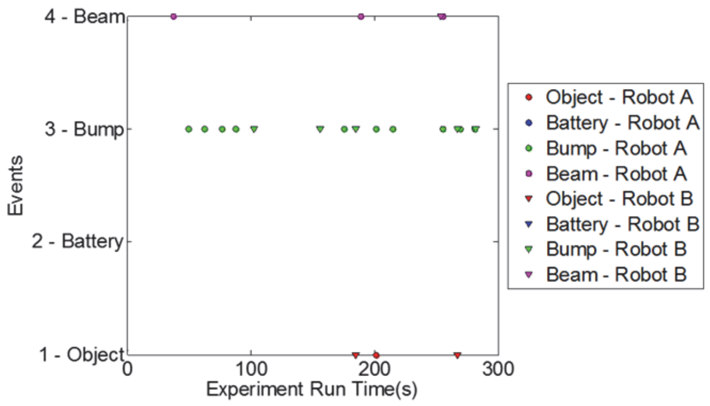

Figure 4.36 Events during the robot motion (Two Robots, Risk-Taking and Distracted)

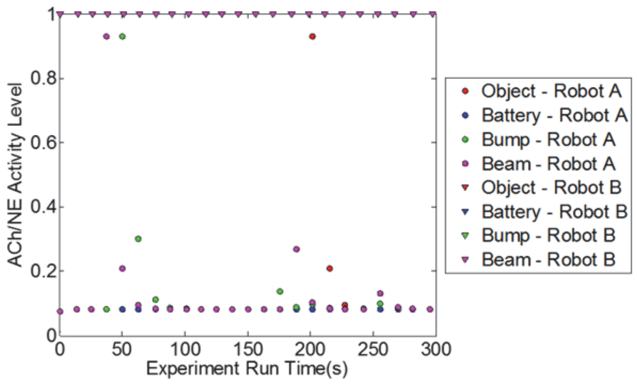

Figure 4.37 Activity of ACh/NE during the robot motion (Two Robots, Risk-Taking and Distracted)

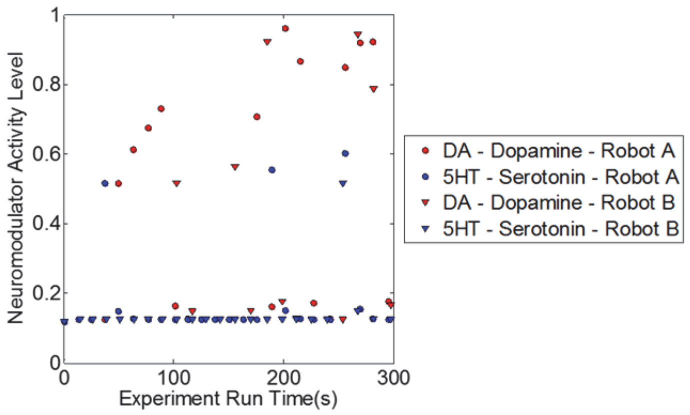

Figure 4.38 The activity of DA and 5-HT neurons during the robot motion (Two Robots, Risk-Taking and Distracted)

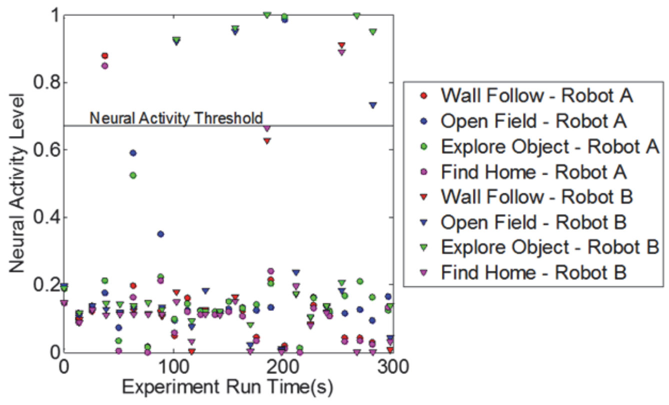

Figure 4.39 Activity of state neurons greater than threshold (0.67) (Two Robots, Risk-Taking and Distracted)

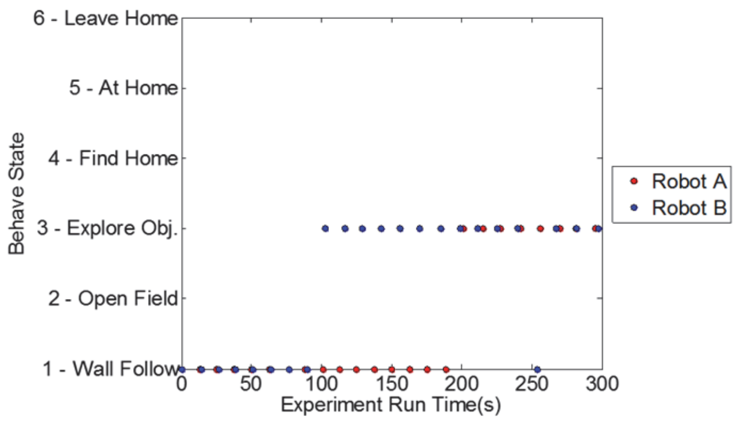

Figure 4.40 State transition during the robot motion (Two Robots, Risk-Taking and Distracted)

4.3 Simple Neuron Model (Three Robots)

The procedure of the two-robot experiments were repeated but with one additional Turtlebot added in the room near the first two when the runs started. Each Turtlebot used the same neuromodulation code but in combination of various modes. Due to the large number of possible combinations only a few are included as representative cases. Under each heading Robot A and Robot B were run in the first mode listed, while Robot C was run in the second mode. So for instance, for two Risk Aversive Robots/One Risk Taking, Robot A and Robot B were in Risk Aversive mode and Robot C was in Risk Taking Mode.

4.3.1 Two Risk Aversive Robots with One Risk Taking Robot

Results are presented in Figure 4.41 through Figure 4.45. As expected from Risk Aversive robots (Robot A and Robot B) there was an abundance of serotonin compared to the amount of dopamine (Figure 4.43) and vice versa for the Risk Taking robot (Robot C). The constant contact with the home base beam (Figure 4.41) led Robot B towards the Find Home state transitions (Figure 4.45). Robot A occasionally broke away from its serotonergic based Wall Following actions to follow Robot B home for a few seconds. Robot C constantly followed either Robot A or Robot B across the room.

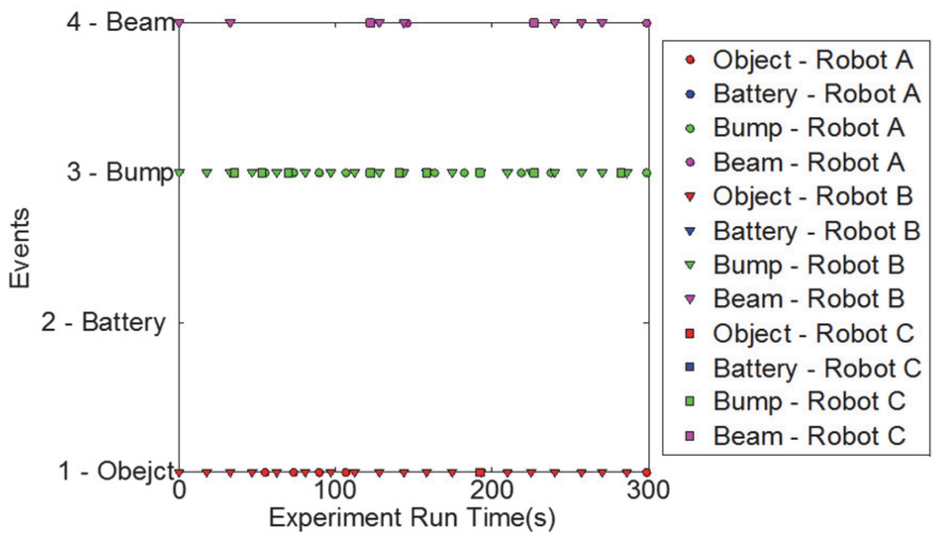

Figure 4.41 Events during the robot motion (Three Robots, Two Risk Aversive/One Risk Taking)

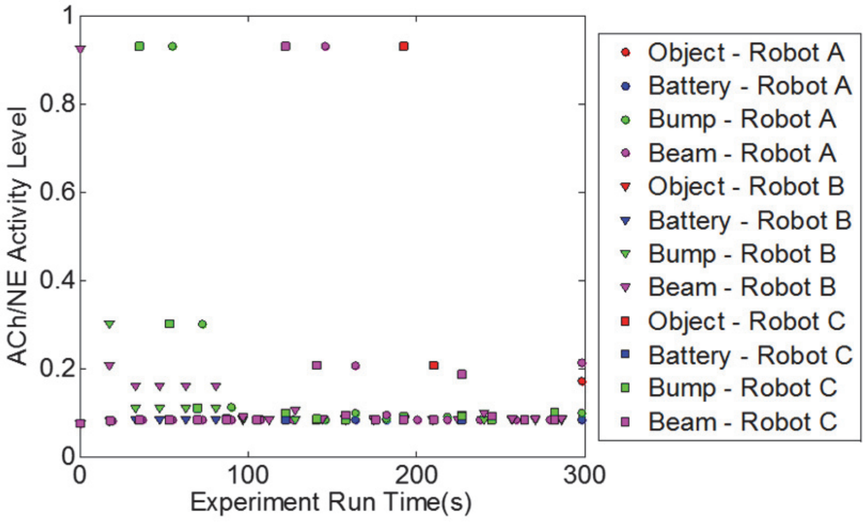

Figure 4.42 Activity of ACh/NE during the robot motion (Three Robots, Two Risk Aversive/One Risk Taking)

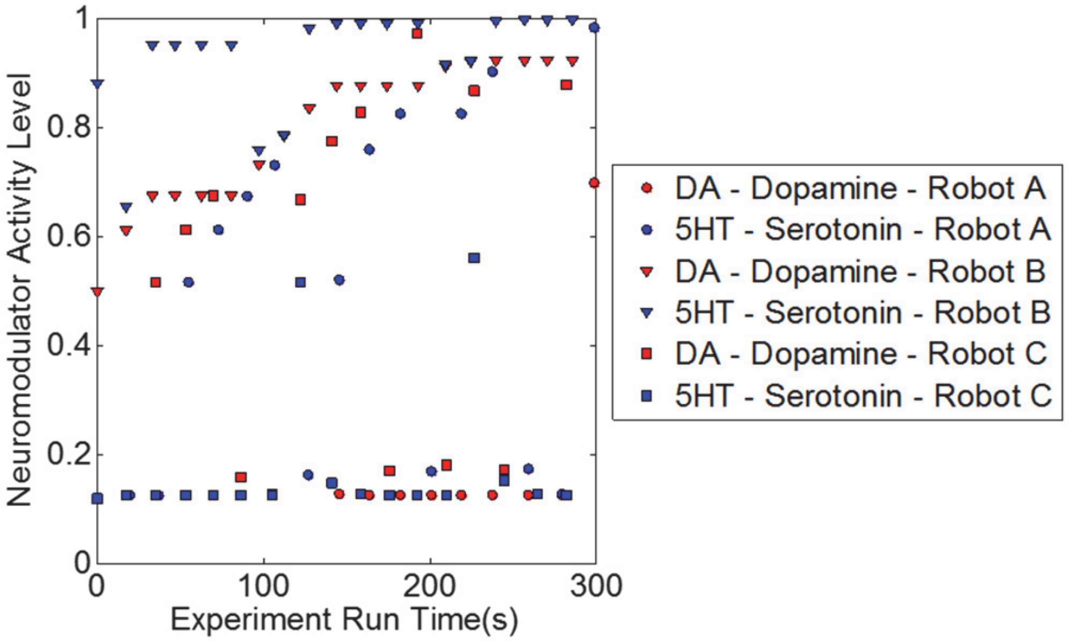

Figure 4.43 The activity of DA and 5-HT neurons during the robot motion (Three Robots, Two Risk Aversive/One Risk Taking)

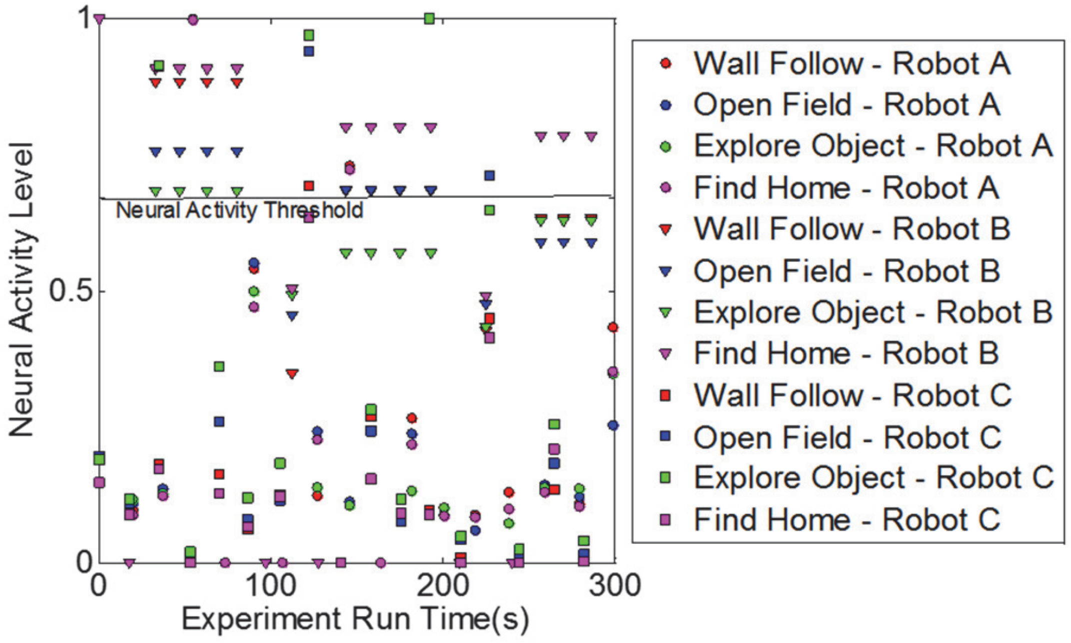

Figure 4.44 Activity of state neurons greater than threshold (0.67) (Three Robots, Two Risk Aversive/One Risk Taking)

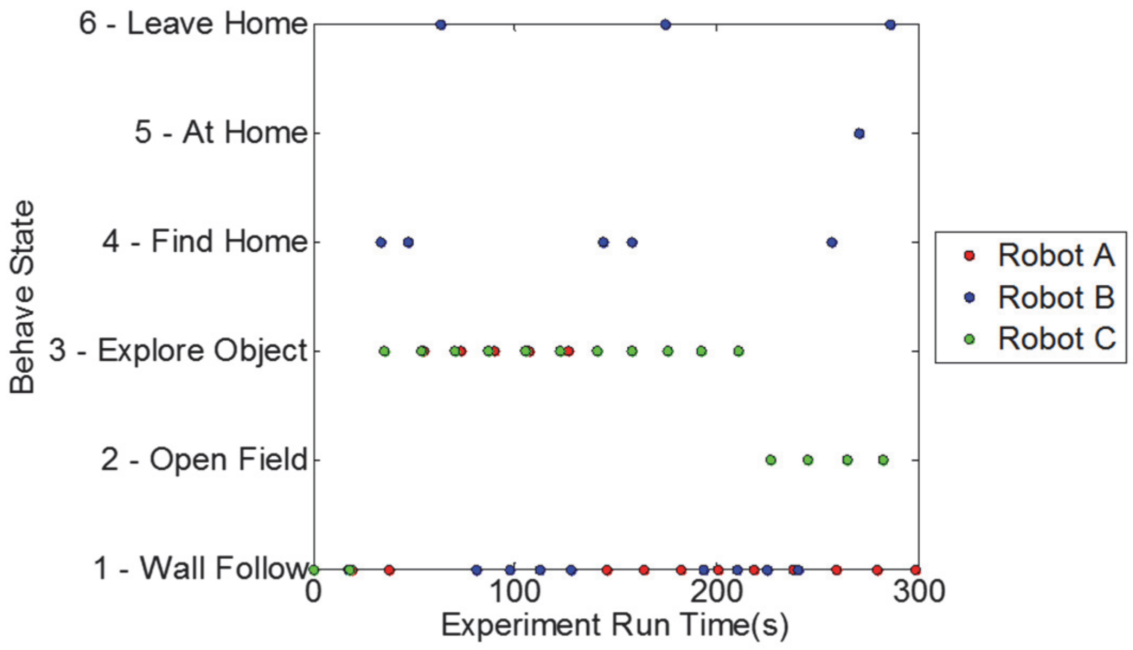

Figure 4.45 State transition during the robot motion (Three Robots, Two Risk Aversive/One Risk Taking)

4.3.2 Two Risk Taking Robots with One Risk Aversive Robot

Results are presented in Figure 4.46 through Figure 4.50. The neural activity for this case was shown to be the exact opposite of the case before. The reactions and actions of Robot C were mostly serotonergic in nature and the reactions/actions of Robot A were mostly dopaminergic in nature. This can be seen in Figure 4.50 with Robot A going into exploratory modes for the entire run (reacting to dopaminergic bumps and spotting objects as seen in Figure 4.46) with Robot C looking for home in between stints of Wall Following. Robot B, the second risk taking robot in the run, wandered off into an open area, therefore never receiving the necessary stimuli to break out the default Wall Follow state.

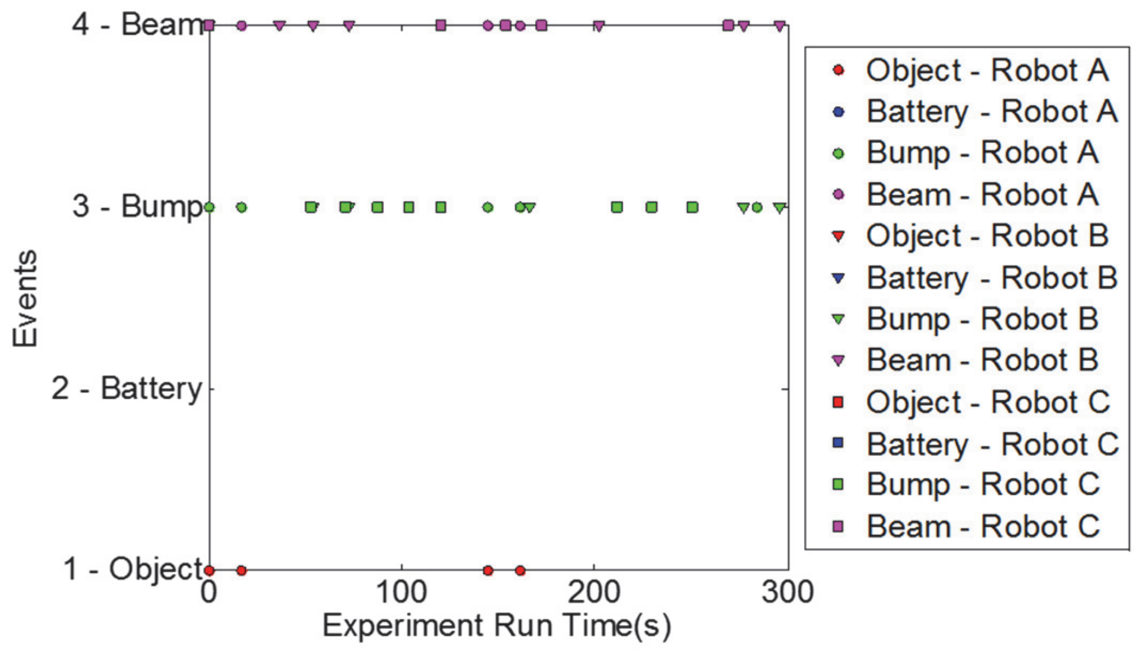

Figure 4.46 Events during the robot motion (Three Robots, Two Risk Taking/One Risk Aversive)

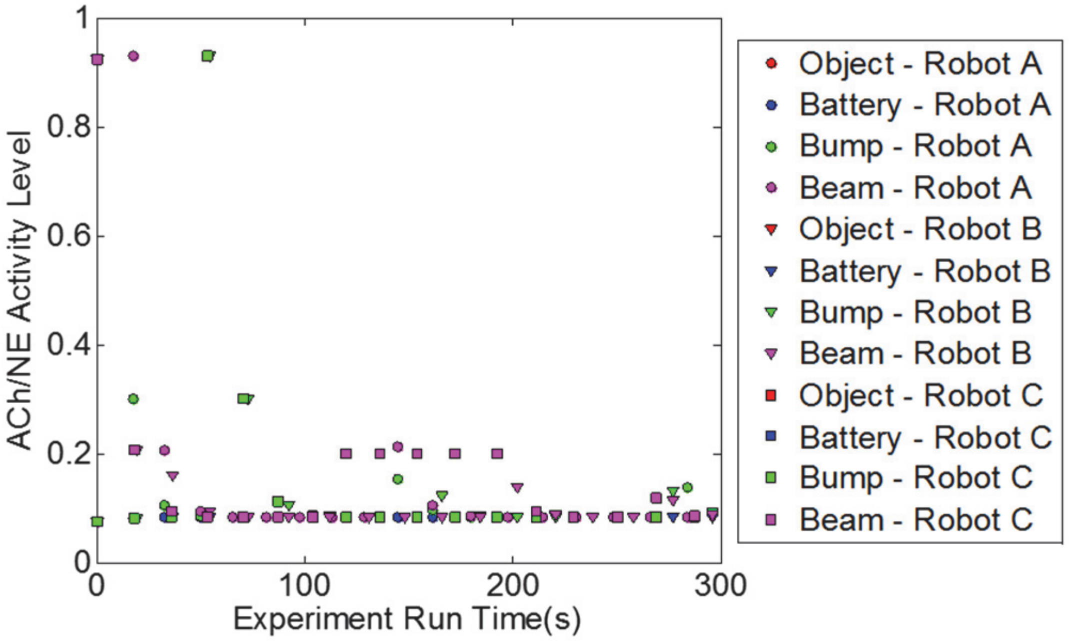

Figure 4.47 Activity of ACh/NE during the robot motion (Three Robots, Two Risk Taking/One Risk Aversive)

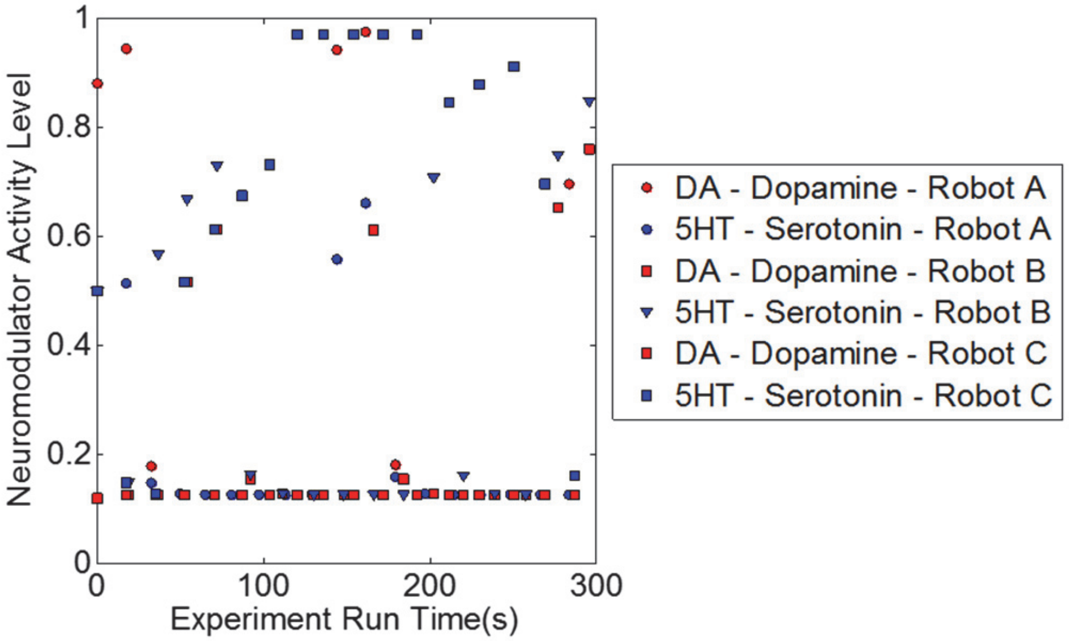

Figure 4.48 The activity of DA and 5-HT neurons during the robot motion (Three Robots, Two Risk Taking/One Risk Aversive)

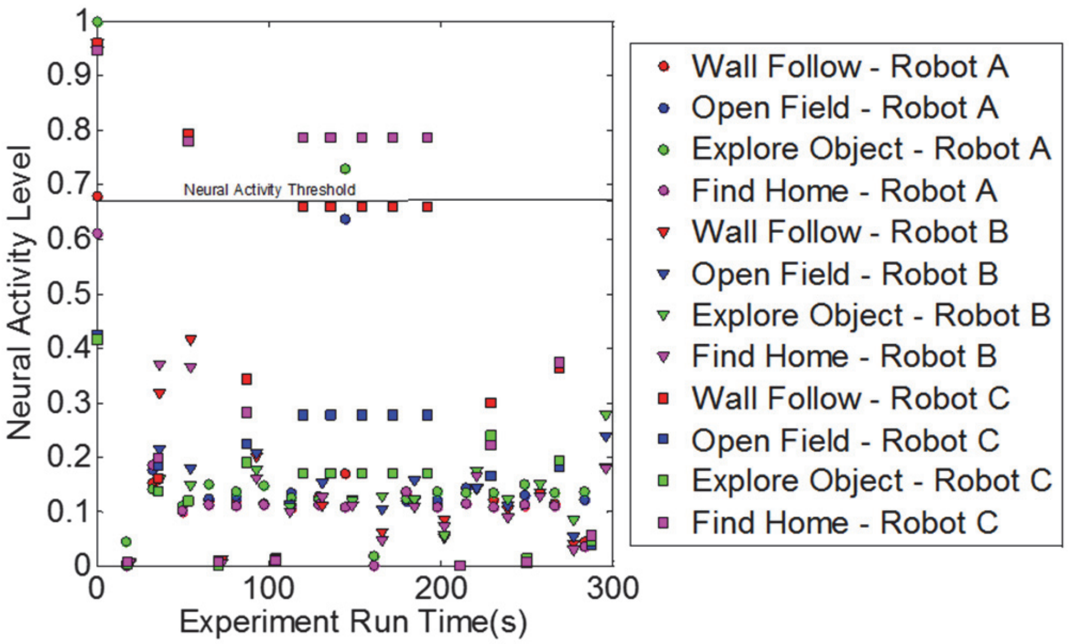

Figure 4.49 Activity of state neurons greater than threshold (0.67) (Three Robots, Two Risk Taking/One Risk Aversive)

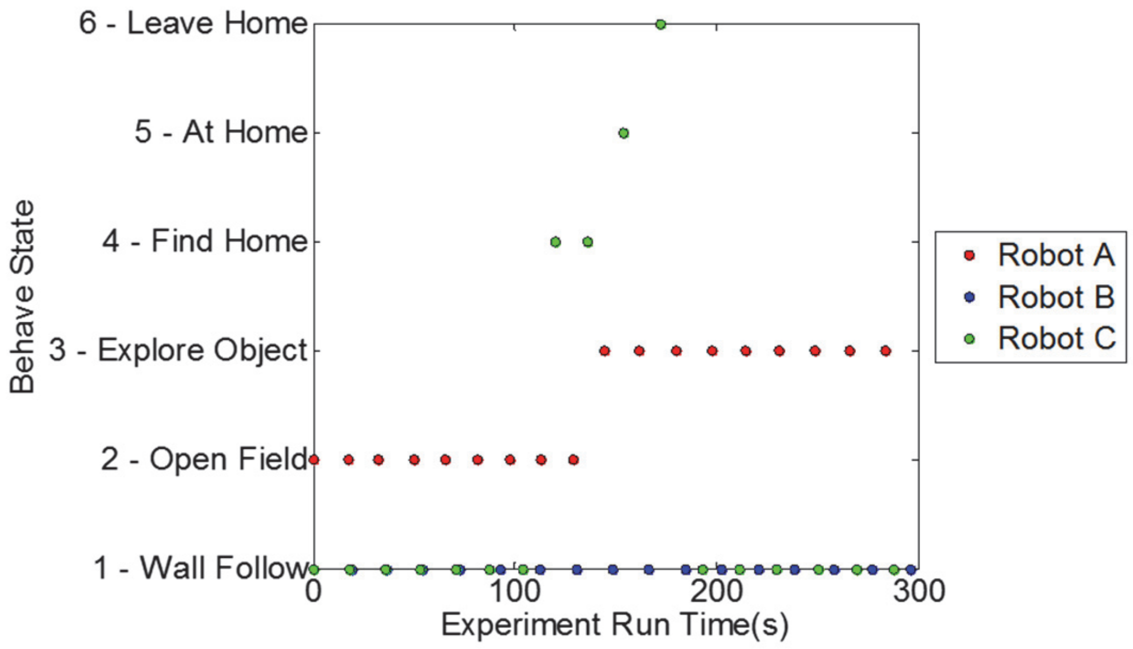

Figure 4.50 State transition during the robot motion (Three Robots, Two Risk Taking/One Risk Aversive)

4.4 Simple Neuron Model (Four Robots)

The format of the single-robot experiments were repeated but with four Turtlebots in the room for each run. Each Turtlebot used the same neuromodulation code but in combination of various modes. As with the three-robot case, due to the large amount of possible combinations only a few are included in this report. Under each heading Robot A, B, C, D were run in the various modes as listed respectively. For instance, for Two Risk Taking Robots/One Risk Aversive Robot/One Distracted Robot, Robots A and B were in Risk Taking mode, Robot C was in Risk Aversive mode, and Robot D was in Distracted mode.

4.4.1 Two Risk Aversive Robots with Two Risk Taking Robots

Results are presented in Figure 4.51 through 4.55. This case is similar to the Risk Aversive/Risk Taking case done with two robots. As the Risk Aversive robots (Robots A and B) searched for the home base (as seen in the events in Figure 4.51), the Risk Taking robots (Robots C and D) occasionally took notice and followed (Figure 4.55). As expected, Robots A and B had mostly serotonergic reactions and Robots C and D had mostly dopaminergic reactions (Figure 4.53).

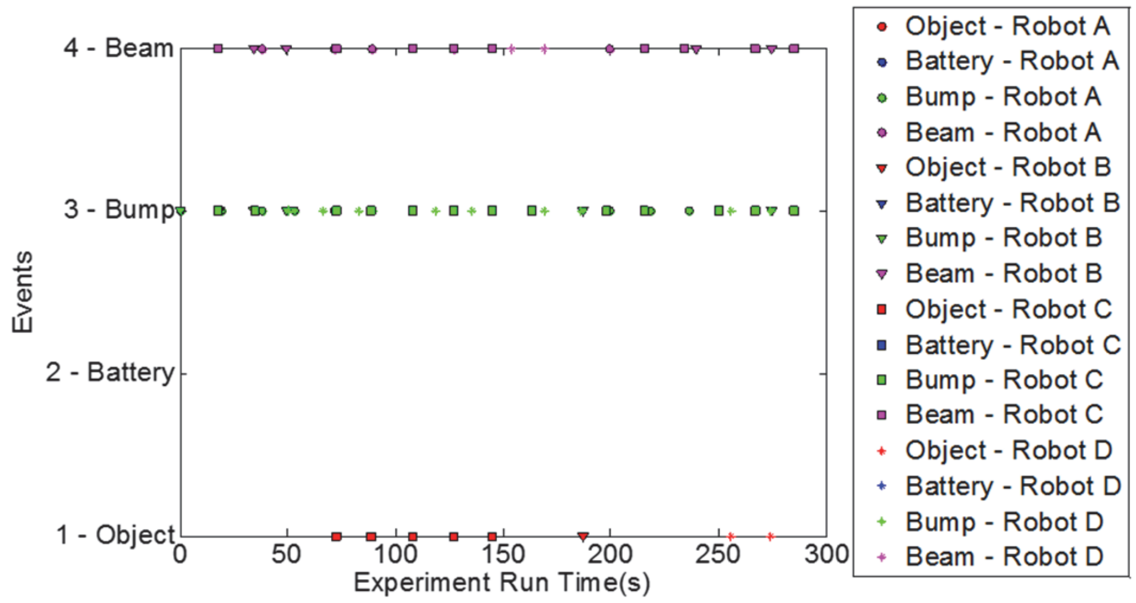

Figure 4.51 Events during the robot motion (Four Robots, Two Risk Aversive/Two Risk Taking)

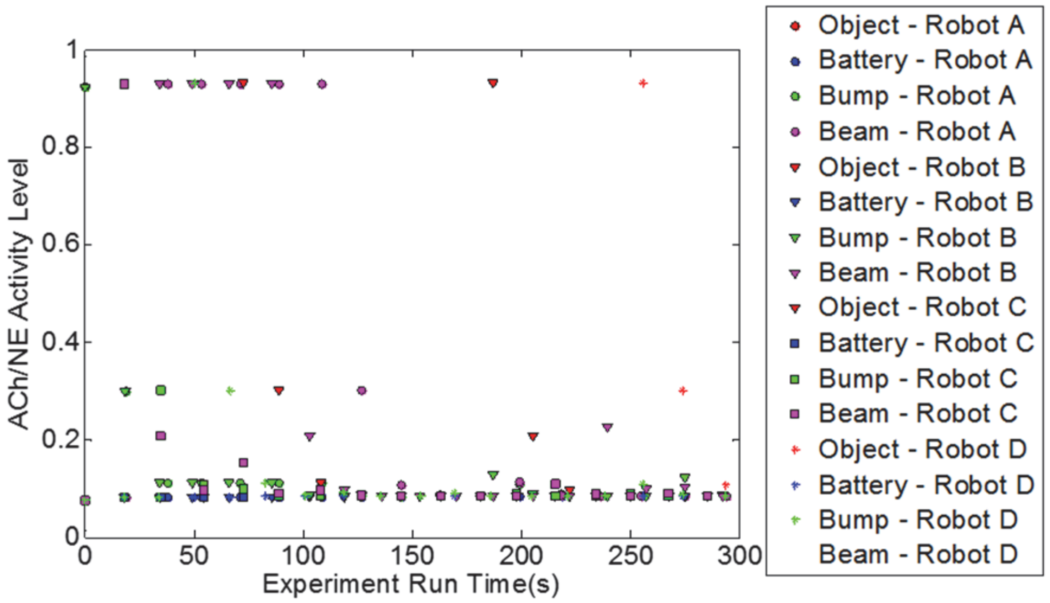

Figure 4.52 Activity of ACh/NE during the robot motion (Four Robots, Two Risk Aversive/Two Risk Taking)

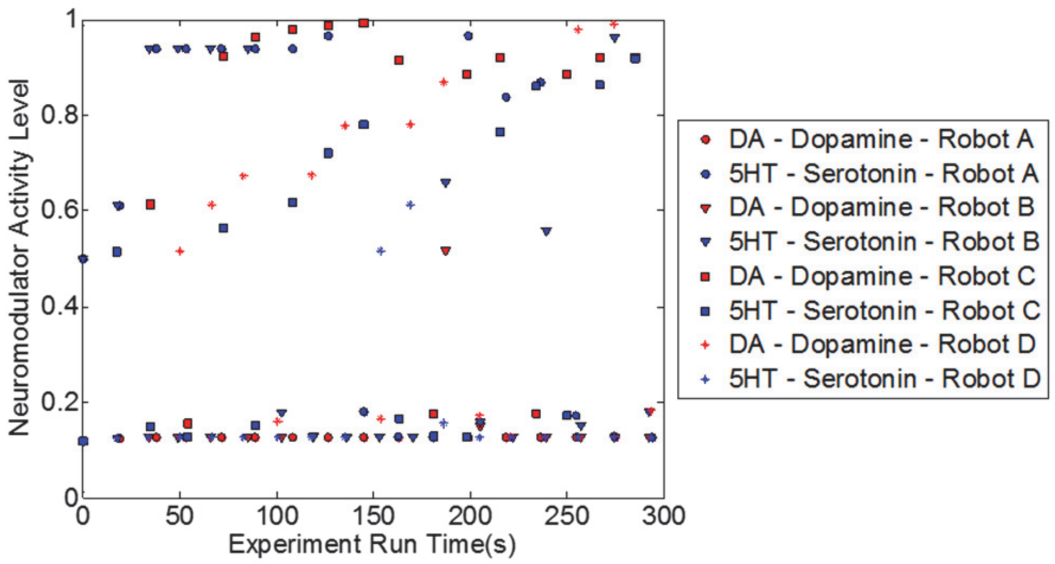

Figure 4.53 The activity of DA and 5-HT neurons during the robot motion (Four Robots, Two Risk Aversive/Two Risk Taking)

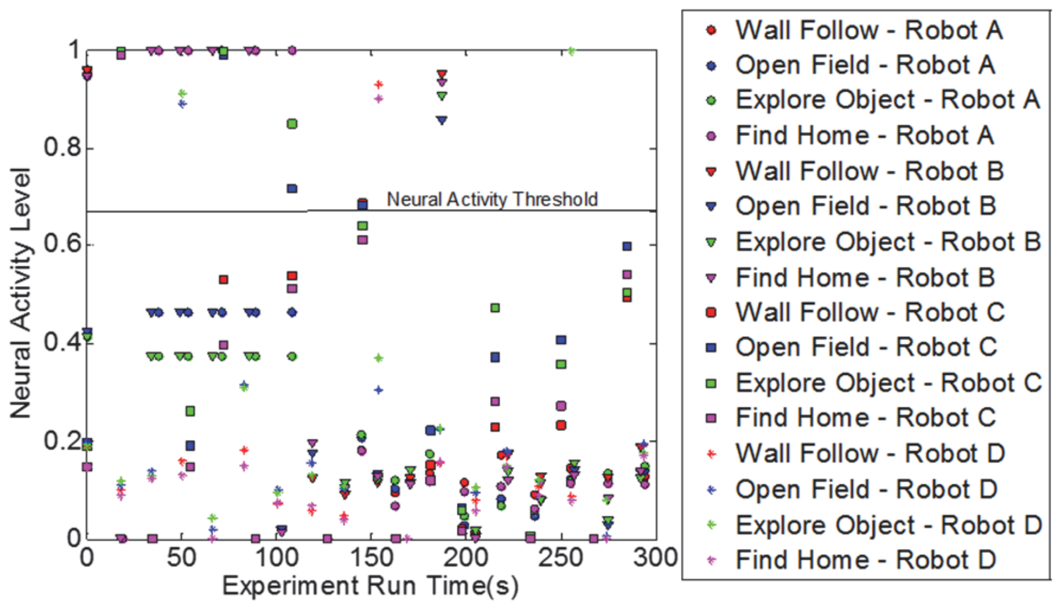

Figure 4.54 Activity of state neurons greater than threshold (0.67) (Four Robots, Two Risk Aversive/Two Risk Taking)

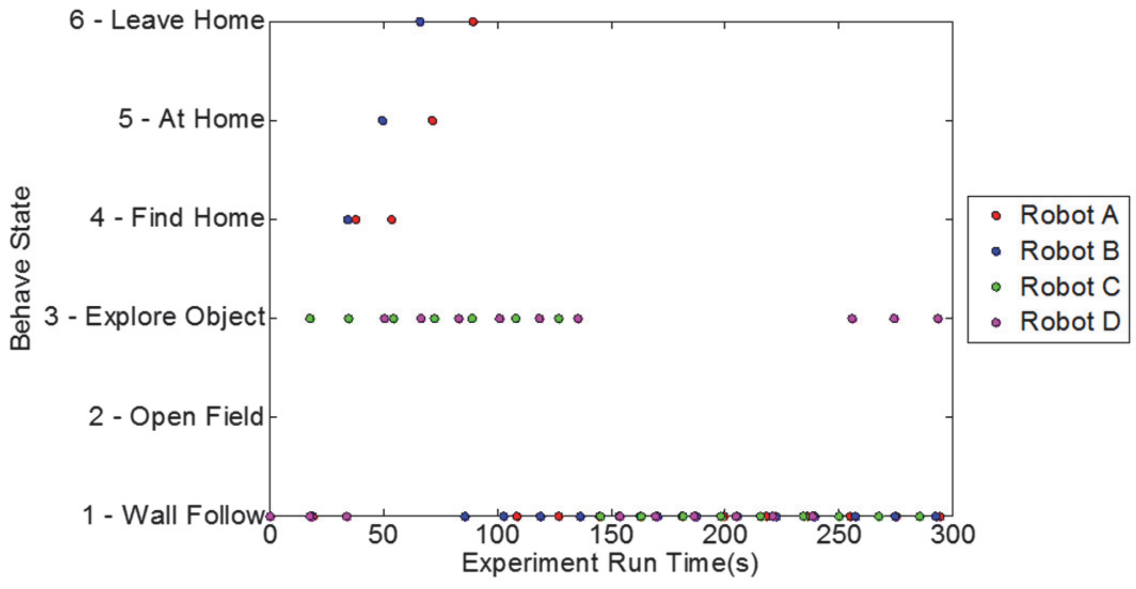

Figure 4.55 State transition during the robot motion (Four Robots, Two Risk Aversive/Two Risk Taking)

4.4.2. Two Risk Taking Robots with One Risk Aversive Robot and One Distracted Robot

Results are presented in Figure 4.56 through 4.60. As in all Distracted cases with the simple neuron model, the ACh/Ne neurons were always active (Figure 4.57) and therefore the corresponding robot's (Robot D) ability to focus on a single input was impaired. This can be seen in Figure 4.59 with the rapid spiking of neurons in Robot D's network. Robots A and B (the risk taking platforms) followed the usual Risk Taking exploratory behaviors while Robot C mostly followed the wall (as expected from a risk aversive robot). Robot D mostly followed the other robots (whichever one passed through its field of vision) around the room during its distracted phase.

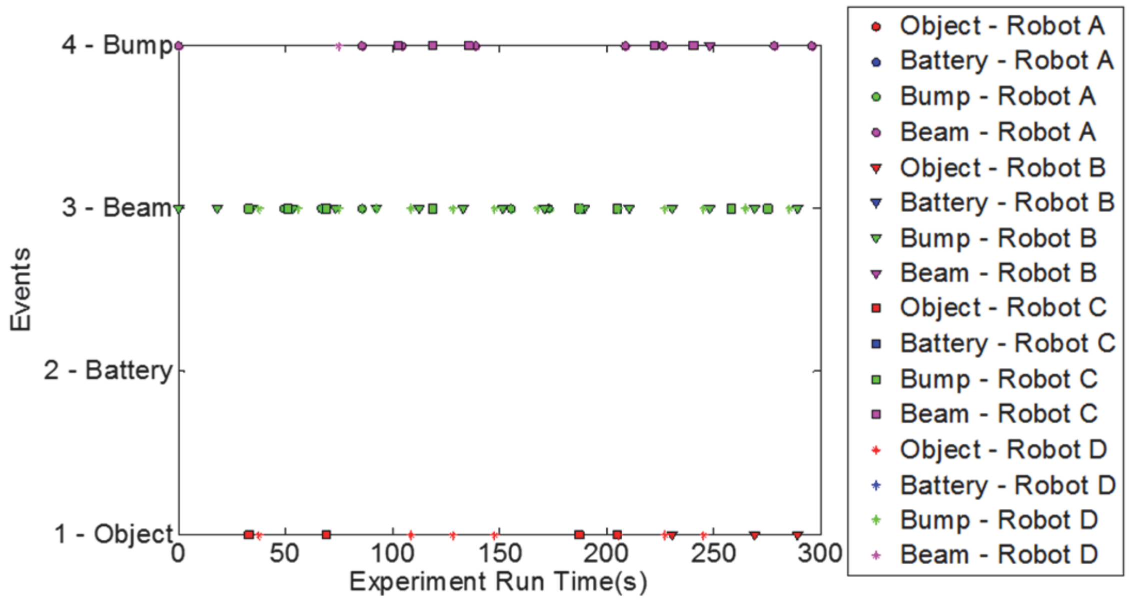

Figure 4.56 Events during the robot motion (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

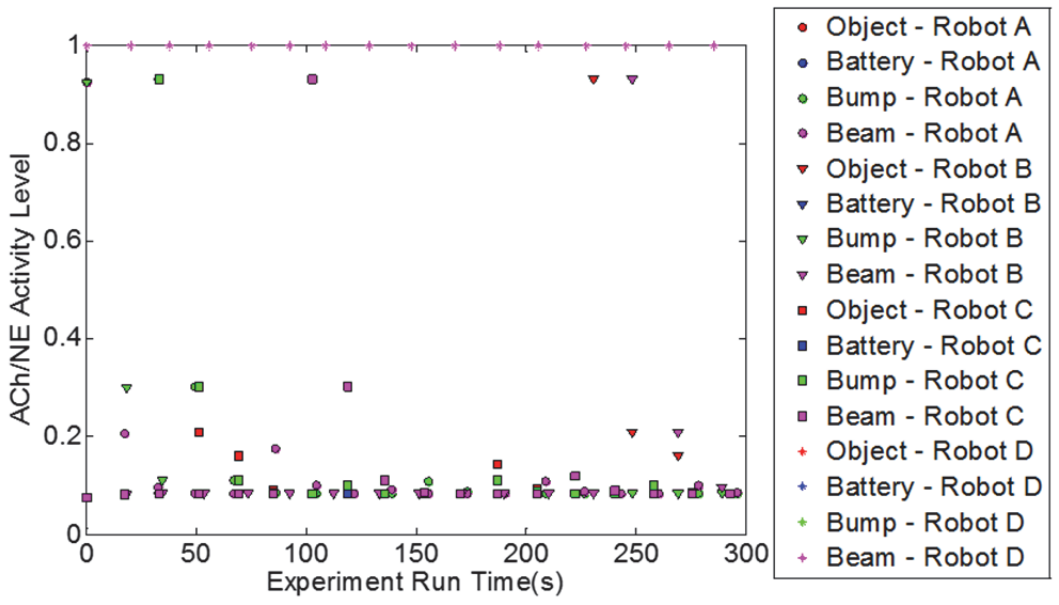

Figure 4.57 Activity of ACh/NE during the robot motion (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

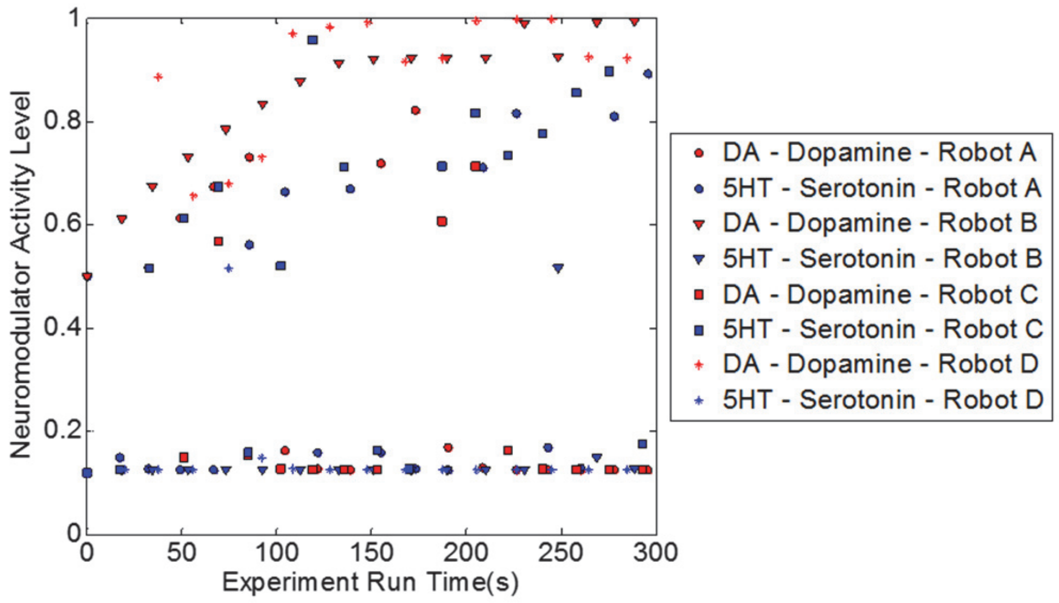

Figure 4.58 The activity of DA and 5-HT neurons during the robot motion) (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

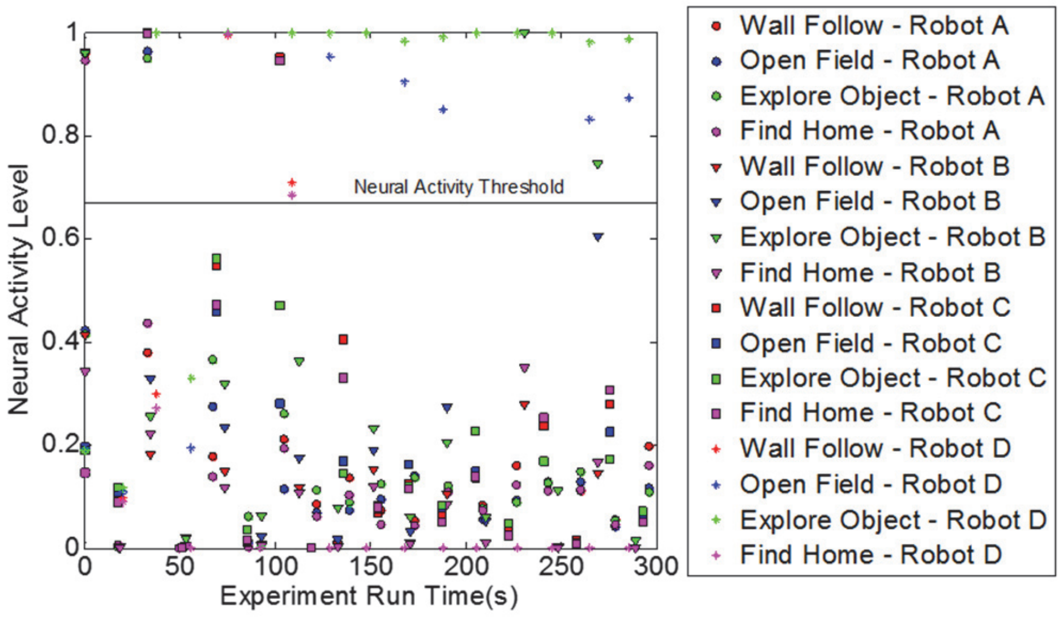

Figure 4.59 Activity of state neurons greater than threshold (0.67) (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

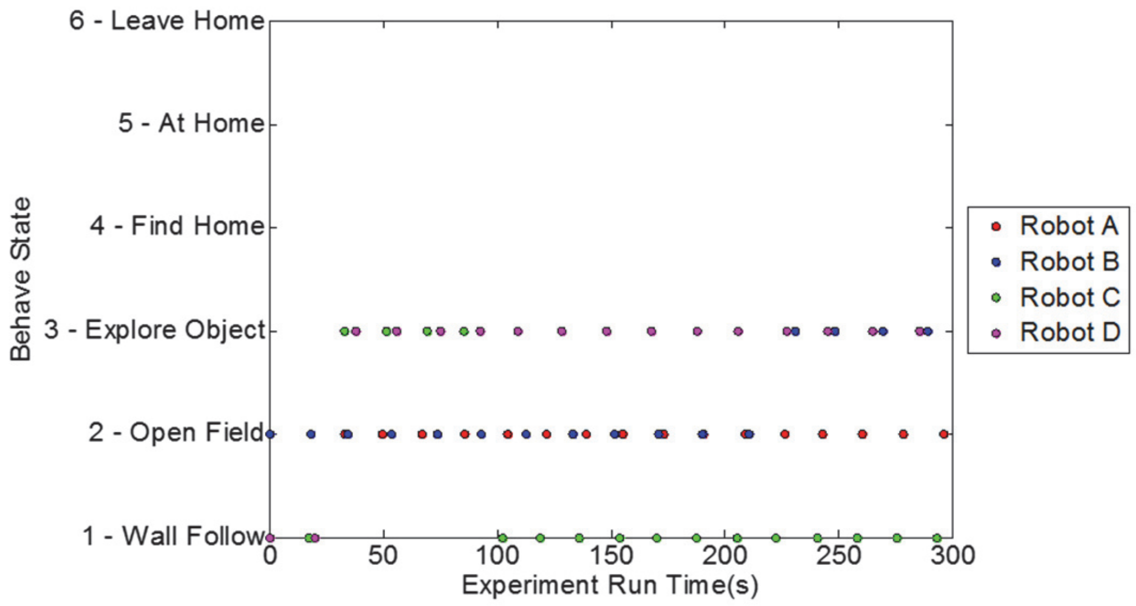

Figure 4.60 State transition during the robot motion (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

4.3 Spiking Neuron Model (Single Robot)

The spiking neuron (SNN) model was tested to see the effects of different event inputs on the neural activity in terms of number of spikes generated. Results are presented in Figure 4.61 through Figure 4.66. As can be seen in these figures, the neural activity spiked as soon as the simulator was activated and then died off as the event input no longer was novel. This was due to the effects of ACh/NE neurons on the neural network. As spiking became less active the number of neurons considered to be "winning" actions was very low. As a result, the simulation in this study was limited to one second.

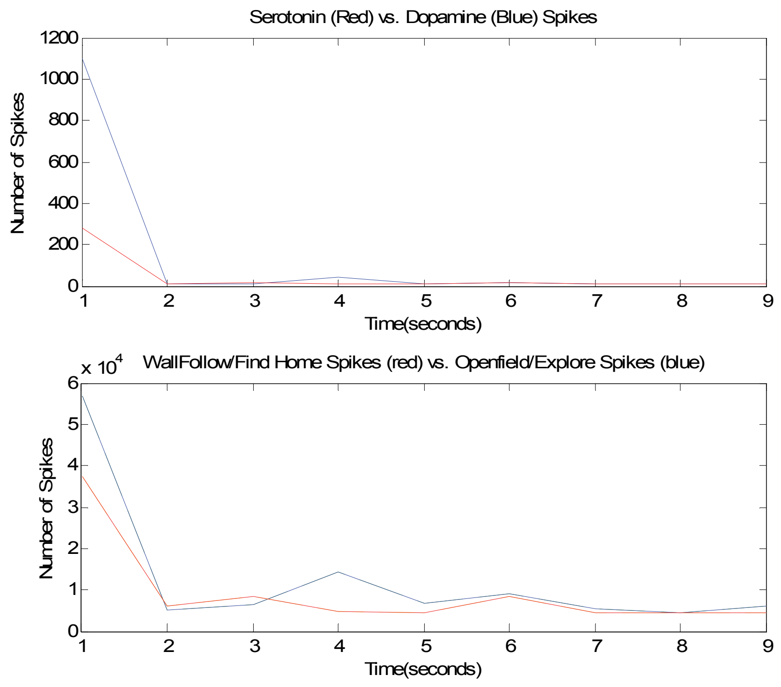

Figure 4.61 Result of Object Detected input into the SNN network

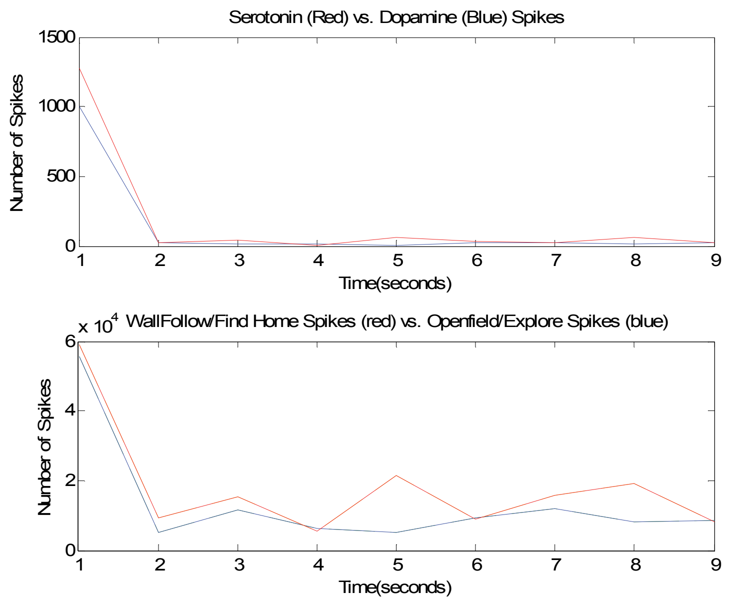

Figure 4.62 Result of Dock Beam input into the SNN network

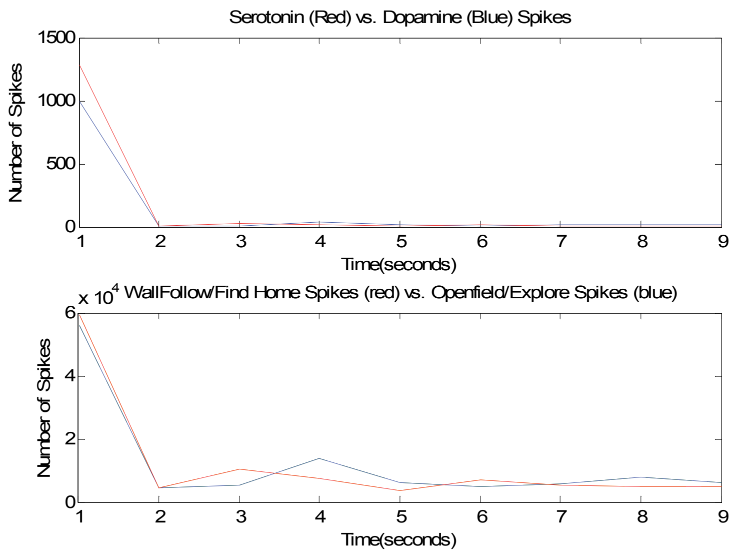

Figure 4.63 Result of Low Battery input into the SNN network

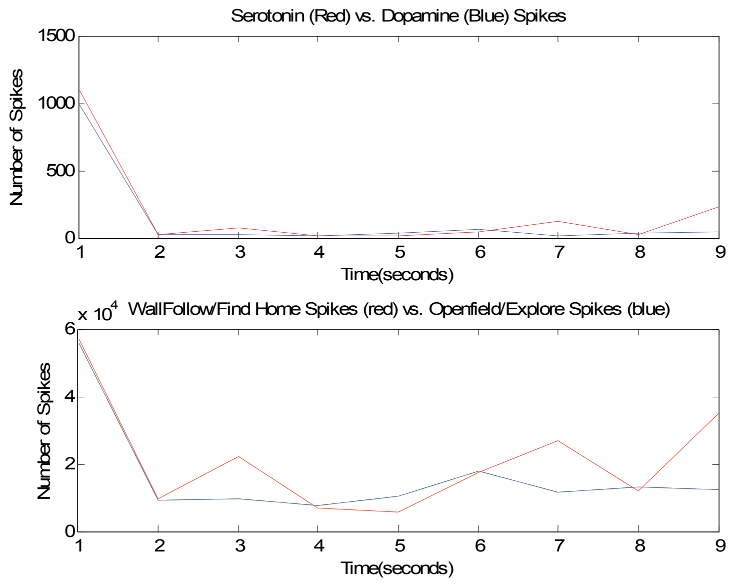

Figure 4.64 Result of Bump input (serotonin and dopamine) into the SNN network

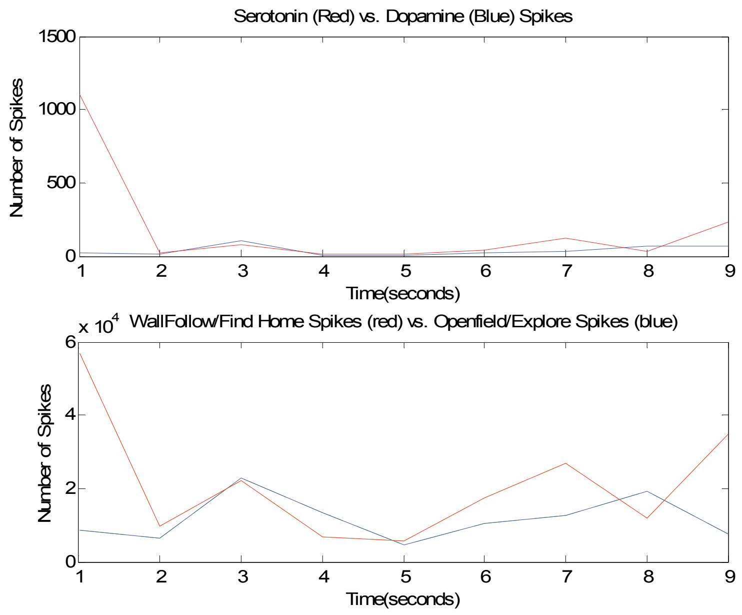

Figure 4.65 Result of Bump input (serotonin) into the SNN network

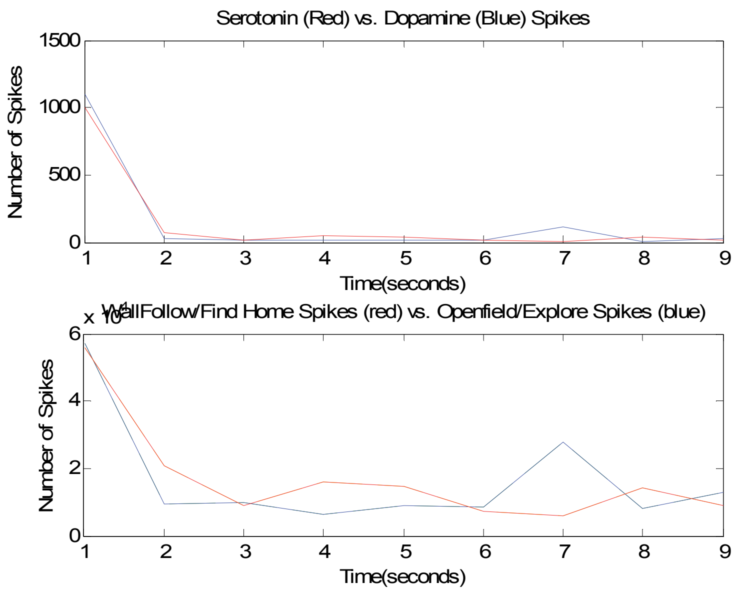

Figure 4.66 Result of Bump input (dopamine) into the SNN network

The experiments of single robot were repeated replacing the simple neuron model with the SNN model run on GPU instead of the on-board laptop. However, the laptop was used for acquisition of sensor data along with transmission of data and commands between the robot base and the remote computing node through wireless communication in ROS environment. Due to the simulator not being based on neuromodulatory levels of 0 to 1 scale, the computation and the basis of decision-making were shifted to the number of spikes in the network for each group.

4.5.1 Risk-Aversive Robot

Results are presented in Figure 4.67 through Figure 4.69 for a single robot in risk aversive mode. In the case of a risk-aversive robot, one would expect a surplus of serotonin (Figure 4.68) as the bump sensor was connected to the serotonin group in this situation. The most constant actions were the ones of a serotonergic nature (Find Home) (Figure 4.69).

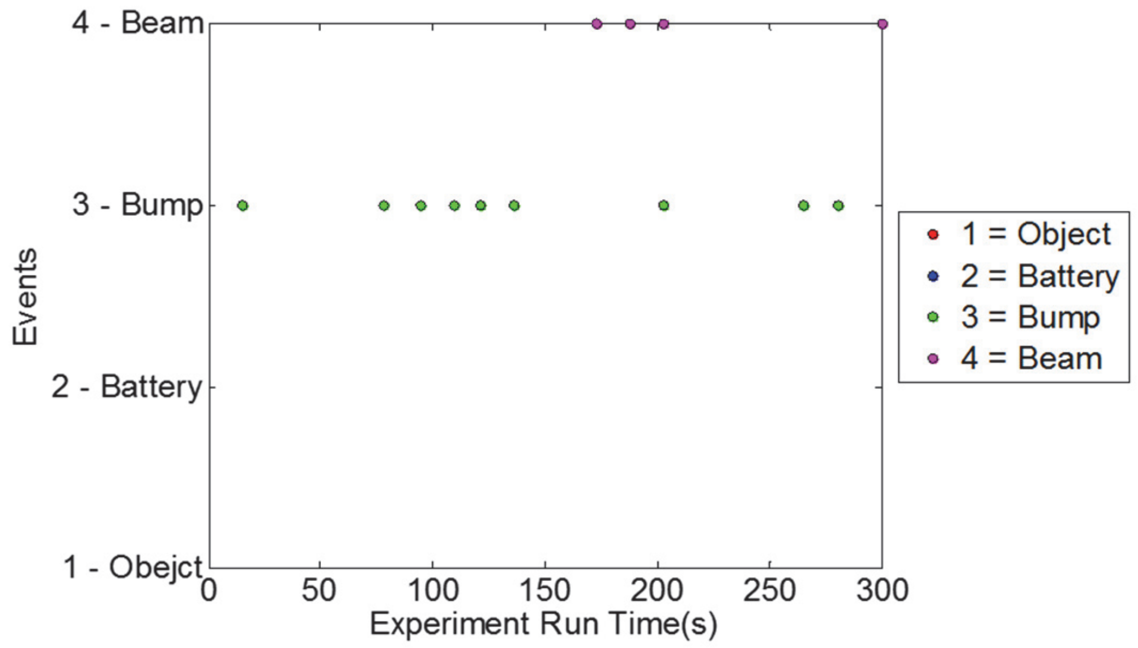

Figure 4.67 Event inputs into the ROS-SNN simulation program (Risk Aversive)

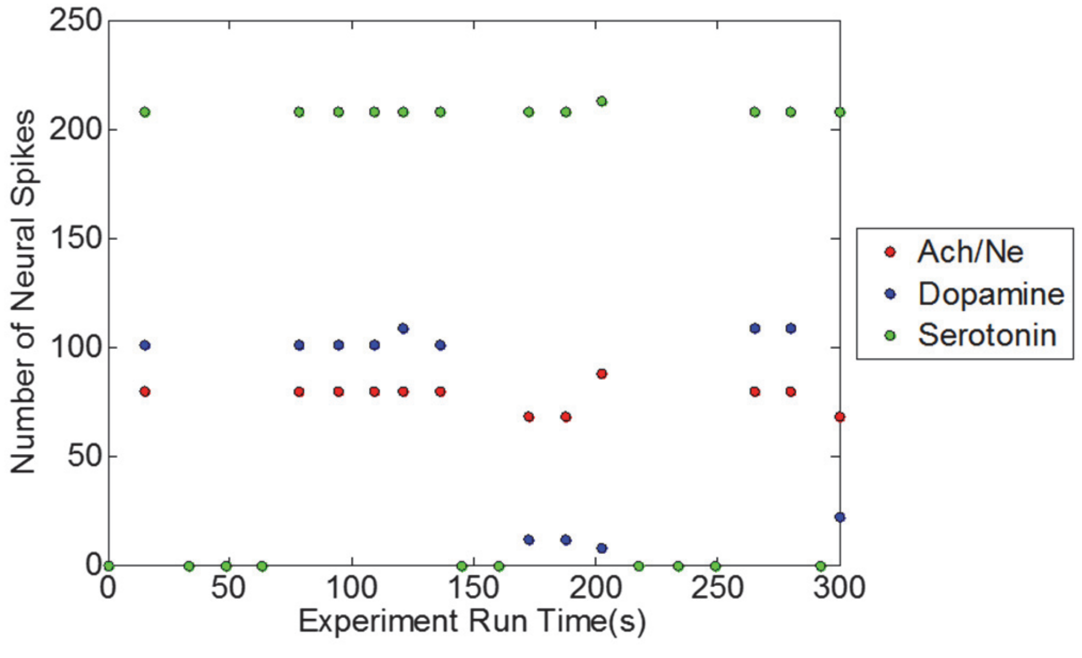

Figure 4.68 Neuromodulator activity of the ROS-SNN simulation program (Risk Aversive)

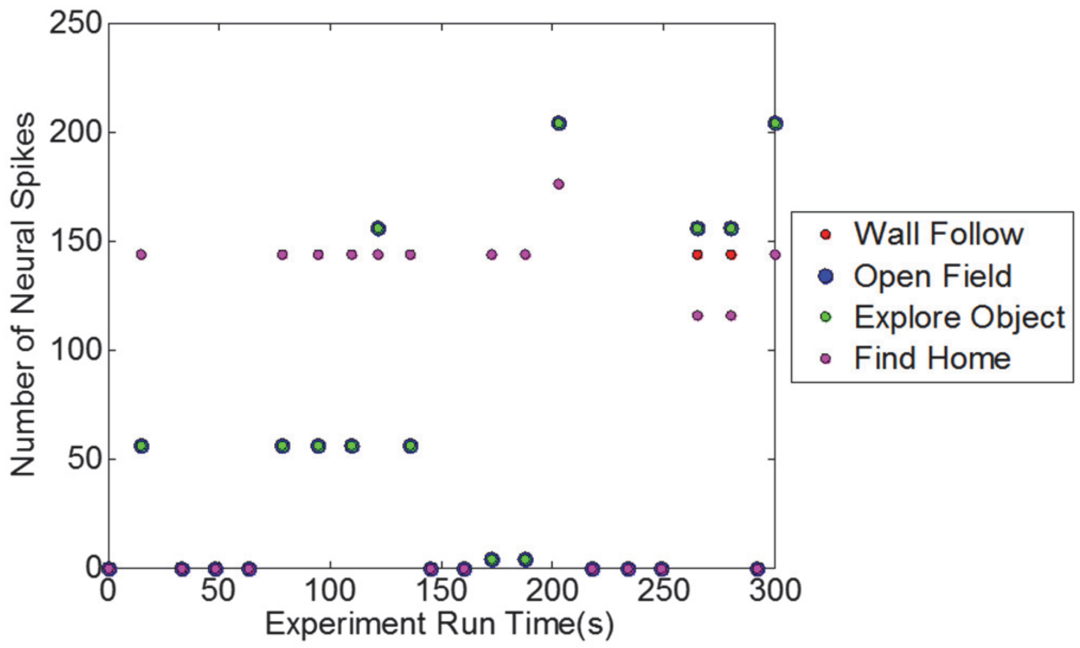

Figure 4.69 State transitions of the ROS-SNN simulation program (Risk Aversive)

4.5.2 Risk-Taking Robot

Results are presented in Figure 4.70 through Figure 4.72 for a single robot in risk taking mode. As can be seen in Figure 4.70, from around the mid-point of the 300 second run, a constant event (home beam) occurred and that led to constant level of associated neuromodulated activity (Figure 4.71). As seen in Figure 4.71, due to the low number of ACh/NE neurons the robot lost the ability to ignore the recurrent event in the short time span unlike the one in the simple neural network model. This demonstrated the downside of the SNN model without the proper tuning of the model parameters. A more biologically realistic model would require tuning of parameters for viable and stable results. This issue would be addressed in future scope of work.

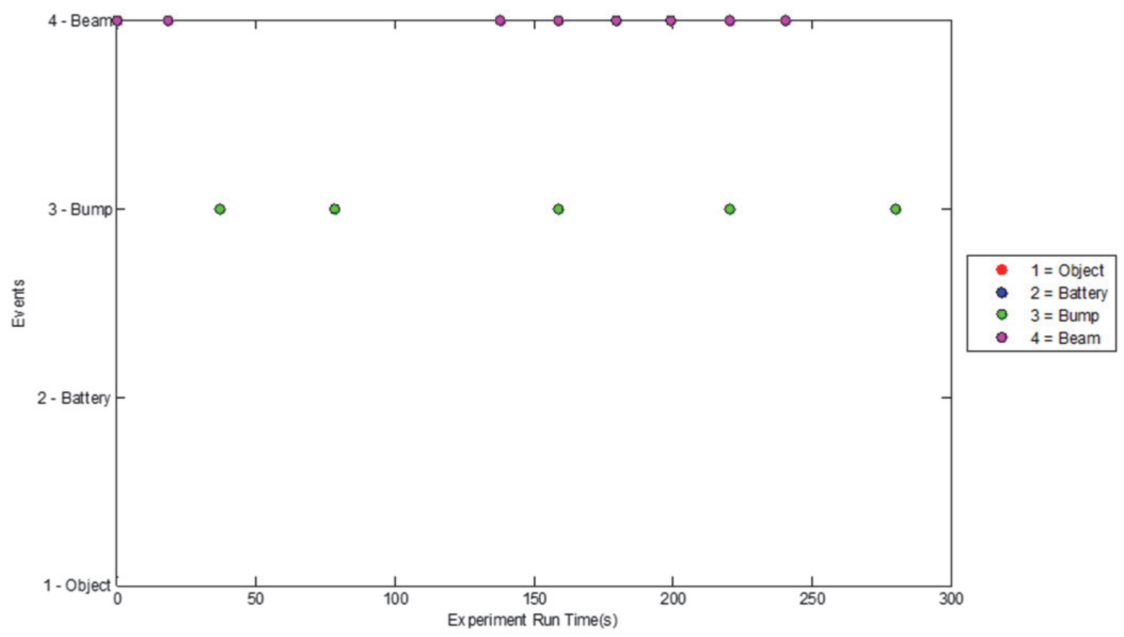

Figure 4.70 Event inputs into the ROS-SNN simulation program (Risk Taking)

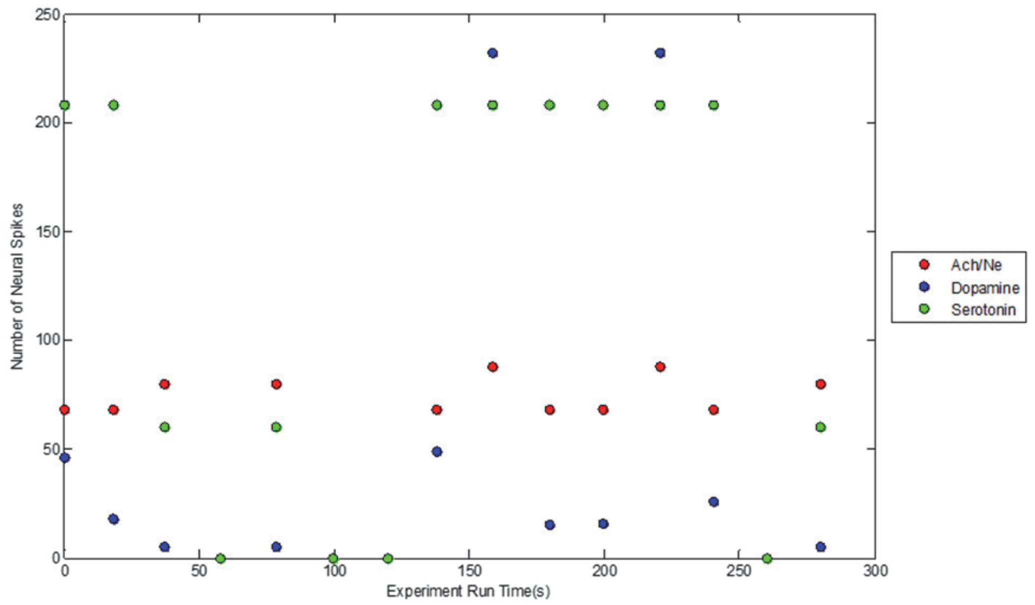

Figure 4.71 Neuromodulator activity of the ROS-SNN simulation program (Risk Taking)

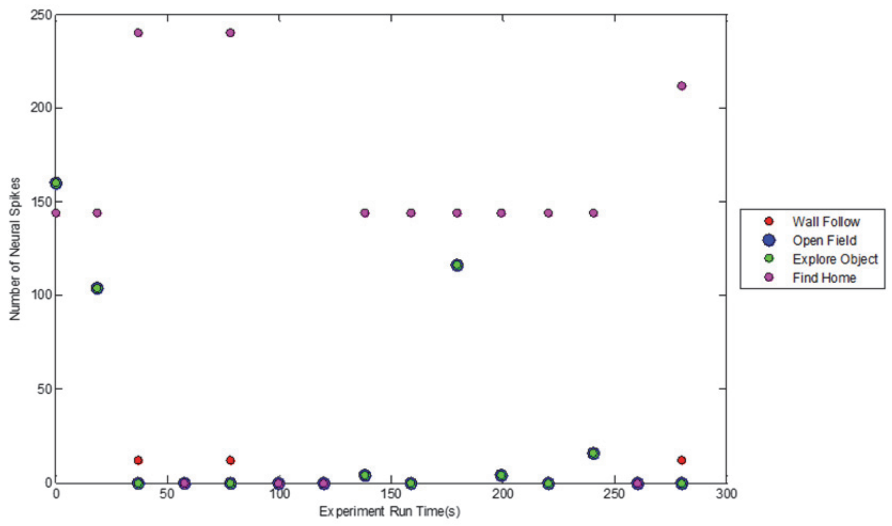

Figure 4.72 State transitions of the ROS-SNN simulation program (Risk Taking)

4.5.3 Distracted Robot

Results are presented in Figure 4.73 through Figure 4.75 for a single robot in distracted mode. In this case with the bump sensor connected to dopamine (DA) group of neurons, the activity level of DA neurons was higher throughout the run period. The ACh/NE stayed at a mostly constant level as expected in a distracted case (Figure 4.74).

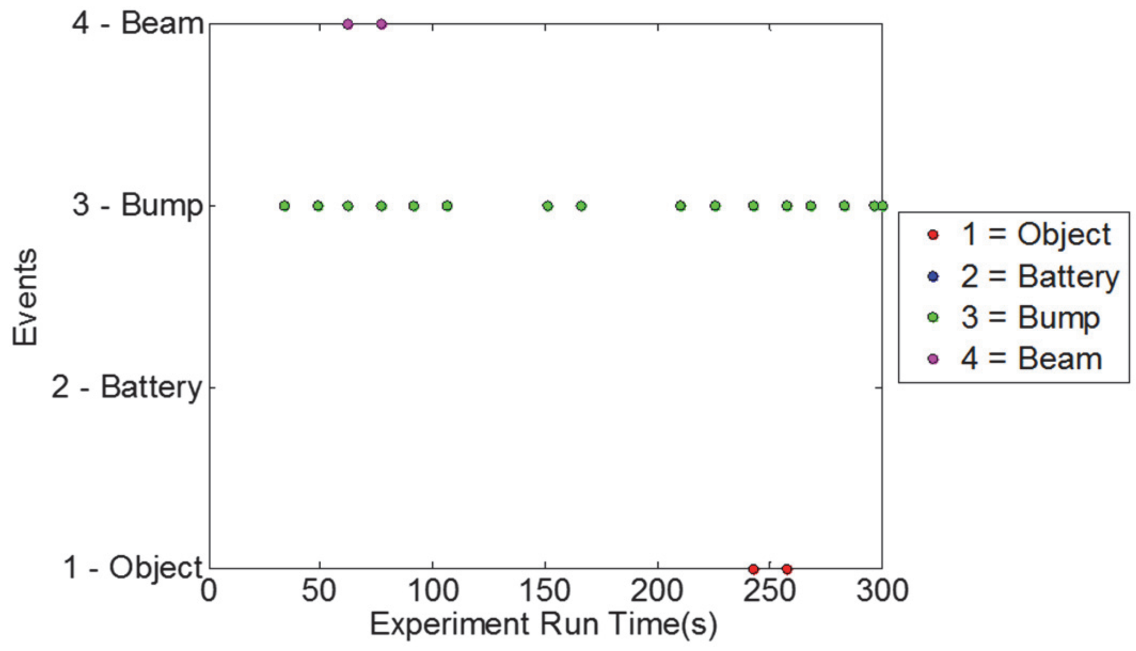

Figure 4.73 Event inputs into the ROS-SNN simulation program (Distracted)

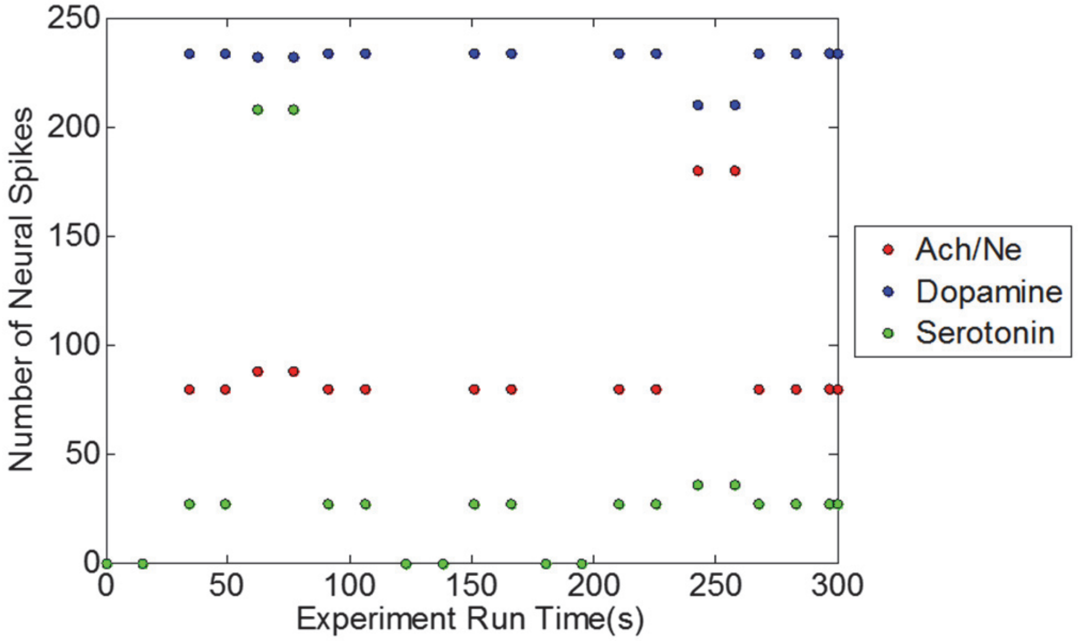

Figure 4.74 Neuromodulator activity of the ROS-SNN simulation program (Distracted)

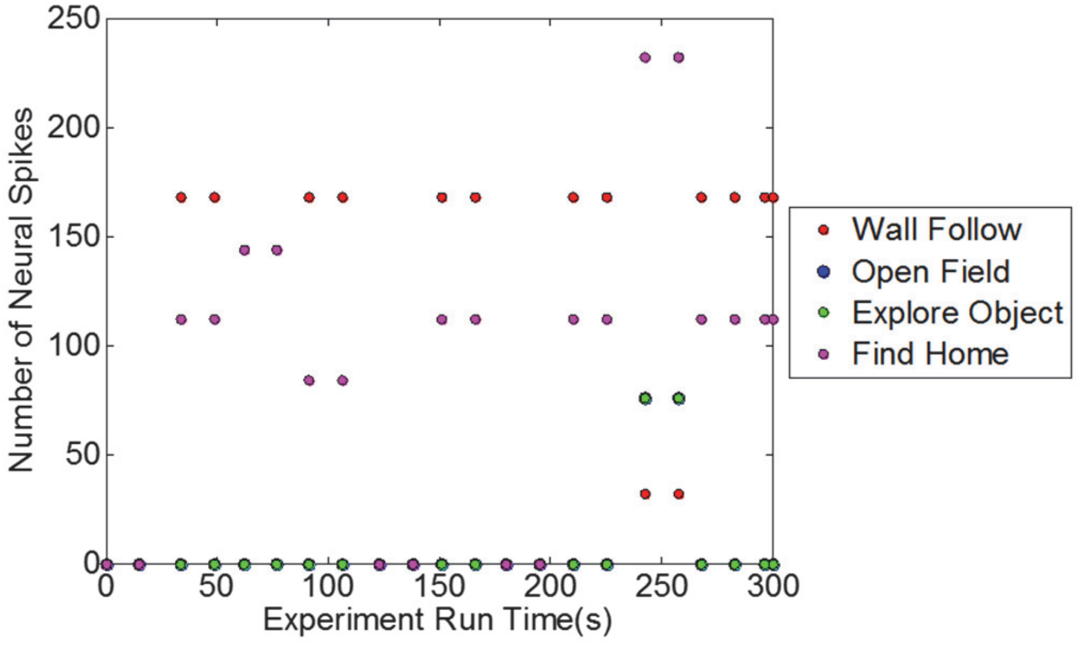

Figure 4.75 State transitions of the ROS-SNN simulation program (Distracted)

4.6 Spiking Neuron Model (Two Robots)

Similar to the case with the simple neural network, two robots were placed in the same environment and had the same group of tests as in Section 4.3. Only difference was that the simple neural network model was replaced with GPU based SNN model.

4.6.1 Risk Aversive Robot with Risk Taking Robot

Results are presented in Figure 4.76 through Figure 4.78 for two robots with Robot A in risk aversive mode and Robot B in risk taking mode. In this case the highest spiking neuromodulator for Robot A (risk aversive) and Robot B (risk taking) was serotonin and dopamine (Figure 4.77), respectively (as expected). The majority of state transitions (Figure 4.78) were based on serotonin and dopamine spike levels of Robot A and Robot B, although some of the highest spiking actions taken by Robot A tended to be dopamine based (due to the bump events connection to the dopamine group). As Robot A detected Robot B the serotonin level of Robot A increased and as Robot B detected Robot A the dopamine level of Robot B increased (Figure 4.76 and Figure 4.77).

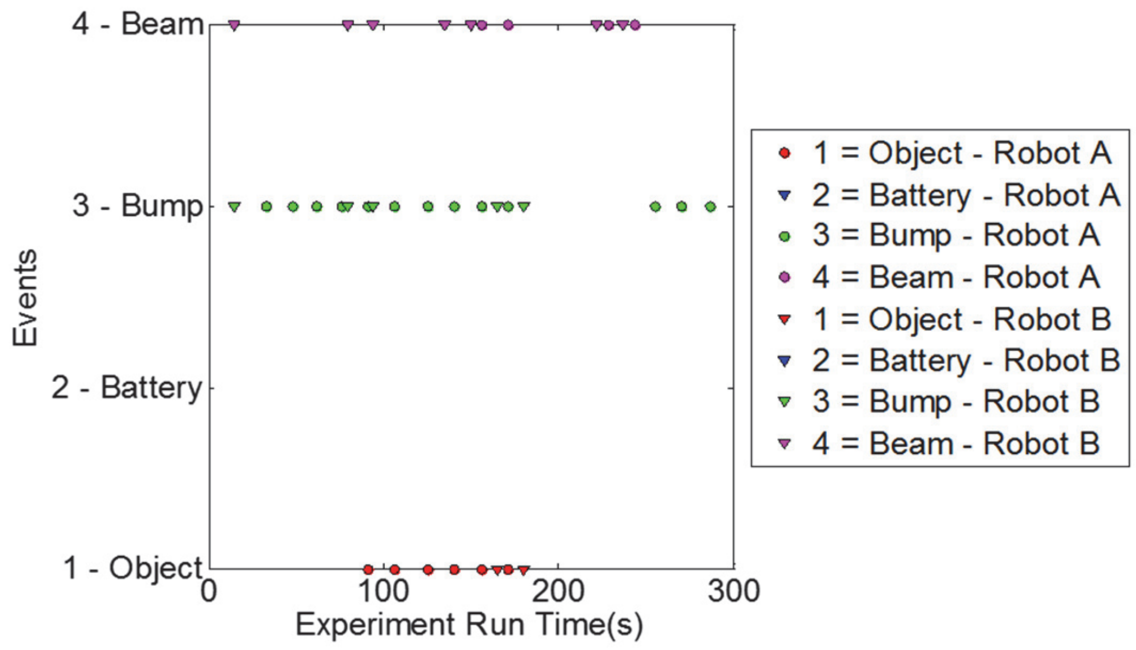

Figure 4.76 Event inputs into the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Taking)

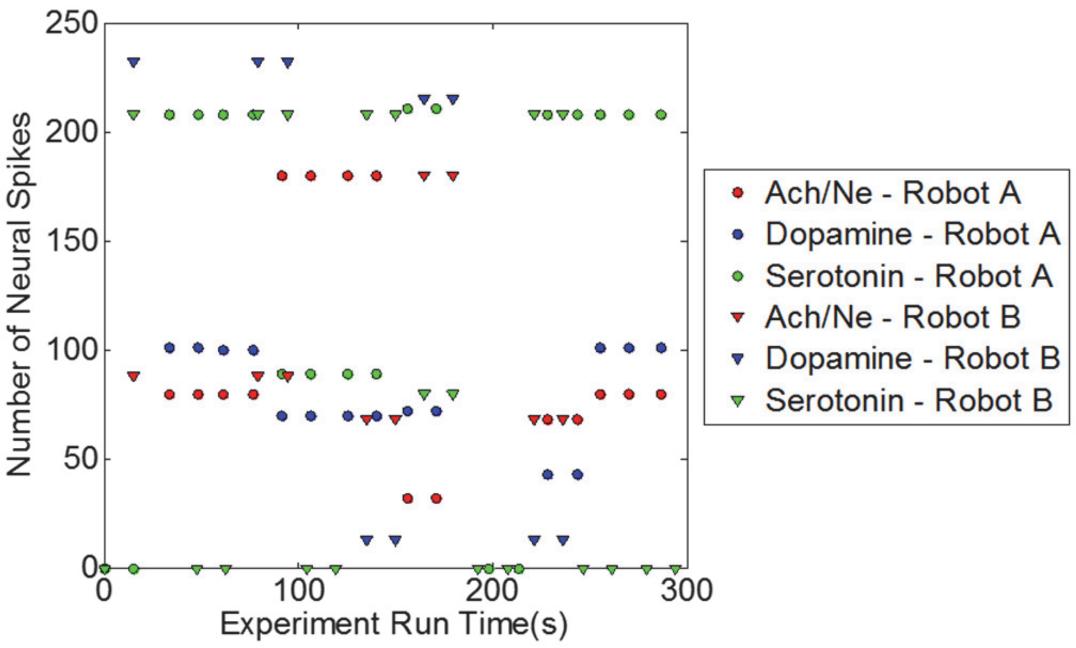

Figure 4.77 Neuromodulator activity of the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Taking)

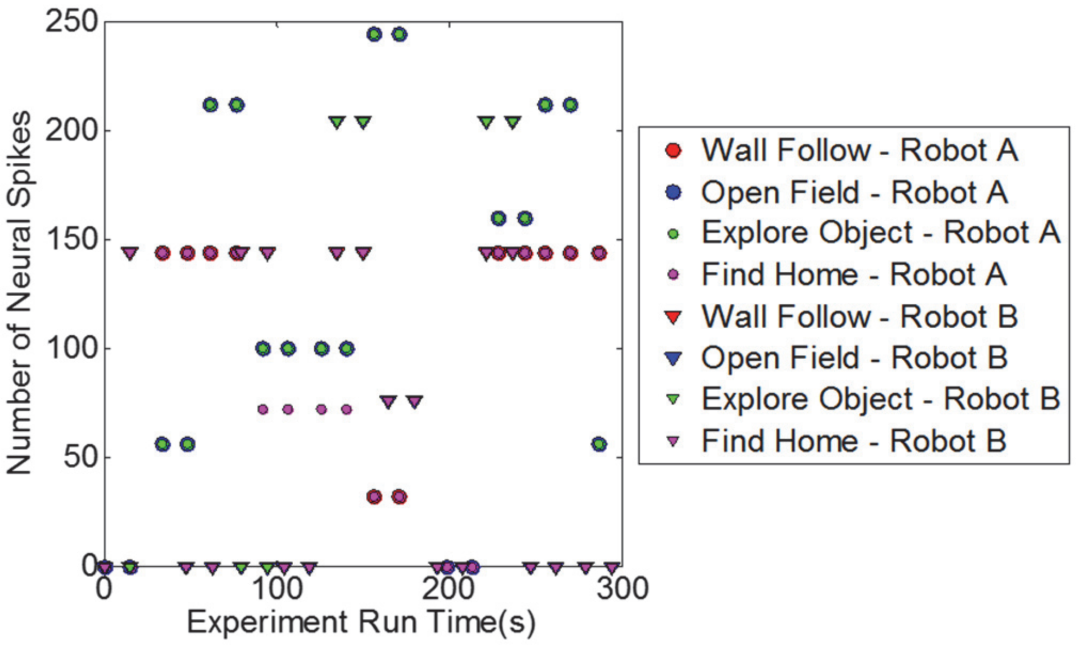

Figure 4.78 State transitions of the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Taking)

4.6.2 Risk Aversive Robot with Risk Aversive Robot

Results are presented in Figure 4.79 through Figure 4.81 for both robots in risk aversive mode. In this case the highest spiking neuromodulator (Figure 4.80) for both robots was serotonin (as expected due to the bump events connection to the serotonin group in this setup). With both robots in the risk aversive mode, there was little to no interaction between one another (as evident from no object detection in Figure 4.79).

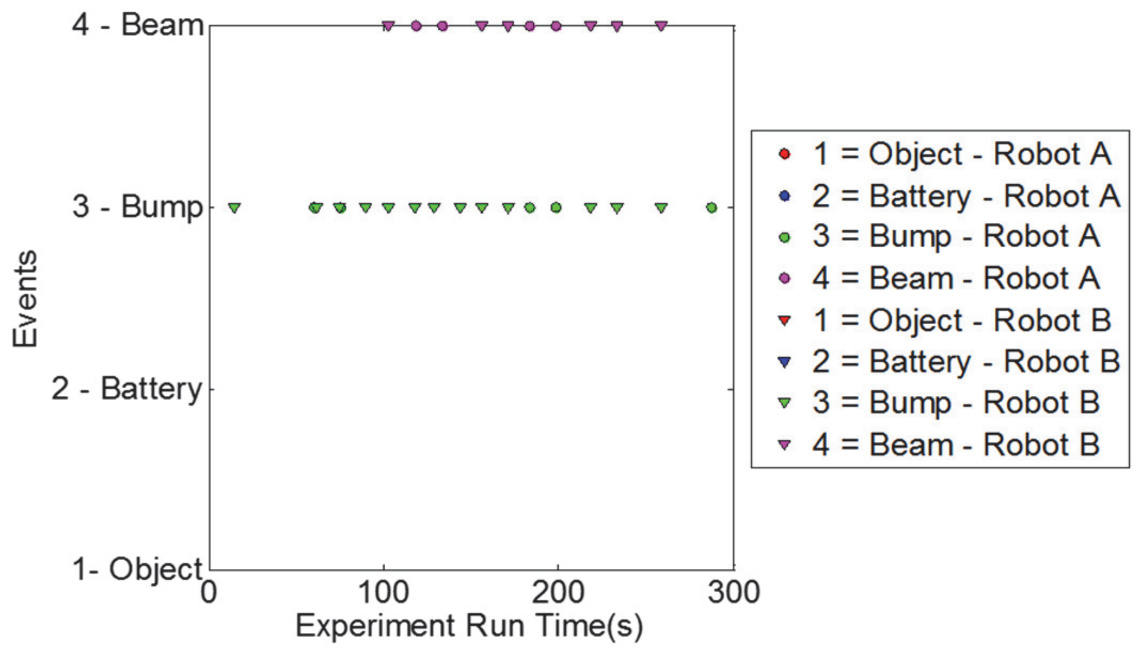

Figure 4.79 Event inputs into the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Aversive)

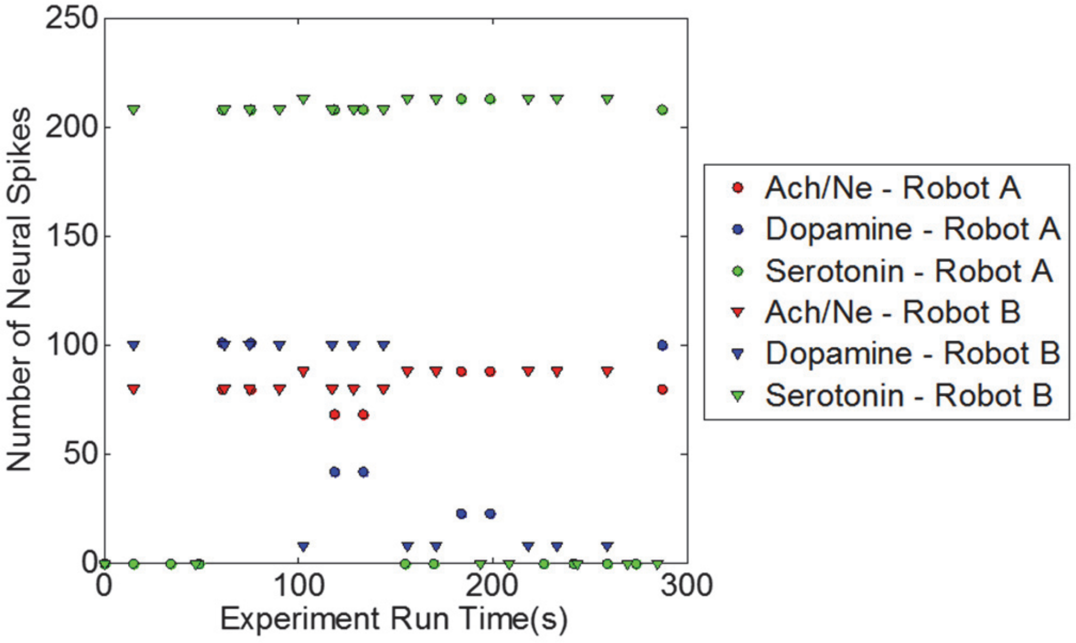

Figure 4.80 Neuromodulator activity of the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Aversive)

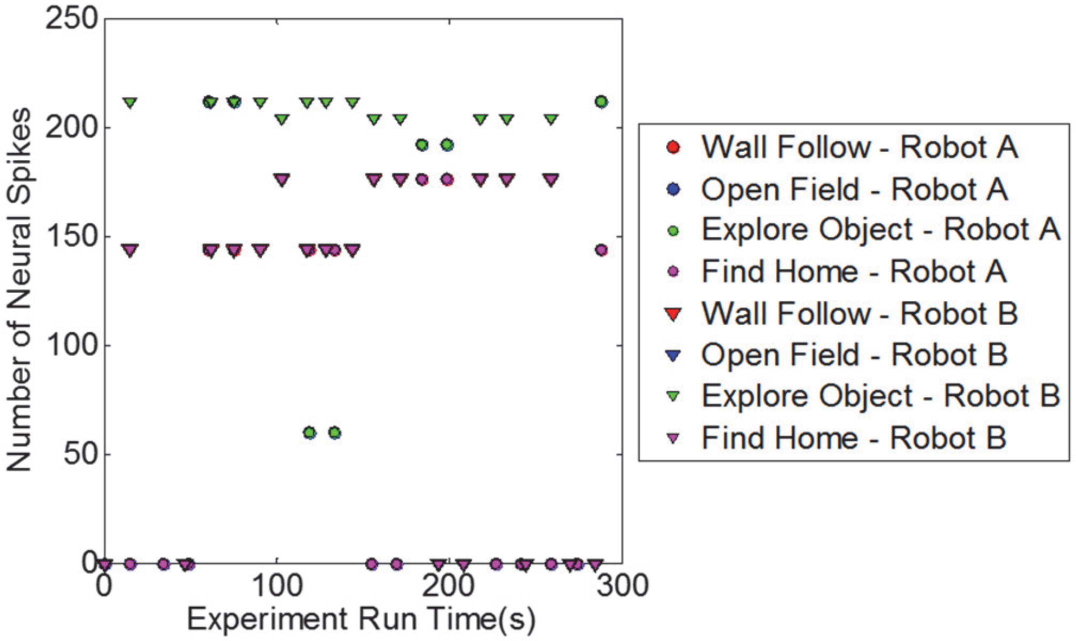

Figure 4.81 State transitions of the ROS-SNN simulation program (Two Robots, Risk Aversive-Risk Aversive)

4.6.3 Risk Aversive Robot with Risk Distracted Robot

Results are presented in Figure 4.82 through Figure 4.84 for two robots with Robot A in risk aversive mode and Robot B in distracted mode. As seen in Figure 4.83, the most active neuromodulator for Robot A (risk aversive) was serotonin and the most prominent action for Robot A is serotonergic in nature (Find Home) (Figure 4.84). For Robot B and its degraded suppression ability of ACh/NE neurons, the most active neuromodulator was dopamine (since the distracted phase was still linked to dopamine through the bump feature) (Figure 4.83). The most active neuron counts of Robot B in the distracted mode seemed to overlap, further indicating its lack of focus.

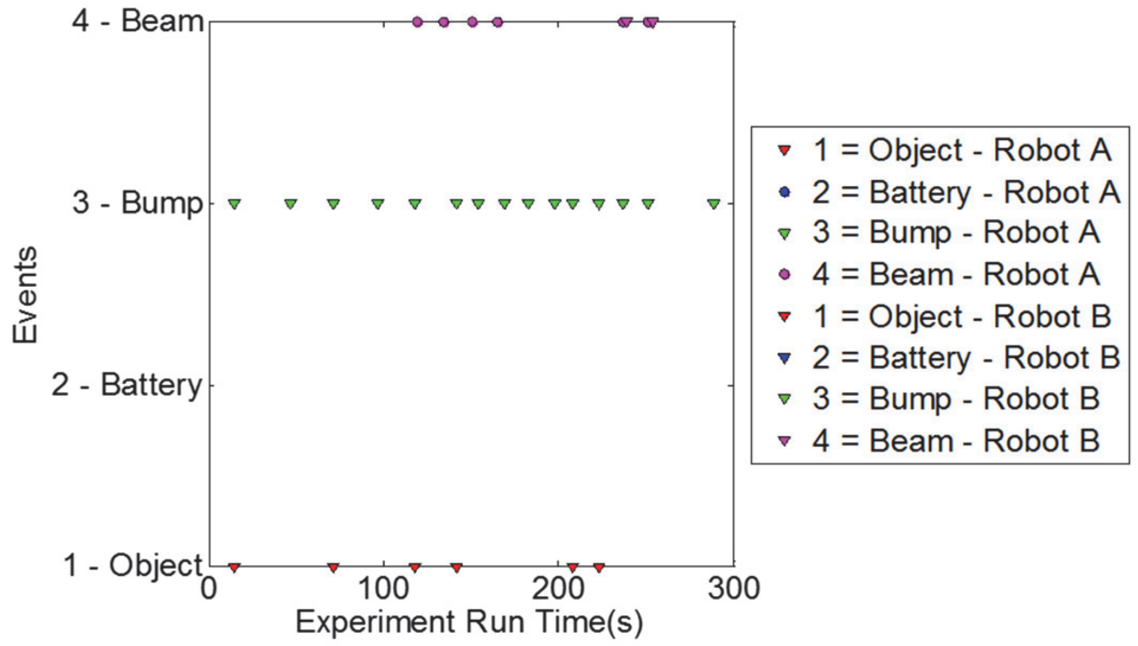

Figure 4.82 Event inputs into the ROS-SNN simulation program (Two Robots, Risk Aversive-Distracted)

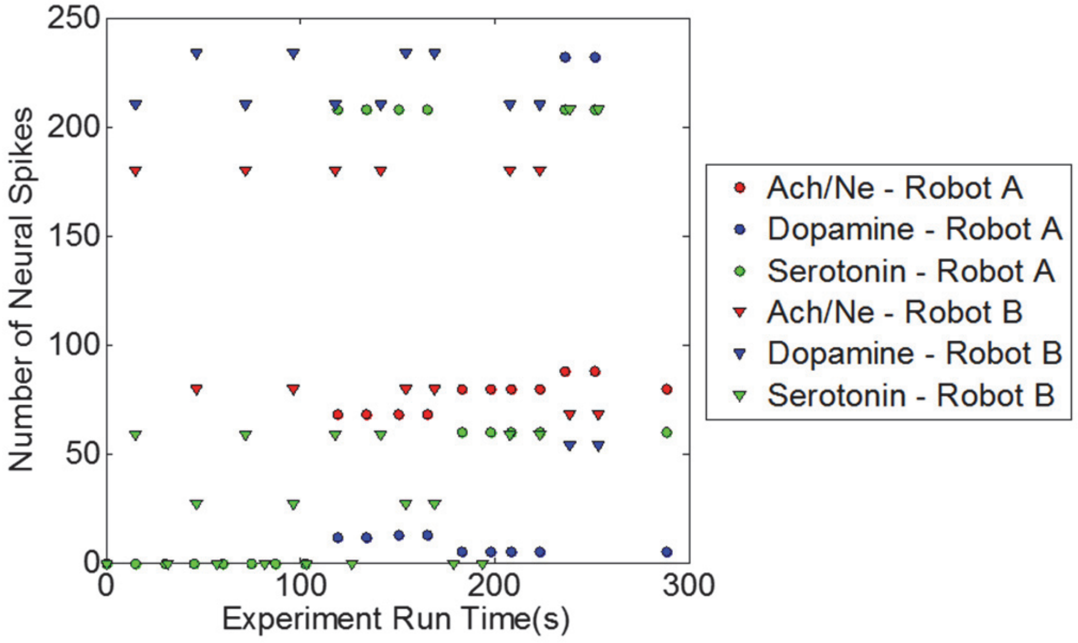

Figure 4.83 Neuromodulator activity of the ROS-SNN simulation program (Two Robots, Risk Aversive-Distracted)

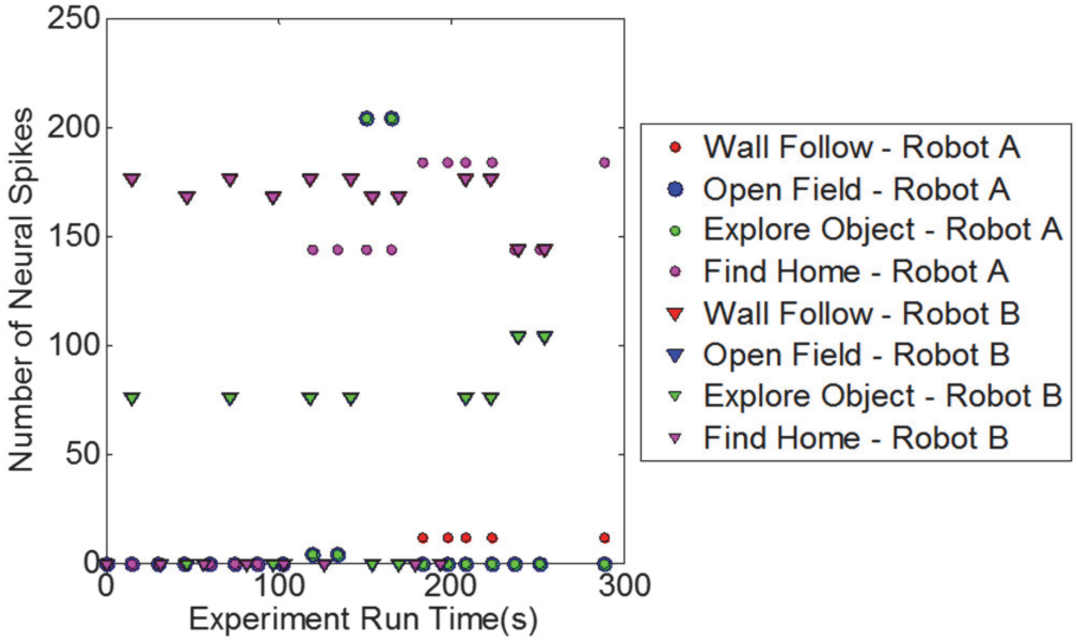

Figure 4.84 State transitions of the ROS-SNN simulation program (Two Robots, Risk Aversive-Distracted)

4.6.4 Risk Taking Robot with Risk Taking Robot

Results are presented in Figure 4.85 through Figure 4.87 for both robots in risk taking mode. As both robots were in risk-taking mode, the most active neuromodulator was dopamine (Figure 4.86). Even though the close proximity to the home base beam (Figure 4.85) constantly induced serotonergic spikes in the network (due to Find Home being a serotonergic action) the highest spiking actions of the robots were still dopaminergic in nature (Figure 4.87).

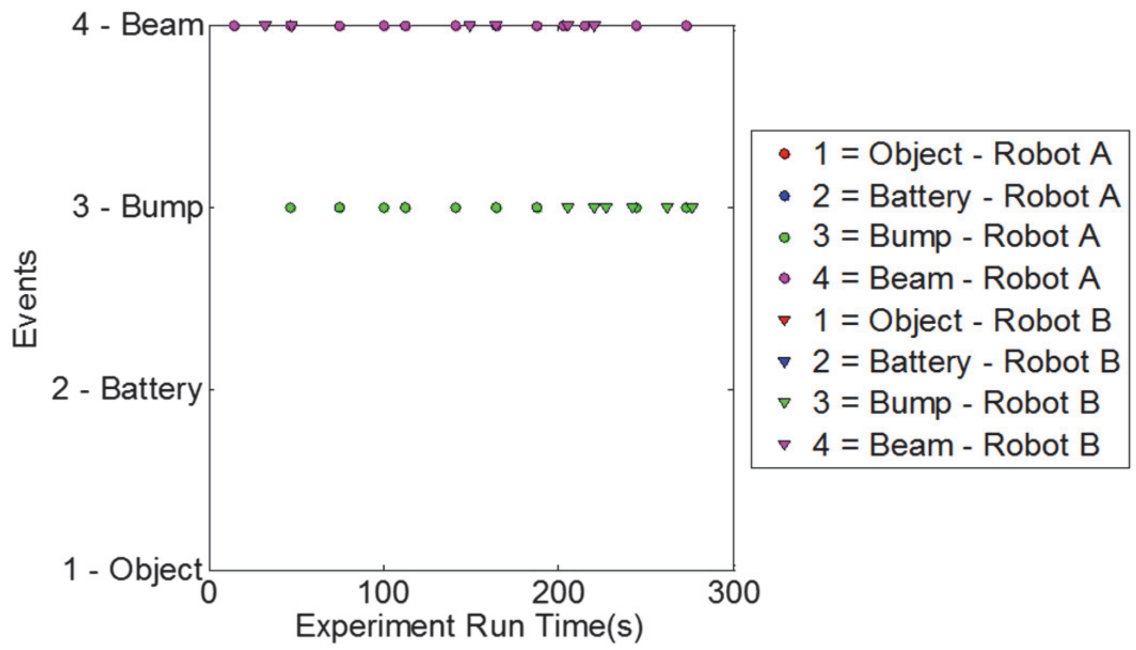

Figure 4.85 Event inputs into the ROS-SNN simulation program (Two Robots, Risk Taking-Risk-Taking)

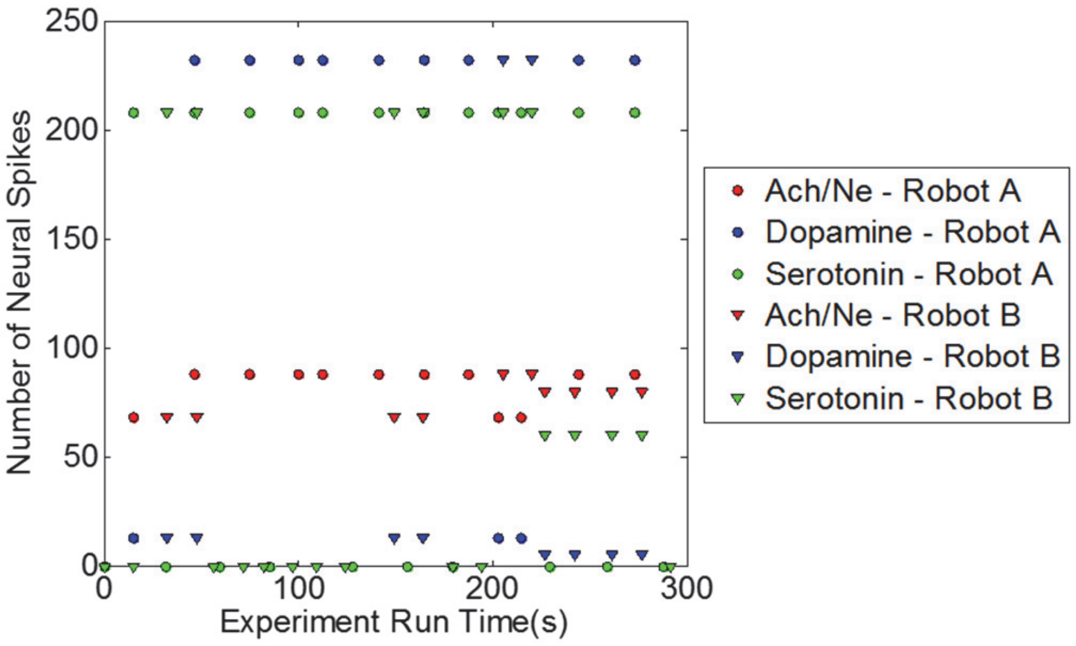

Figure 4.86 Neuromodulator activity of the ROS-SNN simulation program (Two Robots, Risk Taking-Risk Taking)

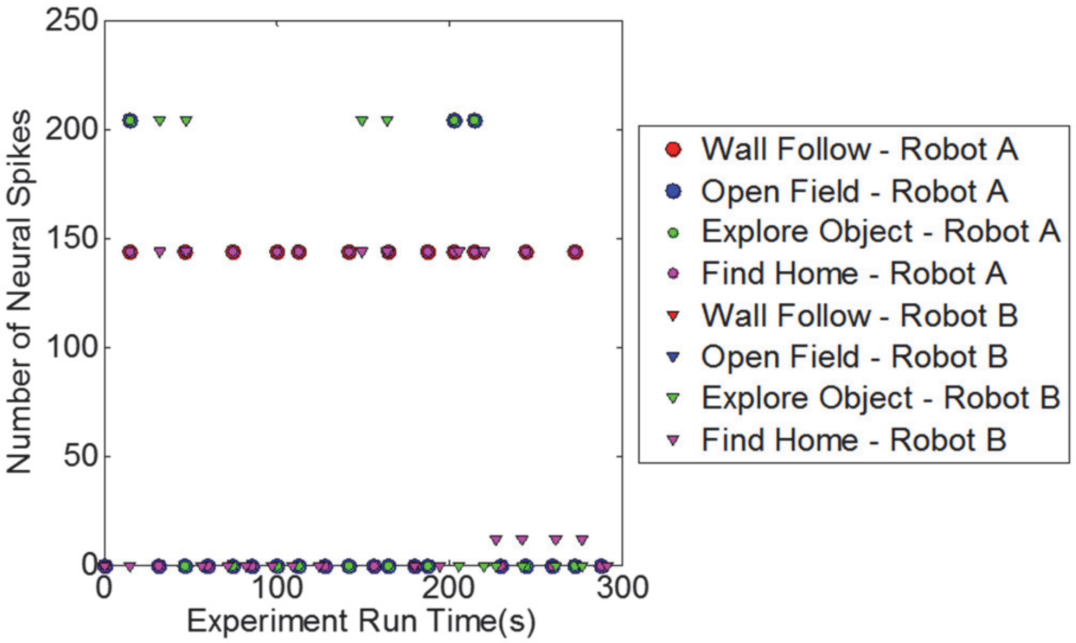

Figure 4.87 State transitions of the ROS-SNN simulation program (Two Robots, Risk Taking-Risk Taking)

4.6.5 Risk Taking Robot with Distracted Robot

Results are presented in Figure 4.88 through Figure 4.90 for two robots with Robot A in risk taking mode and Robot B in distracted mode. Being in an open area the risk taking robot (Robot A) did not register an event until half-way in the run (Figure 4.88). Once it did it turned out to be serotonergic in nature (Beam). But once the Bump event was active, the serotonin levels in the network for Robot A dropped and the dopamine levels increased (Figure 4.89). As seen before, even though there were serotonergic actions induced by the Beam event, the highest spiking actions were those of dopaminergic actions. For the distracted robot, there was the usually large number of dopaminergic spikes with a constant level of ACh/NE under it.

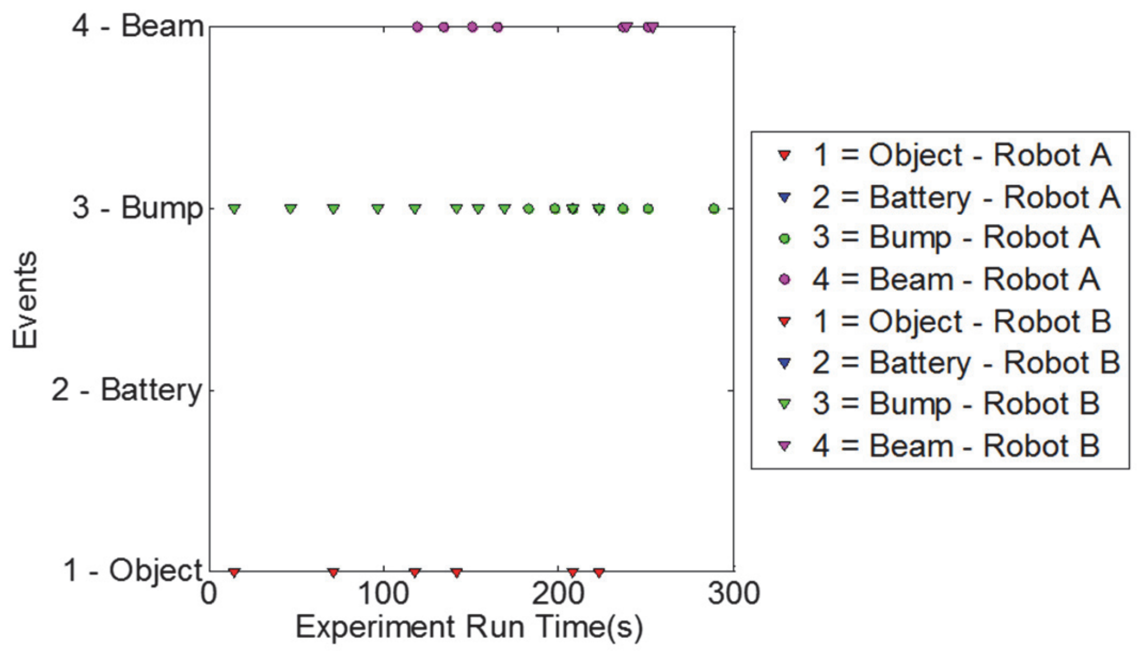

Figure 4.88 Event inputs into the ROS-SNN simulation program (Two Robots, Risk Taking-Risk-Taking)

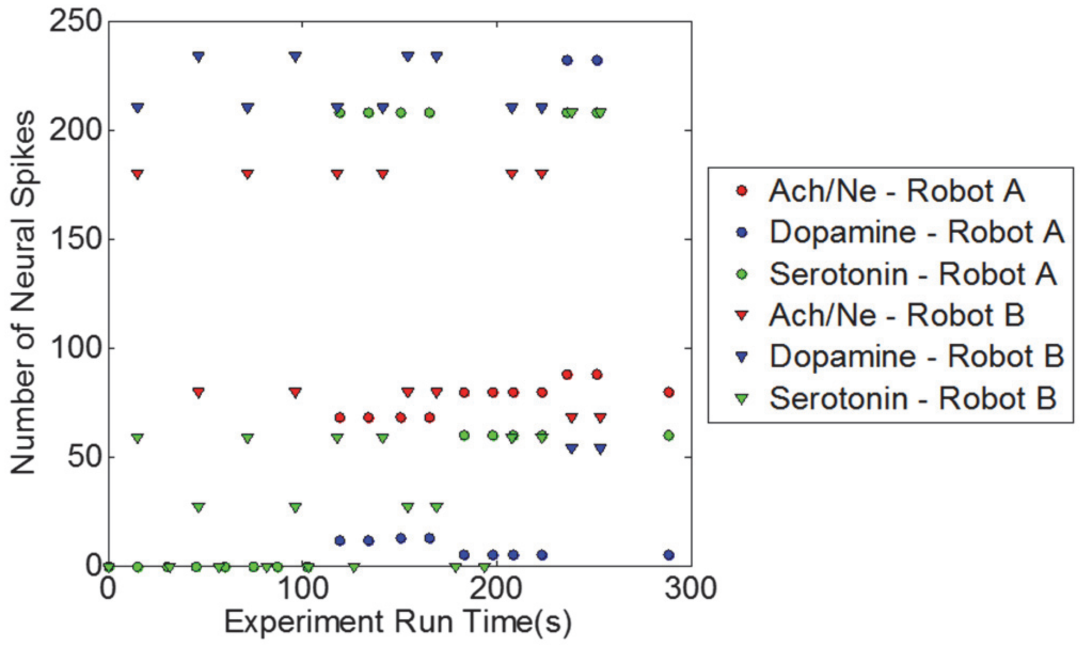

Figure 4.89 Neuromodulator activity of the ROS-SNN simulation program (Two Robots, Risk Taking-Risk Taking)

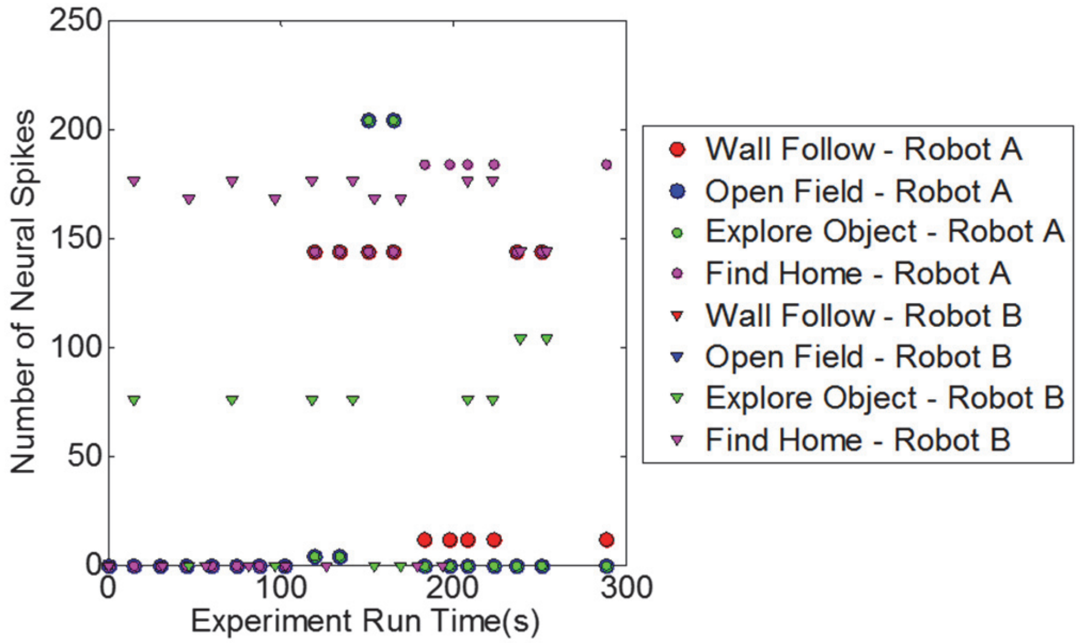

Figure 4.90 State transitions of the ROS-SNN simulation program (Two Robots, Risk Taking-Risk Taking)

4.7 Spiking Neuron Model (Three Robots)

The use of the GPU accelerated model was then applied to the same three-robot scenarios presented in Section 4.4. This continues the evolution of the previous simple neuron models.

4.7.1 Two Risk Aversive Robots with One Risk Taking Robot

Results are presented in Figure 4.91 through Figure 4.93 for three robots with two robots (Robot A, Robot B) in risk aversive mode and the third (Robot C) in risk taking mode. The most active neuromodulator in this run was serotonin for Robot A and Robot B as expected. What was not expected was that the risk taking robot (Robot C) would have mostly serotonin spiking throughout the period of run. This showed that while the spiking neuron model was based on a more biologically realistic model the tuning of the Izhikevich model of Regular Spiking must be done for proper functioning of SNN. This could also be shaped by the constant induction of the beam event on Robot C in this particular run. While the ACh/NE neurons were used to eventually suppress constant event inputs this did not change the fact that the home beam input was serotonergic in nature.

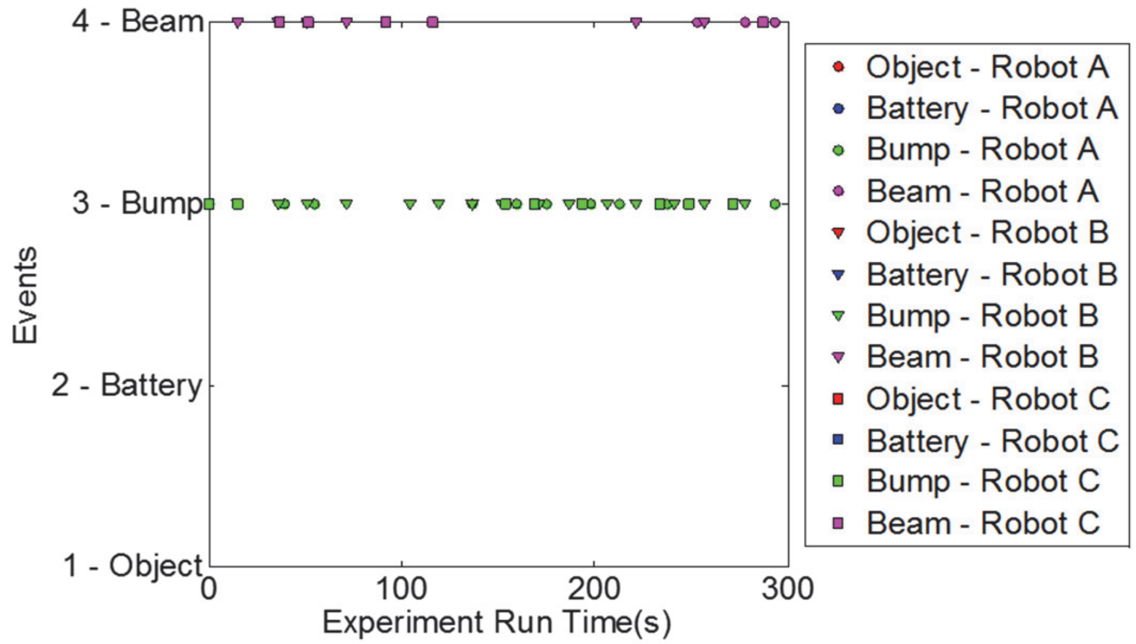

Figure 4.91 Event inputs into the ROS-SNN simulation program (Three Robots, Two Risk Aversive/One Risk Taking)

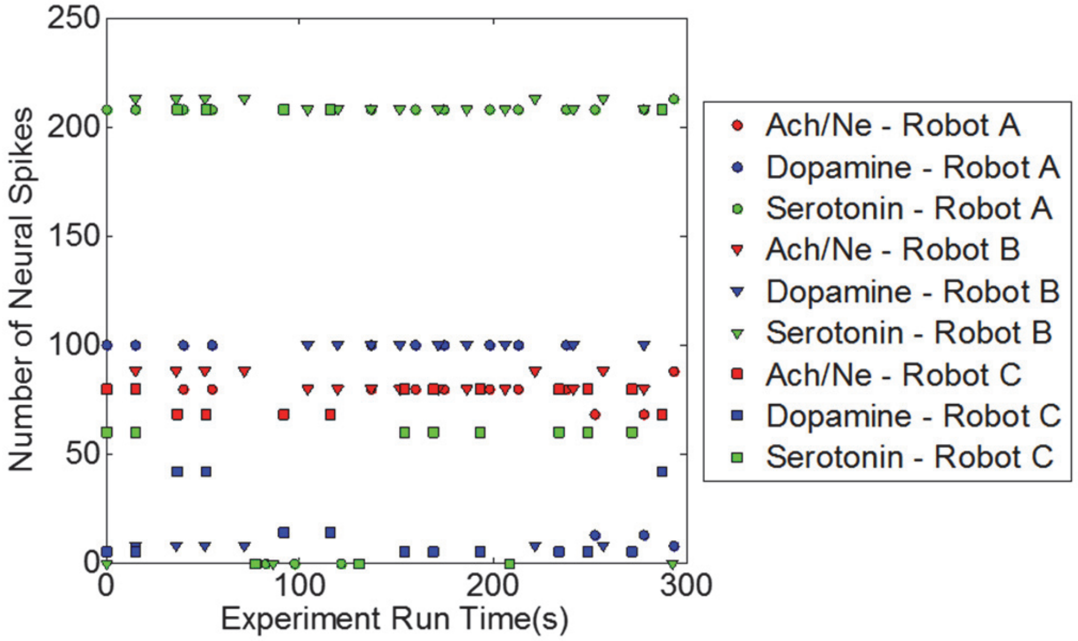

Figure 4.92 Neuromodulator activity of the ROS-SNN simulation program (Three Robots, Two Risk Aversive/One Risk Taking)

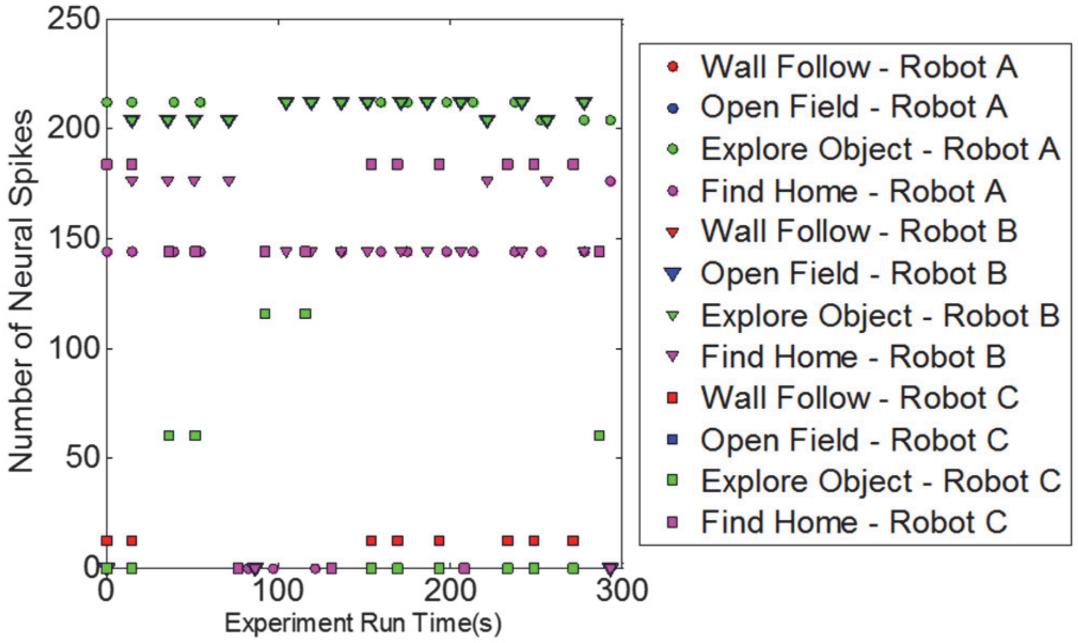

Figure 4.93 State transitions of the ROS-SNN simulation program (Three Robots, Two Risk Aversive/One Risk Taking)

4.7.2 Two Risk Taking Robots with One Risk Aversive Robot

Results are presented in Figure 4.94 through Figure 4.96 for three robots with two robots (Robot A, Robot B) in risk taking mode and the third (Robot C) in risk aversive mode. In this case, the number spikes in dopamine group reached the highest of any of the neuromodulator for Robot A and robot B with a consistently high number of serotonin spikes for Robot C (Figure 4.95). With Robot A having high number of Explore Objects and Open Field neuron spikes, the exploratory nature of the robotic platform in dopaminergic actions was demonstrated. Robot B, the second risk taking robot, showed its internal network battle between the Find Home spiking neuron group induced by the home base and the Explore Object neuron group induced by its core dopaminergic nature. With occasional spikes of Explore Object neural activity, the main action of Robot C was to Find Home (Figure 4.96).

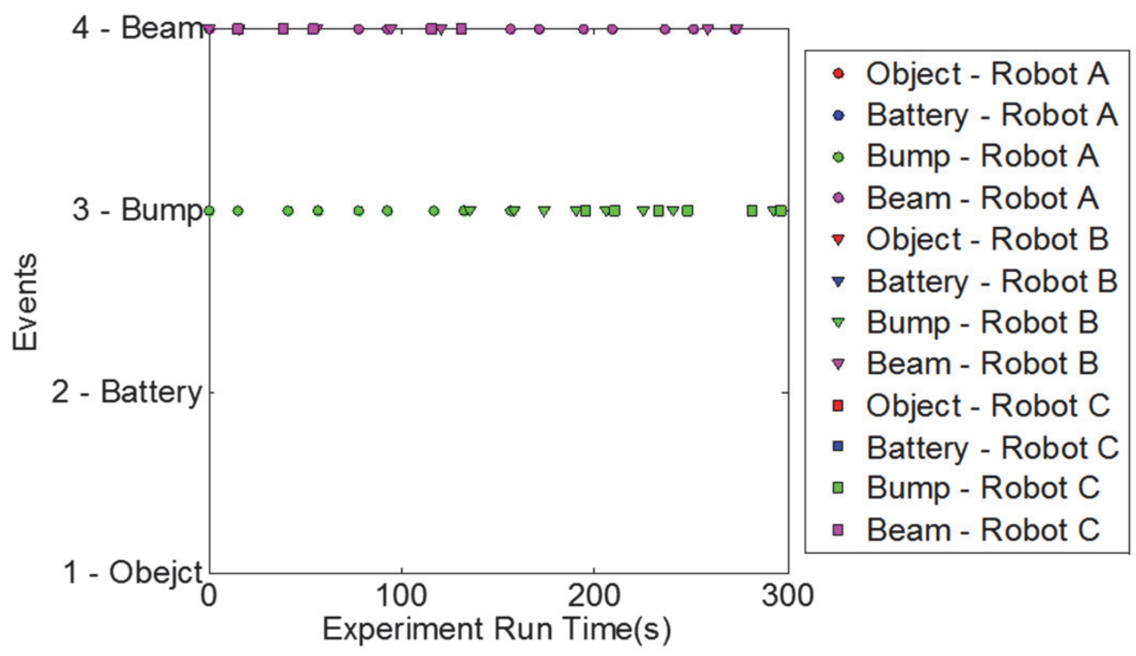

Figure 4.94 Event inputs into the ROS-SNN simulation program (Three Robots, Two Risk Taking/One Risk Aversive)

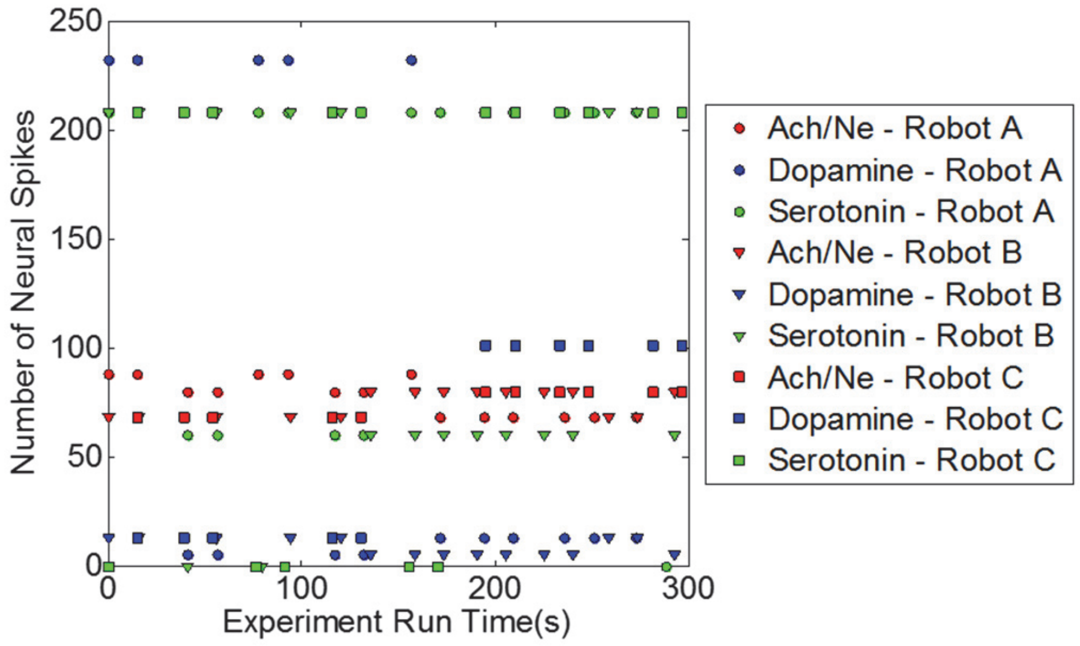

Figure 4.95 Neuromodulator activity of the ROS-SNN simulation program (Three Robots, Two Risk Taking/One Risk Aversive)

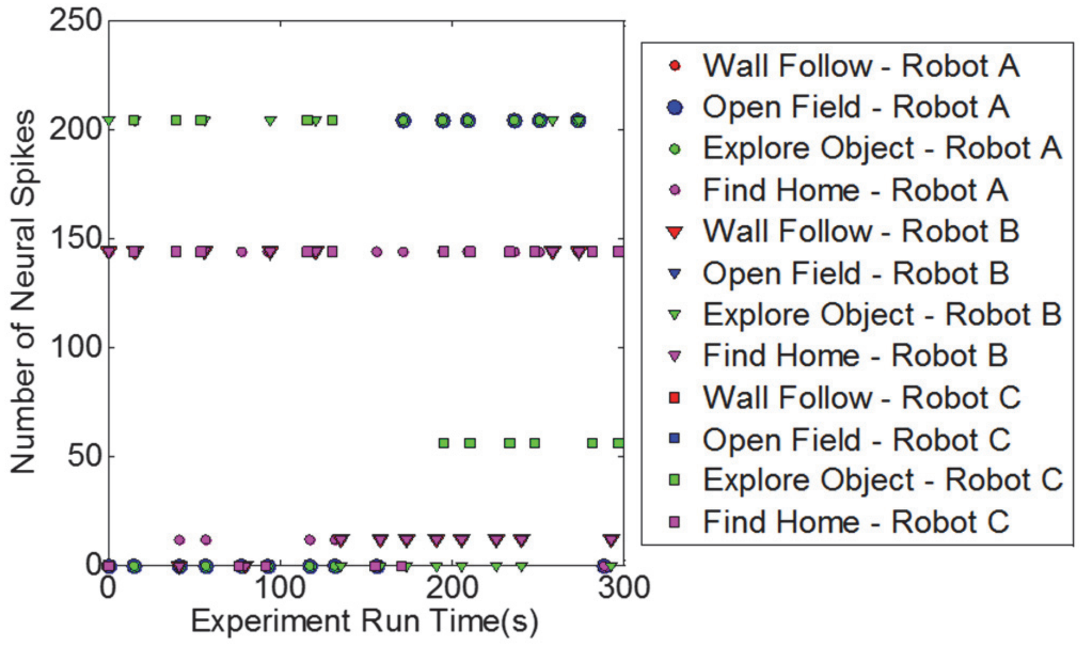

Figure 4.96 State transitions of the ROS-SNN simulation program (Three Robots, Two Risk Taking/One Risk Aversive)

4.8 Spiking Neuron Model (Four Robots)

4.8.1 Two Risk Aversive Robots with Two Risk Taking Robots

Results are presented in Figure 4.97 through Figure 4.99 for four robots with two robots (Robot A, Robot B) in risk aversive mode and the other two (Robot C and Robot D) in risk taking mode. In this case the first two robots (A, B) had highest numbers of serotonergic spikes and the other two robots (C, D) had highest numbers of dopaminergic (Figure 4.98). Robot C's exploratory nature can be seen as active in Figure 4.99. The high level of home beam input into the network induced more serotonergic action spiking in the risk taking Robot D (Figure 4.99).

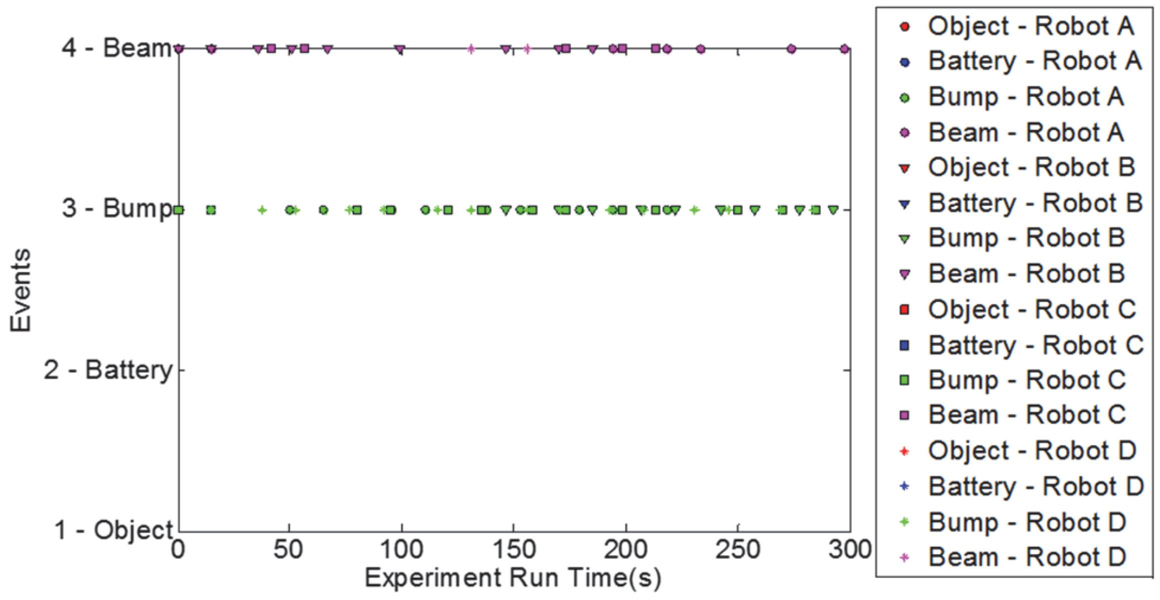

Figure 4.97 Event inputs into the ROS-SNN simulation program (Four Robots, Two Risk Aversive/Two Risk Taking)

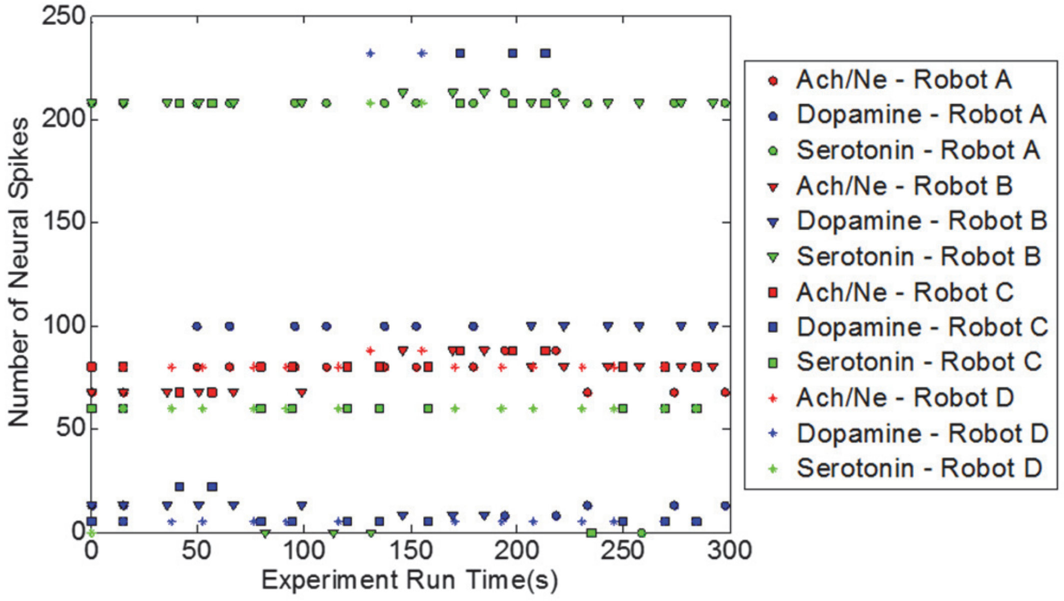

Figure 4.98 Neuromodulator activity of the ROS-SNN simulation program (Four Robots, Two Risk Aversive/Two Risk Taking)

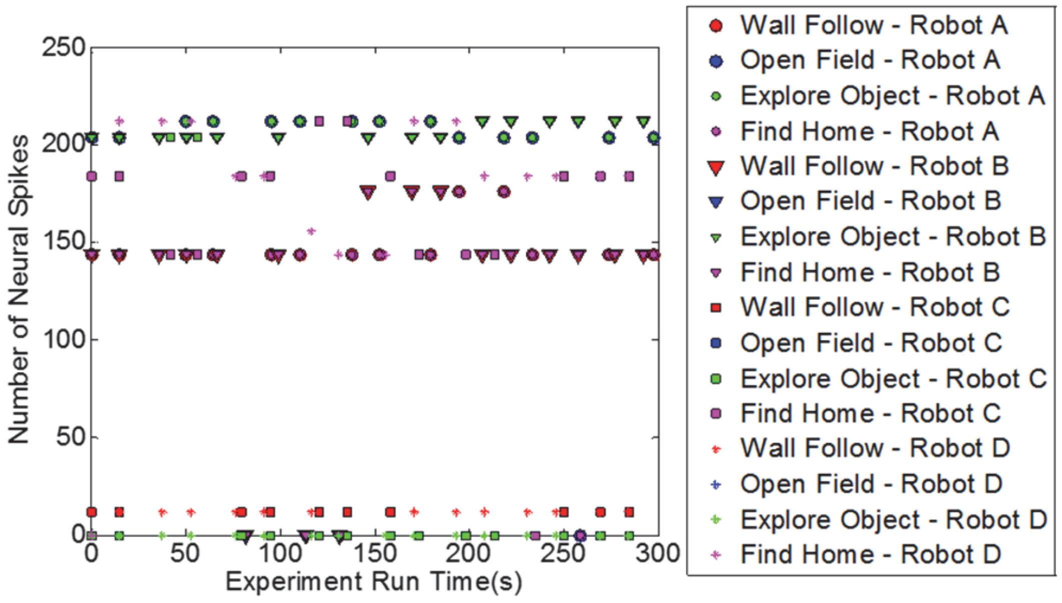

Figure 4.99 State transitions of the ROS-SNN simulation program (Four Robots, Two Risk Aversive/Two Risk Taking)

4.8.2. Two Risk Taking Robots with One Risk Aversive Robot and One Distracted Robot

Results are presented in Figure 4.100 through Figure 4.102 for four robots with two robots (Robot A, Robot B) in risk taking mode, one robot (Robot C) in risk aversive mode and the fourth robot (Robot D) in distracted mode. As expected there were high levels of dopamine spikes throughout the run with Robot A and Robot B being exploratory in nature (Figure 4.101). For Robot C constant Find Home spiking activity was consistent with its risk aversive mode set within the robotic neural network. The various shifting levels of neural activity for Robot D in both neuromodulators and state neuron activity demonstrated its distracted mode.

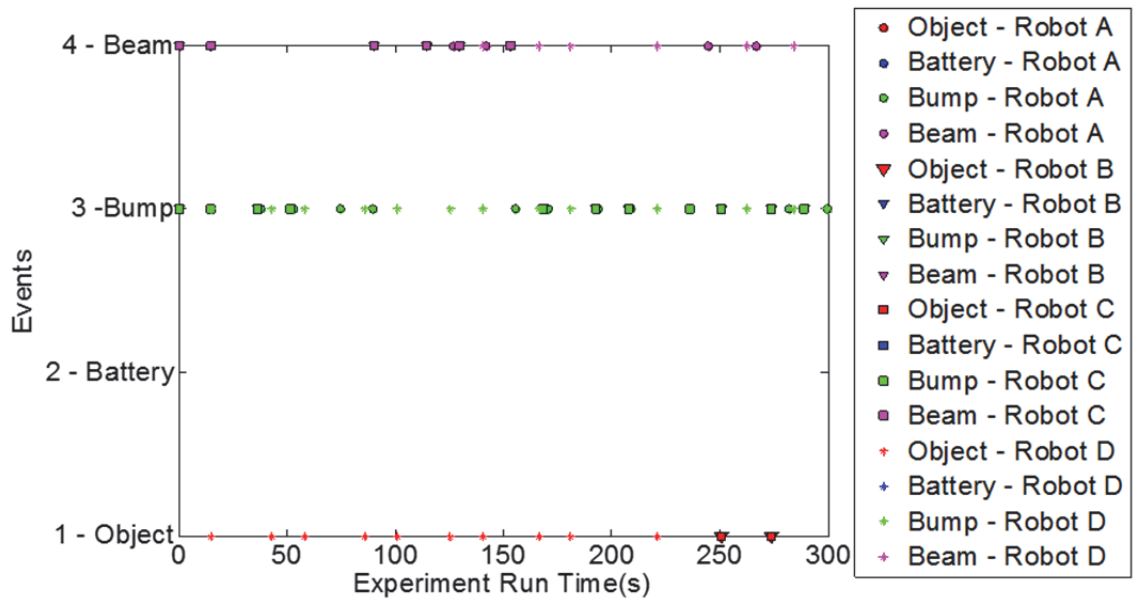

Figure 4.100 Event inputs into the ROS-SNN simulation program (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

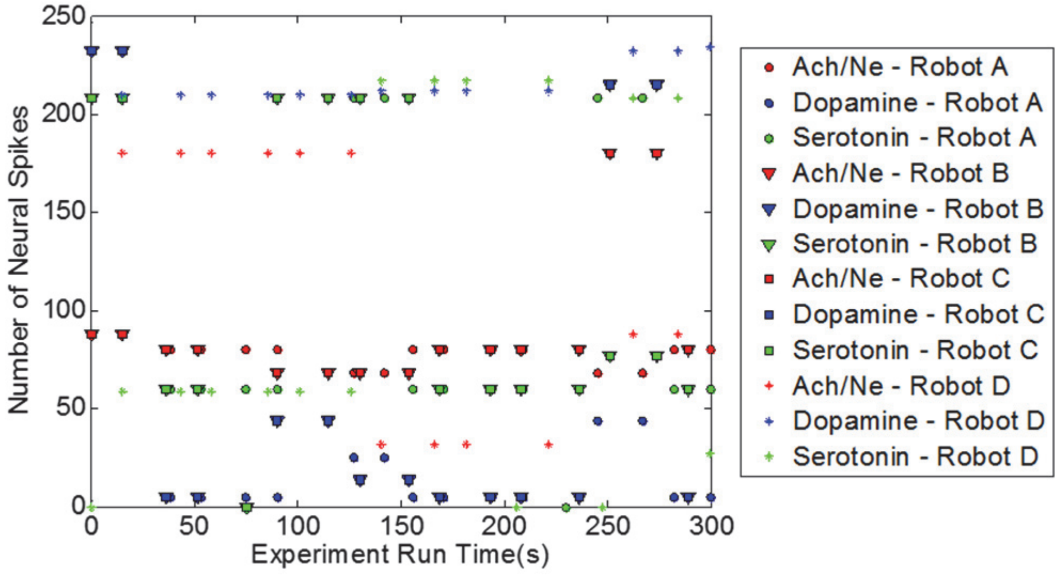

Figure 4.101 Neuromodulator activity of the ROS-SNN simulation program (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

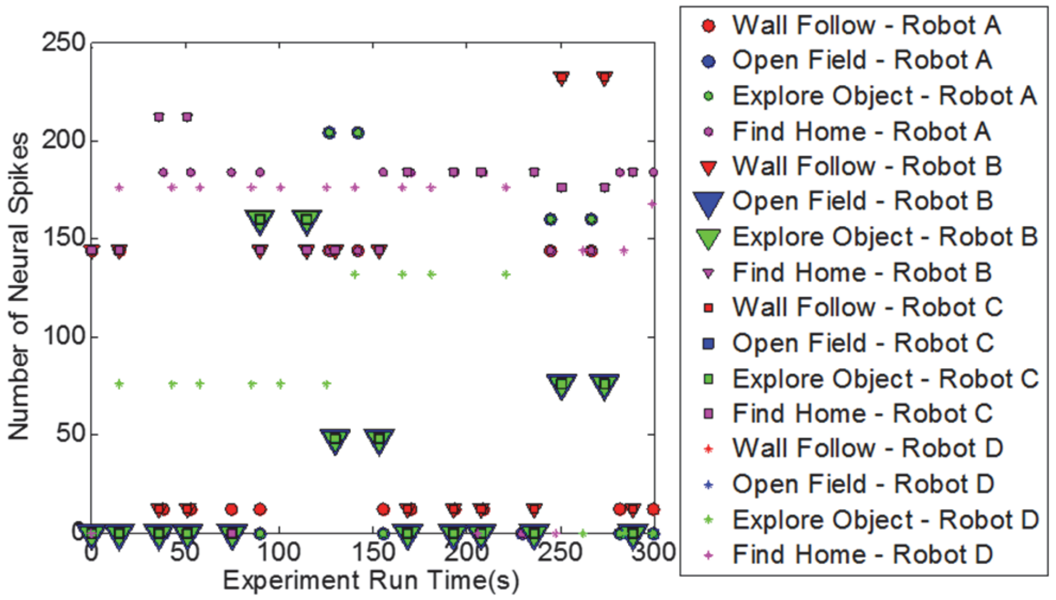

Figure 4.102 State transitions of the ROS-SNN simulation program (Four Robots, Two Risk Taking/One Risk Aversive/One Distracted)

4.9 Discussion of Results

As the experiments continued, comparisons between the various cases using the simple neuron model and the spiking neuron model began to show interesting observations. For instance, even though the SNN solution introduced cloud computing network platforms that were dependent on traffic size of data and the size of the SNN models, the simulation of neuromodulated activity was around the same speed as the on-board solutions. The truly demonstrated the computing power of GPUs for the SNN models. The overall behavior patterns of the robots between the simple neuron model and the SNN model were qualitatively similar. However, SNN based model, due to its modeling details, had the capability of giving better insights into the behavioral patterns of the robots. It should also be noted that the apparent contradiction of some behavioral patterns of robots in SNN based model could be resolved with considerations on tuning the SNN model parameters. The ROS-cloud computing based implementation of the models was very effective and could be improved further.

CHAPTER 5 . CONCLUSIONS AND RECOMMENDATIONS

5.1 Summary of the Present Work

In order to demonstrate the effectiveness of a spiking neural network's ability to simulate a vertebrate neuromodulation control system, robotic code was created starting with a simple neural network. This simple neural network was a three layer network, consisting of the following: an input layer to receive the effects of stimuli in the robot's environment; a decision making layer consisting of neurotransmitters- dopamine, serotonin, noradrenaline, and acetylcholine; and an output layer representing the defined "states" the robot could go into. Noradrenaline and acetylcholine could help filter out noise and increase focus on a novel event. Dopamine acted as a reward motivator and increased the level of risk-taking by a vertebrate. Serotonin decreased the level of risk-taking by a vertebrate.

Once this simple neuromodulation code was achieved, a robotic software platform was needed to implement actual applications. This came in the form of the Robot Operating System (ROS) platform (Nickels and Kerr 2012). The ROS platform is an open source software platform widely used for its reusability and flexibility on multiple hardware platforms. Once the ROS platform was established and the simple neural network was shown to be responsive on ROS, a publicly available SNN simulator that could harness the abilities of GPU parallel processing was integrated into the control scheme. Instead of simply facilitating connections and data, ROS would be responsible for starting and stopping the simulation of thousands of neurons.

Each of the SNN's neurons was based on the Izhikevich neuron model, a model known for being biologically descriptive and relatively efficient to compute in large numbers. For the study, the RS (regular spiking) variation of the Izhikevich neuron model was used. RS Izhikevich neurons represent typical "default" neurons. The simple neural network was adapted to the SNN simulation platform by turning each neuron in the previous experiment into groups of many neurons. Using GPU equipped PCs as cloud servers for parallel processing, models of 2000+ neurons were implemented that lasted a second whenever the robot sent sensory inputs over the ROS network.

The current study showed that spiking neural networks could be a plausible control system implemented on a highly adaptable framework like ROS in cloud computing environment. Due to the increased complexity in SNN model, it might be necessary to tune the model parameters for wider applications. Compared to the simple neural network, the tradeoff between complexity, details of the SNN model and speed of computation would need further consideration.

5.2 Scope of Future Work

While the ability of the Izhikevich neuron model was more biologically complex and realistic there was a tradeoff for stability. When using neuron counts over 5,000 in the simulation, output states and neuromodulator levels stopped correlating and the results became more random. In order to harness the power of large networks, it would be necessary to tune the model parameters, similar to the evolutionary algorithm based technique proposed recently (Carlson, et al. 2014).

While the ROS framework used in this project was only used to support a maximum of four robots, the architecture can scale to a larger number of robotic platforms as long as the networking hardware can handle the data traffic. Expanding the number of robots used in this type of project would produce novel results.

The ROS framework also supports the networking of heterogeneous robotic platforms. Using a neural network on a “ground” robot like the Turtlebot simultaneously with an “air” robot like a UAV would add another dimension to this work.

In the present work, the models were implemented in ROS-cloud computing environment without any network optimization. The issues of latency, safety and security of the ROS- cloud computing environment would have to be considered in future.

REFERENCES

Alexander, William H. , and Olaf Sporns. "Neuromodulation and plasticity in an autonomous robot." *Neural Networks*, 2002: 761-774.

Carlson, Kristofor David, Jayram M Nageswaran, Nikil Dutt, and Jeffrey L Krichmar. "An efficient automated parameter tuning framework for spiking neural networks." *Frontiers in Neuroscience* 8, no. 10 (2014).

Cox, Brian R., and Jeffrey L. Krichmar. "Neuromodulation as a Robot Controller: A Brain Inspired." *IEEE Robotics & Automation Magazine*, September 2009: 72-80.

Gerstner, Wulfram. "Spiking Neurons." In *Pulsed Neural Networks*, by Wolfgang Maass, & Christopher Bishop, 3-54. The MIT Press, 1999.

Izhikevich, Eugene M. "Simple Model of Spiking Neurons." *IEEE Transactions on Neural Networks*, November 2003: 1569-1572.

Kistler, Werner M., and Wulfram Gerstner. *Spiking Neuron Models.* Cambridge: Cambridge University Press, 2002.

Krichmar, Jeffery L. "The Neuromodulatory System: A Framework for Survival and Adaptive Behavior in a Challenging World." *Adaptive Behavior* 16, no. 6 (2008): 385-399.

Krichmar, Jeffrey L. "A neurorobotic platform to test the influence of neuromodulatory signaling on anxious and curious behavior." *Frontiers in Neurorobotics* 7, no. 1 (2013).

Krichmar, Jeffrey L. "A biologically inspired action selection algorithm based on principles of neuromodulation." *Neural Networks (IJCNN), The 2012 International Joint Conference on.* 2012. 1 - 8.

McCulloch, Warren, and Walter Pitts . "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics*, 1943: 115-133.

Nageswaran , Jayram Moorkanikara, Nikil Dutt, Jeffrey L Krichmar, Alex Nicolau, and Alex Veidenbaum. "Efficient simulation of large-scale spiking neural networks using CUDA graphics processors." *Proceedings of the 2009 International Joint Conference on Neural Networks.* Atlanta: IEEE Press, 2009. 3201-3208.

Nickels, K., and J. Kerr. "Robot operating systems: Bridging the gap between human and robot." *2012 44th Southeastern Symposium on System Theory (SSST).* Jacksonville: IEEE, 2012. 99 - 104.

Prince, Akimul. "Neuromodulation based control of an autonomous robot." *Master's Thesis, Georgia Southern University*, 2013.

Prince, Akimul, and Biswanath Samanta. "Control of Autonomous Robots Using the Principles of Neuromodulation." *ASME 2013 Dynamic Systems and Control Conference.* 2013. 8.

Ramòn y Cajal, Santiago. *Histologie du système nerveux de l'homme & des vertébrés* . Paris: Maloine, 1909.

Richert, Micah, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L. Krichmar. *An efficient simulation environment for modeling large-scale cortical processing.* September 2011. http://www.frontiersin.org/Neuroinformatics/10.3389/fninf.2011.00019/full (accessed August 2012).

ROS.org | Powering the world's robots. 2013. http://www.ros.org/.

Rosenblatt, Frank. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, 1958: 386-408.

Turner, Jonathan G. "Intelligent neural network control system design and FPGA based implementation." *Master's Thesis, Georgia Southern University*, 2012.

TurtleBot. 2013. http://turtlebot.com/.

Vreeken, Jilles. "Spiking neural networks, an introduction." *Technical Report UU-CS 2003-008* (Utrecht University: Information and Computing Sciences), 2003: 1-5.

APPENDICIES

Appendix A

SOFTWARE IMPLEMENTATION OF NEUROMODULATED NETWORK IN ROS-COMPATIBLE C++

```
//Ros Header
#include "ros/ros.h"

//Msg headers
#include <std_msgs/String.h>
#include <std_msgs/Empty.h>

#include <tf/tf.h>

#include <geometry_msgs/Twist.h> //Movement Msgs

#include <nav_msgs/Odometry.h> //Odom Msgs


#include <sensor_msgs/Image.h> //Depth Image Msgs
#include <sensor_msgs/CameraInfo.h> //CameraInfo Msgs
#include <sensor_msgs/LaserScan.h> //LaserScan Msgs

#include <kobuki_msgs/DockInfraRed.h> //Kobuki IR Sensor Status Msgs
#include <kobuki_msgs/SensorState.h> //Kobuki interal Sensor Msgs
#include "complete_test/irobotdock.h"


#include <fstream>
#include <iterator>
#include <string>
#include <vector>

#include <sstream>
using namespace std;

// Global Variables

// -------------
// - control/flow variables
const double PI = 3.1415926; //PI constant for rotation
int enable_sub = 0; //rotation enable
int controlint = 0; //rotation control
float originyaw; //radian control
int stage = 0; //rotation stage control
float fromleft = 0; //needed in DestinRad Subfunction
float fromright = 0; //needed in DestinRad Subfunction
float camera; //global camera variable
int charger = 0; //charger status
int bumper = 0; //bumper status
std::vector<int>IR_Sensor (3,0); //IR Sensor vector
vector <double> locate(5,0); //Camera Vectors
int locate_enable = 0; //Vectoring Enable

int chc = 0; //neuron choice


// --------------
// - recording data
vector <vector <double> > loginx; //total data
vector<double> logvec(28,0); //individual run vectors

// ----------------
// - state neurons
int STATE_WALL_FOLLOW = 1;
int STATE_OPEN_FIELD = 2;
int STATE_EXPLORE_OBJECT = 3;
int STATE_FIND_HOME = 4;
int STATE_AT_HOME = 5;
int STATE_LEAVE_HOME = 6;

// ----------------
// - events
int e_ping_value = 1;
int e_battery = 2;
int e_bump = 3;
int e_beam = 4;
int e = 4;
//event = zeros (1,e);
std::vector<double>event (e,0);

// ----------------
// - parameters for return home
int FIND_HOME_TIMEOUT = 1;
int dock_time = 0;
int found_home = 0;

// ----------------
// - parameters for wall following
int WALL_LEFT = -1;
int WALL_RIGHT = 1;
int wall = 0;
int wallfollow_time = 0;

// ----------------
// - neurons
int N = 4;
//global n;
```

```
//n = zeros (1,N);
std::vector<double> n (N,0);

// ----------------
// - neuromodulators
int NM_DA =  1;
int NM_5HT = 2;
int NM = 2;
//global nm;
//nm = zeros (1,NM);
std::vector<double> nm (NM,0);

// ----------------
// - ACH/NE neuromodulation
//global achne;
//achne = zeros (1,e);
//%achne =ones(1,e);
std::vector<double> achne (e,0);

// ----------------
// - parameters for ping sensor event
//int comparing_distance = 20; //% 20 centimeter
float comparing_distance = 0.52;
//%comparing_max_distance = 70;

// ----------------
// - parameters for battery event
float battery;
float battery_level;
float battery_initial_level;
int battery_lock = 0;

// ----------------
// - parameters for bump event
float too_close = 0.72; //% meausred in m

// ----------------
// - parameters for beam event
int force_field = 242;
int buoy_and_force_field = 250;

// ----------------
// - set weights to their initial values
vector <vector <double> > w_n_n_exc;
vector <vector <double> > w_n_n_inh;
vector <vector <double> > w_nm_n;
vector <vector <double> > w_e_n;
vector <vector <double> > w_e_nm;
vector <double> w_e_achne;

ros::Publisher chatter_pub; //movement publisher
ros::Publisher chatter_pub2; //odom reset publisher
```

```
void roomba_net_init()
{
      // state neuron intrinsic connectivity
      for (int r = 1; r <= N; r++)
      {
            vector <double> row(N, 0.5);
            w_n_n_exc.push_back(row);
      }

      for (int r = 1; r <= N; r++)
      {
            vector <double> row(N, -1.0);
            w_n_n_inh.push_back(row);
      }

      for (int r = 1; r <N; r++)
      {
            ROS_INFO("Pass 3.1");
            w_n_n_exc[r-1][r-1] = 0.0;
      w_n_n_inh[r-1][r-1] = 0.0;
      ROS_INFO("Pass 3.2");
     }



      // neuromodulator to state neuron connectivity
      //w_nm_n = zeros(NM,N);
      for (int r = 1; r <= NM; r++)
      {
            vector <double> row(N, 0);
            w_nm_n.push_back(row);
      }
      w_nm_n[NM_5HT-1][STATE_FIND_HOME-1] = 5;//% 5*rand;5;
      w_nm_n[NM_5HT-1][STATE_WALL_FOLLOW-1] = 5 ;//%5*rand;%5;
      w_nm_n[NM_DA-1][STATE_EXPLORE_OBJECT-1] =5;//%5*rand;%5;
      w_nm_n[NM_DA-1][STATE_OPEN_FIELD-1] =5;   //% 5*rand;%5;

      // event neuron to state neuron connectivity
      //w_e_n = ones(e,N);
      //% w_e_n(e_bump,STATE_FIND_HOME) = 0;
      //%w_e_n = rand(e,N);
      for (int r = 1; r <= e; r++)
      {
            vector <double> row(N, 1);
            w_e_n.push_back(row);
      }

      // event neuron to neuromodulator  connectivity
      //w_e_nm = zeros(e,NM);
      for (int r = 1; r <= e; r++)
      {
            vector <double> row(NM, 0);
            w_e_nm.push_back(row);
      }
      w_e_nm[e_ping_value-1][NM_DA-1] =1; //% 0+(1-0).*rand;
      w_e_nm[e_battery-1][NM_5HT-1] =1;  //%0+(1-0).*rand;%1;
      w_e_nm[e_beam-1][NM_5HT-1] =1;  //%0+(1-0).*rand;% 1;
      //w_e_nm[e_bump-1][NM_5HT-1] = 1;  //%0+(1-0).*rand; %1;    %
risk averse behavior

      w_e_nm[e_bump-1][NM_DA-1] = 1; //0+(1-0).*rand;   % risk taking
behavior

      //event neuron to neuromodulator  connectivity
      //w_e_achne = ones(1,e); //% 0+(1-0).*rand(1,e);
      for (int r = 1; r <= e; r++)
      {
            w_e_achne.push_back(1);
      }

}




double activity(double I, double g)
{
/*% activity - sigmoid activation function
%
% Description
%   Sigmoid activation function for rate neuron
%
% Inputs
%   I - synaptic input
%   g - gain or slope of sigmoid curve
%
% Outputs
%   s - activity of neuron between 0 and 1
*/
      double sigmoid;
      sigmoid = 1 / (1 + exp(-g*I));
      return sigmoid;
}

double stp(double xin, double p, double tau, double maxi, bool spk)
{
/*% stp - Short-Term Plasticity
%
% Description
%   Simple version of short-term plasticity rule
%
% Inputs
%   xin - current weight value
%   p - amount to increase or decrease weight if there is a spike
%   tau - recovery time constant.
%   max - maximum weight value
%   spk - 1 if spike occurred.
%
% Outputs
%   x - new weight value
%
*/
double x;

     if (spk == true)
          x = p*xin;
     else
          x = xin + (1-xin)/tau;

     x = min(maxi,x);
     return x;
}



double rand_num()
{
     int v1 = (rand() % 10001);
    double number = (rand() % 10001) / 10000.0;
     return number;
}

double sum_vector(vector <double> vector)
{
     double sum;
     for(int i = 0; i < vector.size();i++)
     {
       sum = sum + vector[i];
     }
 return sum;
}

double min_vector(vector <double> vector)
{
     double min_value;
     min_value = vector[0];

     for (int i = 0;i < (vector.size()-1);i++)
     {
          if (min_value > vector[i+1]){
               min_value = vector[i+1];
          }

     }
```

```
      return min_value;
}

void max_vector_element(vector <double> vector, double& c, double& I)
{
      c = vector[0]; //in case if all elements equal
      I = 0;
      for (int i = 0;i < (vector.size()-1);i++)
      {
            if (c < vector[i+1]){
                  c = vector[i+1];
                  I = i+1;}
      }
}

double DestinRad(double ori, double dest) //computes destination
radian
{
      double destra;
      if ((ori - dest) < -PI) //readjust distance from lower limit
      {
            destra = (ori - dest) + (2*PI);
            fromright = 1; //apply 2pi correction to right direction
below limit
            fromleft = 0;
      }
      else if ((ori - dest) > PI) //readjust distance from upper limit
      {
            destra = (ori - dest) - (2*PI);
            fromright = 0;
            fromleft = 1; //apply 2pi correction to left direction
above limit

      }
      else //within limit
      {
            destra = (ori - dest);
            fromright = 0;
            fromleft = 0;
      }
      return destra;
}

void reveal_vector(vector <double> vector)
{
      ROS_INFO("--DIRECTION VECTOR--");
      ROS_INFO("--DIRECTION VECTOR SIZE-- [%lu]", vector.size());
      for(int i = 0; i < vector.size();i++)
      {
            ROS_INFO("Element [%i]: [%f]", i, vector[i]);
      }
}
```

```
void roomba_net_cycle(std::vector<double> event, int& choice,
std::vector<double>& achne_out, std::vector<double>& n_out,
std::vector<double>& nm_out)
{
      double ACTION_SELECTION_THRESHOLD = 0.67;

      //parameters for state neuron activation function
      double N_ACT_GAIN = 2;
      double N_ACT_PERSIST = 0.25;
      double N_ACT_BASECURRENT = -1.0;
      std::vector<double> nprev = n;

      //parameters for neuromodulatory neuron activation function and
synaptic
      //plasticity
      //global NM;
      //global nm;
      std::vector<double> nmprev = nm;
      double NM_ACT_GAIN =2; //%2+(5-2)*rand; %2;    % gain for sigmoid
function
      double NM_ACT_BASECURRENT = -1.0;
      double NM_ACT_PERSIST = 0.25;  //% persistence of synaptic
current
      double NM_STP_GAIN = 1.1;  //% facillitating synapse
      double NM_STP_DECAY = 50;  //% recovery time constant
      double NM_STP_MAX = 2;     //% weight value ceiling

      //% parameters for ACh/NE neuron activation function and synaptic
plasticity
      //global achne;
      std::vector<double> achneprev = achne;
      double ACHNE_ACT_GAIN =  5; //%2+(5-2)*rand ; gain for sigmoid
function
      double ACHNE_ACT_BASECURRENT = -0.5;
      double ACHNE_ACT_PERSIST = 0.25;   //% persistence of synaptic
current
      double ACHNE_STP_GAIN = 0.1;   //% depressing synapse
      double ACHNE_STP_DECAY = 50;   //% recovery time constant
      double ACHNE_STP_MAX = 1;      //% weight value ceiling

      //calculate cholinergic/noradrenergic neural activity
      for (int i = 1; i <= e; i++)
      {
      //achne(i) =  activity (ACHNE_ACT_BASECURRENT + ACHNE_ACT_PERSIST
* achneprev(i) + event(i)*w_e_achne(i), ACHNE_ACT_GAIN);
      achne[i-1] =  activity(ACHNE_ACT_BASECURRENT +
ACHNE_ACT_PERSIST*achneprev[i-1] + event[i-1]*w_e_achne[i-1],
ACHNE_ACT_GAIN);
      ROS_INFO("achne[i-1] %f, achneprev[i-1] %f,",achne[i-
1],achneprev[i-1]);
```

```
      ROS_INFO("event[i-1] %f",event[i-1]);
      }

      /*
      //%achne = ones(1,4); % FOR DISTRACTED BEHAVIOR
      for (int i = 1; i <= e; i++) //FOR DISTRACTED BEHAVIOR
      {
      achne[i-1] = 1;
      }
      */

      // calculate neuromodulatory activity
      for (int i = 1; i <= NM; i++)
      {
      //I = NM_ACT_BASECURRENT + NM_ACT_PERSIST * nmprev(i);
      double I = NM_ACT_BASECURRENT + NM_ACT_PERSIST * nmprev[i-1];

      for (int j = 1; j <=e; j++)
      {
       //I = I + event(j)*w_e_nm(j,i);
            I = I + event[j-1]*w_e_nm[j-1][i-1];
           }

           //nm(i) =  activity (I, NM_ACT_GAIN);
      nm[i-1] =  activity (I, NM_ACT_GAIN);
     }


     // calculate state neural activity
      for (int i = 1; i <= N; i++)
      {
           //I = N_ACT_BASECURRENT+0.5*rand+N_ACT_PERSIST * nprev(i);
           double I = N_ACT_BASECURRENT+0.5*rand_num()+N_ACT_PERSIST *
nprev[i-1];

           //intrinsic synaptic input
           for (int j = 1; j <= N; j++)
           {
                //I = I + nprev(j) * w_n_n_exc(j,i) + (sum(nm)) *
nprev(j) * w_n_n_inh(j,i);
                I = I + nprev[j-1] * w_n_n_exc[j-1][i-1] +
(sum_vector(nm)) * nprev[j-1] * w_n_n_inh[j-1][i-1];
           }

           //event synaptic input
           for (int j = 1; j <= e; j++)
           {
                for (int k = 1; k <= NM; k++)
                {
                     //I = I + nm(k) * w_nm_n(k,i) * achne(j)*
event(j) * w_e_n(j,i);
```

```
                      I = I + nm[k-1] * w_nm_n[k-1][i-1] * achne[j-1]*
event[j-1] * w_e_n[j-1][i-1];
                  }
            }

            //n(i) = activity (I, N_ACT_GAIN);
            n[i-1] = activity (I, N_ACT_GAIN);
            ROS_INFO("n[i-1]: %f",n[i-1]);
      }

      //update plastic weights with short-term plasticity rule. a spike
occurs
      //when an event occurs.
      for (int i = 1; i <= e; i++)
      {
            //w_e_achne(i) = stp (w_e_achne(i), ACHNE_STP_GAIN,
ACHNE_STP_DECAY, ACHNE_STP_MAX, event(i) > 0.5);
            w_e_achne[i-1] = stp (w_e_achne[i-1], ACHNE_STP_GAIN,
ACHNE_STP_DECAY, ACHNE_STP_MAX, event[i-1] > 0.5);
      }

      for (int i = 1; i <= e; i++)
      {
            for (int j = 1; j <= NM; j++)
            {
                  //if w_e_nm (i,j) > 0
             //   w_e_nm (i,j) = stp (w_e_nm (i,j), NM_STP_GAIN,
NM_STP_DECAY, NM_STP_MAX, event(i) > 0.5);
            //end

            if ((w_e_nm[i-1][j-1]) > 0)
                  w_e_nm[i-1][j-1] = stp (w_e_nm[i-1][j-1], NM_STP_GAIN,
NM_STP_DECAY, NM_STP_MAX, event[i-1] > 0.5);
            }
      }

      //find most active state neuron. perform action selection if
activity is
      //above threshold
      //[y,i] = max(n)
      double y;
      double i_active;
      max_vector_element(n,y,i_active);
      if (y > ACTION_SELECTION_THRESHOLD){
            choice = i_active;}
      else
            {choice = -1;} //if no neuron active, no selection made
(selection depends on chc >= 0)

      achne_out = achne;
      n_out = n;
      nm_out = nm;
```

```
}

void CoreInfo(const kobuki_msgs::SensorState::ConstPtr& msg2)
//void IRInfo(const complete_test::irobotdock::ConstPtr& msg)
{
     //ROS_INFO("Battery voltage: [%i]",msg2->battery);

     if (battery_lock == 0){
     battery_initial_level = msg2->battery;
     battery_lock++; //locks up initial value of battery voltage
     }
     else{
     battery_level = msg2->battery;
     }



     //ROS_INFO("Bumper Values: [%i]", msg2->bumper);
     bumper = msg2->bumper;
     //ROS_INFO("Charge Values: [%i]", msg2->charger);
     charger = msg2->charger;
     //ROS_INFO("Wheel Drops: [%i]", msg2->wheel_drop);


}

void IRInfo(const kobuki_msgs::DockInfraRed::ConstPtr& msg)
//void IRInfo(const complete_test::irobotdock::ConstPtr& msg)
{
     //ROS_INFO("Data 0: [%i]",msg->data[0]);
     //ROS_INFO("Data 1: [%i]",msg->data[1]);
     //ROS_INFO("Data 2: [%i]",msg->data[2]);

     IR_Sensor[0] = msg->data[0]; //data to be passed onto main
subfunction
     IR_Sensor[1] = msg->data[1];
     IR_Sensor[2] = msg->data[2];

}

void LsrInfo(const sensor_msgs::LaserScan::ConstPtr& msg)
{
     //int times_run;

     //ros::param::set("/camera/depthimage_to_laserscan_loader/range_m
ax", 10.0);
     //ROS_INFO("Begin loop - LaserScan Info");
     //ROS_INFO("Range Min: [%f]", msg->range_min);
     //ROS_INFO("Range Max: [%f]", msg->range_max);
     //ROS_INFO("Size of Ranges: [%lu]",msg->ranges.size());
     float minVal = (msg->range_max); //to begin loop
```

```
      float finalVal = 0;
      for (int i = 0;i<(msg->ranges.size()); i++){
            if ((msg->range_min) <= (msg->ranges[i]) && (msg-
>ranges[i]) <= (msg->range_max)){  //if in range, valid dist.
                  if ((msg->ranges[i]) < minVal){
                                                //if lower than prev. min.
dist.
                        minVal = (msg->ranges[i]);
                        finalVal = minVal;
                  }
            }
      }





      ROS_INFO("Min. Dist: [%f]", finalVal);
      camera = finalVal;
      /*if (locate_enable > 0)
      {
        ROS_INFO("--DIRECTION RECORDED!!!!!!--");
        ROS_INFO("Location Enable: [%i]", locate_enable);
        locate[2] = finalVal;
        //ros::Duration(2).sleep();
      }
      */

      switch(locate_enable){
      case 1:
      locate[0] = finalVal;
      //ROS_INFO("LOC 1: [%f]", finalVal);
      break;
      case 2:
      locate[1] = finalVal;
      //ROS_INFO("LOC 2: [%f]", finalVal);
      break;
      case 3:
      locate[2] = finalVal;
      //ROS_INFO("LOC 3: [%f]", finalVal);
      break;
      case 4:
      locate[3] = finalVal;
      //ROS_INFO("LOC 4: [%f]", finalVal);
      break;
      case 5:
      locate[4] = finalVal;
      //ROS_INFO("LOC 5: [%f]", finalVal);
      break;
      }
      //ROS_INFO("BEFORE FUNCTION");
      //ROS_INFO("DIRECTION #: [%i]", dir_vector);
      //SetFront(dir_vector, finalVal);
      //ROS_INFO("AFTER FUNCTION");
      //dir_vector++;
      //ROS_INFO("DIRECTION #: [%i]", dir_vector);
      //WallFollow(finalVal);
   /*
      SetFront(finalVal);
      SetRightFront(finalVal);
      SetRight(finalVal);
      SetLeft(finalVal);
      SetLeftFront(finalVal);
   */
      //ROS_INFO("End loop - LaserScan Info");
}

void OdomInfo2(const nav_msgs::Odometry::ConstPtr& msg)
{


 if (locate_enable == 0){ //resets odometry

      std_msgs::Empty resetodom;

      double secs = ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs) <= 1.5){
      chatter_pub2.publish(resetodom);
      }
 }



 if (enable_sub == 1){ //to use vectorized distances
    //ROS_INFO("ODOM ACTIVATED!");
      double yaw;

      yaw = tf::getYaw(msg->pose.pose.orientation);
      //ROS_INFO("Yaw Angle: [%f]", yaw);

      geometry_msgs::Twist vel;
      float deg90 = (PI/2);
      float deg45 = (PI/4);
      float deg180 = (PI);
      float destrad;


      if (controlint == 0)
      {
           originyaw = yaw;
           ROS_INFO("Origin locked in: [%f]", originyaw);
```

```
            controlint++;
      }

      switch (stage){
      case 0: //turn right -45 degrees
      {
      locate_enable = 1;
      ROS_INFO("--Case 0---");
      destrad = DestinRad(originyaw, deg45);
      if ((tf::getYaw(msg->pose.pose.orientation) + (fromright*2*PI)) >
destrad){

      //ROS_INFO("--START Case 0--");
      //ROS_INFO("Current Yaw: [%f]", (tf::getYaw(msg-
>pose.pose.orientation) + (fromright*2*PI)));
      //ROS_INFO("Destrad: [%f]", destrad);
      //ROS_INFO("Stage: [%i]", stage);
      vel.linear.x = 0.0;
      vel.angular.z = -1.0;
      ROS_INFO("Origin locked in: [%f]", originyaw);
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      }
      else{
      ROS_INFO("--DONE Case 0--");
      vel.linear.x = 0.0;
      vel.angular.z = 0.0;
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      //ROS_INFO("90 degrees done");
      stage++;
      controlint= 0;
      //ros::Duration(3).sleep();
      }
      chatter_pub.publish(vel);
      //ROS_INFO("Stage: [%i]", stage);
      ROS_INFO("--PUBLISHING--");

      }
      break;
      case 1: //turn right 45 degrees
      {
      locate_enable = 2;
      ROS_INFO("--Case 1---");
      //ROS_INFO("Stage: [%i]", stage);
      destrad = DestinRad(originyaw, deg45);
      if ((tf::getYaw(msg->pose.pose.orientation) + (fromright*2*PI)) >
destrad)
      {
      vel.linear.x = 0.0;
      vel.angular.z = -1.0;
      ROS_INFO("Origin locked in: [%f]", originyaw);
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      }
      else
      {
      //locate_enable = 2;
      vel.linear.x = 0.0;
      vel.angular.z = 0.0;
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      //ROS_INFO("90 degrees done");
      stage++;
      controlint = 0;
      //ros::Duration(3).sleep();
      }
      chatter_pub.publish(vel);
      //ROS_INFO("--PUBLISHING--");
      }
      break;
      case 2: //turn left 180 degrees
      {
      locate_enable = 3;
      ROS_INFO("--Case 2---");
      destrad = DestinRad(originyaw, deg180);
      if ((tf::getYaw(msg->pose.pose.orientation) - (fromleft*2*PI)) <=
destrad)
      {
      vel.linear.x = 0.0;
      vel.angular.z = 1.0;
      ROS_INFO("Origin locked in: [%f]", originyaw);
      ROS_INFO("Yaw Angle Target (+180 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (+180 degrees):
[%f]",tf::getYaw(msg->pose.pose.orientation));
      }
      else
      {
      //locate_enable = 3;
      vel.linear.x = 0.0;
      vel.angular.z = 0.0;
      ROS_INFO("Yaw Angle Target (+180 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (+180 degrees):
[%f]",tf::getYaw(msg->pose.pose.orientation));
      //ROS_INFO("90 degrees done");
      stage++;
      controlint = 0;
      //ros::Duration(3).sleep();
      }
      chatter_pub.publish(vel);
      //ROS_INFO("--PUBLISHING--");
      }
      break;
      case 3: //turn right 45 degrees
      {
      locate_enable = 4;
      ROS_INFO("--Case 3---");
      destrad = DestinRad(originyaw, deg45);
      if ((tf::getYaw(msg->pose.pose.orientation) + (fromright*2*PI)) >
destrad)
      {
      vel.linear.x = 0.0;
      vel.angular.z = -1.0;
      ROS_INFO("Origin locked in: [%f]", originyaw);
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      }
      else
      {
      //locate_enable = 4;
      vel.linear.x = 0.0;
      vel.angular.z = 0.0;
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      //ROS_INFO("90 degrees done");
      stage++;
      controlint= 0;
      //ros::Duration(3).sleep();
      }
      chatter_pub.publish(vel);
      //ROS_INFO("--PUBLISHING--");
      }
      break;
      case 4: //turn right 45 degrees
      {
      locate_enable = 5;
      ROS_INFO("--Case 4---");
      destrad = DestinRad(originyaw, deg45);
      if ((tf::getYaw(msg->pose.pose.orientation) + (fromright*2*PI)) >
destrad)
      {
      vel.linear.x = 0.0;
      vel.angular.z = -1.0;
      ROS_INFO("Origin locked in: [%f]", originyaw);
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      }
      else
      {
```

```
      //locate_enable = 5;
      vel.linear.x = 0.0;
      vel.angular.z = 0.0;
      ROS_INFO("Yaw Angle Target (-45 degrees): [%f]", destrad);
      ROS_INFO("Yaw Angle Progress (-45 degrees): [%f]",tf::getYaw(msg-
>pose.pose.orientation));
      //ROS_INFO("90 degrees done");
      stage++;
      controlint= 0;
      //ros::Duration(3).sleep();
      }
      chatter_pub.publish(vel);
      //ROS_INFO("--PUBLISHING--");
      }
      break;
      case 5: //origin point
      {ROS_INFO("---DONE!----");

      //reveal_vector(locate);
      //ros::Duration(3).sleep();
      stage = 0;
      controlint = 0;
      enable_sub = 0; //turn off vectorized distances
      locate_enable = 0;
      }
      break;
      default:
      {
      ROS_INFO("--DEFAULT--");
      controlint = 0;
      stage = 0;
      }
      break;
      }

      }

 else{
      ROS_INFO("Turning subfunction not activated");
      }


}



void getinfo(double duration)
{     ROS_INFO("Inside GetInfo Loop!");
      double secs =ros::Time::now().toSec();
      ROS_INFO("Get Info Loop SECS! [%f]", secs);
      while ((ros::Time::now().toSec() - secs) <= duration)
      {
```

```
      ROS_INFO("Inside GetInfo While Loop!");
      //ROS_INFO("Sleeping");
      //ros::spinOnce();
      //ros::spin();

      //ros::AsyncSpinner spinner(4); // Use 4 threads
      //spinner.start();


      ROS_INFO("End of GetInfo While Loop!");
      }
      ROS_INFO("Done with GetInfo Loop!");

      //ros::AsyncSpinner spinner(4); // Use 4 threads
      //spinner.start();

}

void WallFollow(float bump, float distVal)
{

ROS_INFO("--WALL FOLLOW--");
ROS_INFO("Bump [%f]", bump);
ROS_INFO("DistVal [%f]", distVal);

geometry_msgs::Twist vel;
//int wall;
//float WALL_CLOSE = 1.5;
//float WALL_FAR = WALL_CLOSE + 2;
float WALL_CLOSE = 0.53;
//float WALL_FAR = WALL_CLOSE + 0.2;
float WALL_FAR = 0.77;
int condition;
//vel.angular.z = 0.0;
//vel.linear.x = 0.0;
float duration;


ROS_INFO("Wall Follow");
      if (bump == 1)
      {
      vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      ROS_INFO("Bump - Wall Follow");
      condition = 1;
      duration = 1.5;
      }
      else if (distVal > WALL_FAR)
      {
      vel.linear.x = 0.3;
      vel.angular.z = 0.4*(-wall);
      ROS_INFO("distVal > WALL_FAR");
      condition = 2;
      duration = 1.0;
      }
      else if (distVal < WALL_CLOSE)
      {
      vel.linear.x = 0.3;
      vel.angular.z = 0.4*wall;
      ROS_INFO("distVal < WALL_CLOSE");
      condition = 3;
      duration = 1.0;
      }
      else
      {
      vel.linear.x = 0.3;
      vel.angular.z = 0.0;
      ROS_INFO("Straight Ahead");
      condition = 4;
      duration = 1.0;
      }

      double secs =ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs) <= duration){
      chatter_pub.publish(vel);
      ROS_INFO("--Wall Follow Stats [%f], [%f]---", bump, distVal);
      ROS_INFO("--Wall Follow Parameters - Condition: [%i], Wall: [%i],
Left: [%f], Right: [%f]", condition, wall, locate[2], locate[1]);
      }
      //return;
}

void OpenField(float bump, float left, float front, float right){

ROS_INFO("--OPEN FIELD--");
ROS_INFO("--Left: [%f], Front: [%f], Right: [%f]--", left, front,
right);

float MAXSPEED = 0.40;
float MINSPEED = 0.10;
float MAXDIST = 300;
float TURN = 0.25;
geometry_msgs::Twist vel;
//int bump = 0;
float duration;

//find the most open area. speed is proportional to the amount of open
space
//go straight
if (front > left && front > right){
    if (front > MAXDIST){
        front = MAXDIST;}

    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---OPEN FIELD - Go Straight - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(front/MAXDIST)^2*MAXSPEED), 0.0);
    vel.linear.x = 0.3;
      vel.angular.z = 0.0;
      duration = 1.0;
      ROS_INFO("---OPEN FIELD - Go Straight - NO Bump---");
}
}

//go left
else if (left > right){
    if (left > MAXDIST){
        left = MAXDIST;}

    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---OPEN FIELD - Go Left - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(left/MAXDIST)^2*MAXSPEED), TURN);
    vel.linear.x = 0.3;
      vel.angular.z = 0.4;
      duration = 1.0;
      ROS_INFO("---OPEN FIELD - Go Left - NO Bump---");
}
}

else{ //go to the right
    if (right > MAXDIST){
        right = MAXDIST;}
    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---OPEN FIELD - Go Right - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(right/MAXDIST)^2*MAXSPEED), -1*TURN);
    vel.linear.x = 0.3;
```

```
      vel.angular.z = -0.4;
      duration = 1.0;
      ROS_INFO("---OPEN FIELD - Go Right - NO Bump---");
}
}

      double secs =ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs) <= duration){
           chatter_pub.publish(vel);}
}

void ExploreObject(float bump, float left, float front, float right){

ROS_INFO("--EXPLORE OBJECT--");
ROS_INFO("--Left: [%f], Front: [%f], Right: [%f]--", left, front,
right);

float MAXSPEED = 0.25;
float MINSPEED = 0.10;
float MAXDIST = 300;
float TURN = 0.5;
//int bump = 0;
geometry_msgs::Twist vel;

float duration;

//speed is proportional to the amount of change
if (front > left && front > right){ //% go straight
    if (front > MAXDIST){
        front = MAXDIST;}

    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---EXPLORE OBJ - Go Straight - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(front/MAXDIST)^2*MAXSPEED), 0.0);
    vel.linear.x = 0.3;
      vel.angular.z = 0.0;
      duration = 1.0;
      ROS_INFO("---EXPLORE OBJ - Go Straight - NO Bump---");
}
}


else if (left > right){     //% go to the left
    if (left > MAXDIST){
        left = MAXDIST;}
    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---EXPLORE OBJ - Go Left - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(left/MAXDIST)^2*MAXSPEED), TURN);
    vel.linear.x = 0.3;
      vel.angular.z = 0.4;
      duration = 1.0;
      ROS_INFO("---EXPLORE OBJ - Go Left - NO Bump---");
}
}

else{ //% go to the right
    if (right > MAXDIST){
        right = MAXDIST;}

    if (bump == 1){
        //SetFwdVelAngVelCreate(serport, 0.0, 2*TURN);
    vel.linear.x = 0.0;
      vel.angular.z = -0.8;
      duration = 1.5;
      ROS_INFO("---EXPLORE OBJ - Go Right - Bump---");
}
    else{
        //SetFwdVelAngVelCreate(serport,
max(MINSPEED,(right/MAXDIST)^2*MAXSPEED), -1*TURN);
      vel.linear.x = 0.3;
      vel.angular.z = -0.4;
      duration = 1.0;
      ROS_INFO("---EXPLORE OBJ - Go Right - NO Bump---");
}
}
      double secs =ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs) <= duration){
           chatter_pub.publish(vel);}
}

void FindHome(){
ROS_INFO("--FIND HOME--");
//system("roslaunch kobuki_auto_docking activate.launch --screen");
}

void AtHome(){
ROS_INFO("--AT HOME--");
}
```

```
void LeaveHome(){
ROS_INFO("--LEAVE HOME--");
geometry_msgs::Twist vel;

      double secs =ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs) <= 1.0){
      vel.linear.x = -0.3;
      vel.angular.z = 0.0;
      chatter_pub.publish(vel);
      }

      double secs2 =ros::Time::now().toSec();
      while ((ros::Time::now().toSec() - secs2) <= 1.0){
      vel.linear.x =  0.0;
      vel.angular.z = 0.3;
      chatter_pub.publish(vel);
      }
}



void main_neuromodulated_program_direct_sensor()
{
      //initialize neural network
      roomba_net_init();

      ros::AsyncSpinner spinner(4); // Use 4 threads
      spinner.start();

    ROS_INFO("Initialization Done!");
      double tic = ros::Time::now().toSec(); //start timer
      ROS_INFO("Tic Saved!");
      //ros::Duration(5).sleep();


      double behave_state_time = (ros::Time::now().toSec()) - tic;
      int behave_state = STATE_WALL_FOLLOW;
      int new_behave_state = behave_state;
      ROS_INFO("Behave State Time Saved!");
      //getinfo(2.5); //% initial battery level
      //ROS_INFO("GetInfo Activated!");
      double current_time = (ros::Time::now().toSec()) - tic;
      //loginx = 0;
      //running the network for approximately five minutes or whatever
minutes...
      ROS_INFO("Current Time Logged!: [%f]", current_time);
      //ros::Duration(5).sleep();
      while (current_time < 300){

      current_time = (ros::Time::now().toSec()) - tic;
      ROS_INFO("Inner Loop - Current Time Logged!: [%f]", 
current_time);
      //ros::Duration(5).sleep();


      //vectorize_sensor_reading -- done in Odom2 subroutine
      //vectorize_sensor_reading = [front right left right_close_front 
left_close_front]

      enable_sub = 1; //activate sensor vectoring


      while(enable_sub == 1){ //stays in loop until vectoring is done
      ROS_INFO("Sensor Vectoring in Progress - Main Subroutine 
Paused");
      }

      ROS_INFO("Sensor Vectoring Finished");

      ROS_INFO("WHILE LOOP DONE!!!!");
      //spinner.stop();
      //ros::Duration(15).sleep();


      //get bump sensor information -- done in CoreInfo subroutine
      //spinner.start();
      //get dock beam status - get information from IR and 
Core(charging) subroutine
      int beam; //true if home base detected

      if ((IR_Sensor[0] > 0) || (IR_Sensor[1] > 0) || (IR_Sensor[2] > 
0)){
      beam = 1;
      }
      else{
      beam = 0;
      }


      int homed; //homed if charging or close to the docking station

      if ((beam == 1) || (charger > 0)){
      homed = 1;
      }
      else{
      homed = 0;
      }

      battery = battery_level/battery_initial_level;
      //% get events (binary 1 == event occurred)
     //event(e_battery) = rand < (1-battery);  % event more likely as
battery level drops

     if (rand_num() < (1 - battery)){
     event[e_battery-1] = 1;
     }
     else{
     event[e_battery-1] = 0;
     }

     //event(e_beam) = beam ~=0;

     if (beam != 0){
     event[e_beam-1] = 1;
     }
     else{
     event[e_beam-1] = 0;
     }

      //event(e_bump) = BumpLeft || BumpRight || BumpFront ||
min(vectorize_sensor_reading) < too_close; % for 5 sensors

      if ((bumper > 0) || (min_vector(locate) < too_close)){
      event[e_bump-1] = 1;
      }
      else{
      event[e_bump-1] = 0;
      }

      if (min_vector(locate) < comparing_distance){
      event[e_ping_value-1] = 1;
      }
      else{
      event[e_ping_value-1] = 0;
      }
      //% for 5 sensors comparing_max_distance = 70

      //publish events here


      //special processing for returning home
      if (behave_state == STATE_FIND_HOME){

           //if near docking station transition to at home sub-state
           if (homed == 1){
            new_behave_state = STATE_AT_HOME;}
        //if timed out searching for docking station, abort search if
no
        //beam detected. if beam is detected continue searching
        else if ((current_time-behave_state_time) >
FIND_HOME_TIMEOUT){
           if (beam == 0)
                new_behave_state = STATE_LEAVE_HOME;
           else
                behave_state_time = (ros::Time::now().toSec()) - tic;
//there was a beam, try a little longer
           }
        }

      else if (behave_state == STATE_AT_HOME){
      new_behave_state = STATE_LEAVE_HOME;}
      else if (behave_state == STATE_LEAVE_HOME){
      new_behave_state = STATE_WALL_FOLLOW;}

      //action selection based on neural network activity. chc is non-
zero
    //if action is selected by the network
    else{ ROS_INFO("ROOMBA NET CYCLE ACTIVATED!");
      roomba_net_cycle(event, chc, achne, n, nm);
      if (chc >= 0){
           new_behave_state = (chc+1);}
    }

    /*
    spinner.stop();
    ROS_INFO("--State Neurons: n[0]: [%f]", n[0]);
    ROS_INFO("--State Neurons: n[1]: [%f]", n[1]);
    ROS_INFO("--State Neurons: n[2]: [%f]", n[2]);
    ROS_INFO("--State Neurons: n[3]: [%f]", n[3]);
    ROS_INFO("--Choice chc [%i]", chc);
    ROS_INFO("--New Behave State [%i]", new_behave_state);
      ros::Duration(15).sleep();

    spinner.start();
    */


      //transitioned to a new state, print state information
    if (behave_state != new_behave_state){
        behave_state_time = (ros::Time::now().toSec()) - tic;

      switch(new_behave_state){
      case 1: //case STATE_WALL_FOLLOW
           //if
min(vectorize_sensor_reading(3),vectorize_sensor_reading(5))<=50 %
while using 5 sensors
           //       left                         left_close_front
           if (min(locate[2], locate[3]) <= 0.77){ //wall value
                wall = WALL_LEFT;}
           else{
                wall = WALL_RIGHT;}
      ROS_INFO("State: WallFollow");
      ROS_INFO("--WALL--", wall);
      break;

      case 2: //case STATE_OPEN_FIELD
      ROS_INFO("State: OpenField");
      break;

      case 3: //case STATE_EXPLORE_OBJECT
      ROS_INFO("State: ExploreObject");
      break;

      case 4: //case STATE_FIND_HOME
      ROS_INFO("State: FindHome");
      break;

      case 5: //case STATE_AT_HOME
      ROS_INFO("State: AtHome");
      //dock_time = toc;
      dock_time = (ros::Time::now().toSec()) - tic;
      break;

      case 6: //case STATE_LEAVE_HOME
      ROS_INFO("State: LeaveHome");
      break;
      }

      behave_state = new_behave_state;
      }

      //handle states
     switch(behave_state){
      case 1: //case STATE_WALL_FOLLOW

           if (wall == 0){ //if wall hasn't been defined by previous
functions
                if (locate[2] < locate[1]){
                     wall = WALL_LEFT;
                }
                else if (locate[1] < locate[2]){
                     wall = WALL_RIGHT;
                }
           }


           //use the smallest ping sensor's value to the appropriate
wall for following
            if (wall == WALL_LEFT){
           //WallFollow (serRoombaport, event(e_bump), left);
           WallFollow (event[e_bump-1], locate[2]);}
            else{
                //WallFollow(serRoombaport, event(e_bump), right);
                WallFollow(event[e_bump-1], locate[1]);}

            break;


        case 2: //case STATE_OPEN_FIELD
           //OpenField (serRoombaport, event(e_bump), left, front,
right);
           OpenField(event[e_bump-1], locate[2], locate[4],
locate[1]);
           break;

        case 3: //case STATE_EXPLORE_OBJECT
           //ExploreObject (serRoombaport, event(e_bump), left, front,
right);
           ExploreObject(event[e_bump-1], locate[2], locate[4],
locate[1]);
           break;

      case 4: //case STATE_FIND_HOME
           //FindHome (serRoombaport);
           FindHome();
           break;

      case 5: //case STATE_AT_HOME
           //AtHome (serRoombaport);
           AtHome();
           break;

      case 6: //case STATE_LEAVE_HOME
           //LeaveHome (serRoombaport);
           LeaveHome();
           break;
    }

      //log state, event, and neural network information for post-
processing
      //loginx = loginx + 1;
     //log(loginx, 1) = current_time;
     logvec[0] = current_time;
     //log(loginx, 2) = behave_state;
     logvec[1] = behave_state;
     //log(loginx, 3:3+e-1) = event;
     logvec[2] = event[0];
     logvec[3] = event[1];
     logvec[4] = event[2];
     logvec[5] = event[3];
     //log(loginx, 3+e:3+e+N+NM+e-1) = [n nm achne];
     logvec[6] = n[0];
     logvec[7] = n[1];
     logvec[8] = n[2];
     logvec[9] = n[3];
     logvec[10] = nm[0];
     logvec[11] = nm[1];
     logvec[12] = achne[0];
     logvec[13] = achne[1];
     logvec[14] = achne[2];
     logvec[15] = achne[3];
     //log(loginx,17:20)= w_e_achne; % These weights are getting
 updated
     logvec[16] = w_e_achne[0];
     logvec[17] = w_e_achne[1];
     logvec[18] = w_e_achne[2];
     logvec[19] = w_e_achne[3];
     //log(loginx,21:22) = w_e_nm(1,1:2); % These weights are getting
 updated
     logvec[20] = w_e_nm[0][0];
     logvec[21] = w_e_nm[0][1];
     //log(loginx,23:24) = w_e_nm(2,1:2);% These weights are getting
 updated
     logvec[22] = w_e_nm[1][0];
     logvec[23] = w_e_nm[1][1];
     //log(loginx,25:26) = w_e_nm(3,1:2);% These weights are getting
 updated
     logvec[24] = w_e_nm[2][0];
     logvec[25] = w_e_nm[2][1];
     //log(loginx,27:28) = w_e_nm(4,1:2);% These weights are getting
 updated
     logvec[26] = w_e_nm[3][0];
     logvec[27] = w_e_nm[3][1];

     loginx.push_back(logvec);
     //spinner.start();
       } //end of timing while loop

  spinner.stop();
  //writing of data to xls
  ofstream f("./src/complete_test/src/results.xls");
       //f << m << " " << n << "n";

       //Data Headers
       f<<"current_time";
       f<<" ";
       f<<"behave_state";
       f<<" ";
       f<<"event[0]-ping";
       f<<" ";
       f<<"event[1]-battery";
       f<<" ";
       f<<"event[2]-bump";
       f<<" ";
       f<<"event[3]-beam";
       f<<" ";
       f<<"n[0]-Wall_Follow";
```

```
f<<" ";
f<<"n[1]-Open_Field";
f<<" ";
f<<"n[2]-Explore_Object";
f<<" ";
f<<"n[3]-Find_Home";
f<<" ";
f<<"nm[0]-DA";
f<<" ";
f<<"nm[1]-5HT";
f<<" ";
f<<"achne[0]-ping";
f<<" ";
f<<"achne[1]-battery";
f<<" ";
f<<"achne[2]-bump";
f<<" ";
f<<"achne[3]-beam";
f<<" ";
f<<"w_e_achne[0]-ping_ach";
f<<" ";
f<<"w_e_achne[1]-bat_ach";
f<<" ";
f<<"w_e_achne[2]-bmp_ach";
f<<" ";
f<<"w_e_achne[3]-bea_ach";
f<<" ";
f<<"w_e_nm[0][0]-ping_DA";
f<<" ";
f<<"w_e_nm[0][1]-ping_5HT";
f<<" ";
f<<"w_e_nm[1][0]-bat_DA";
f<<" ";
f<<"w_e_nm[1][1]-bat_5HT";
f<<" ";
f<<"w_e_nm[2][0]-bmp_DA";
f<<" ";
f<<"w_e_nm[2][1]-bmp_5HT";
f<<" ";
f<<"w_e_nm[3][0]-bea_DA";
f<<" ";
f<<"w_e_nm[3][1]-bea_5HT";
f<<"\n";


//writes data
for (int i = 0; i < loginx.size(); i++)
{
      for (int j = 0; j < loginx[i].size(); j++)
{
      f << loginx[i][j];
      f << " ";
      }
            f << "\n";
      }


}







int main(int argc, char **argv)
{

      // %Tag(INIT)%
      ros::init(argc, argv, "nuero");
      // %EndTag(INIT)%

      // %Tag(NODEHANDLE)%
      ros::NodeHandle node_handle;
      // %EndTag(NODEHANDLE)%

      //initialize neural network
      //roomba_net_init();

      ros::Duration(15).sleep();

      ros::Subscriber sub1 = node_handle.subscribe("/odom", 0,
OdomInfo2); //Internal Odometry sensors
      ros::Subscriber sub2 = node_handle.subscribe("/scan", 0,
LsrInfo); //Kinnect Camera
      ros::Subscriber sub3 =
node_handle.subscribe("/mobile_base/sensors/dock_ir", 0, IRInfo);
//kobuki IR sensors
      ros::Subscriber sub4 =
node_handle.subscribe("/mobile_base/sensors/core", 0, CoreInfo);
//kobuki core sensors
      chatter_pub =
node_handle.advertise<geometry_msgs::Twist>("/mobile_base/commands/vel
ocity", 1);
      chatter_pub2 =
node_handle.advertise<std_msgs::Empty>("/mobile_base/commands/reset_od
ometry", 1);
      main_neuromodulated_program_direct_sensor();
      //ros::spin();

      /*
      int chc;
      for (int i = 1;i <=10;i++)
      {
```

```
      ROS_INFO("---- LOOP NUMBER %i ----",i);
      roomba_net_cycle(event, chc, achne, n, nm);
      ROS_INFO("It worked.");
      ROS_INFO("Chc: %i",chc);
      }

      event[0] = 1;
      for (int j = 11;j <=20;j++)
      {
      ROS_INFO("---- LOOP NUMBER %i ----",j);
      roomba_net_cycle(event, chc, achne, n, nm);
      ROS_INFO("It worked.");
      ROS_INFO("Chc: %i",chc);
      }
      */
}
```

EXAMPLE OF SPIKING NEURAL NETWORK SIMULATION CODE IN C++

```
/*
 * Copyright (c) 2013 Regents of the University of California. All
rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above
copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 *
 * 3. The names of its contributors may not be used to endorse or
promote
 *    products derived from this software without specific prior
written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
```

```
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
**********************************************************************
************************ *
 * CARLsim
 * created by:        (MDR) Micah Richert, (JN) Jayram M. Nageswaran
 * maintained by:     (MA) Mike Avery <averym@uci.edu>, (MB) Michael
Beyeler <mbeyeler@uci.edu>,
 *                         (KDC) Kristofor Carlson <kdcarlso@uci.edu>
 *
 * CARLsim available from
http://socsci.uci.edu/~jkrichma/CARL/CARLsim/
 * Ver 10/09/2013
 */

#include <snn.h>
#include <ROSevent.h>

//#include "../../../testbed/src/complete_test/src/vector_maker.cpp"

#define N    1000

#define action 70
#define actionin 1000

#define actionN N*0.1
#define eventN N*0.1

int main()
{
     // create a network
     CpuSNN s("global");

        int event_value = 2; // 0; // 0; // 0; // 0; // 6; // 6; // 6;
// 4; // 0; // 6; // 12; // 4; // 0;
        cout << "The value is: " << event_value;
        //std::cin.ignore().get();

        //ACh/NE group
     int ach=s.createGroup("ach", 4, EXCITATORY_NEURON);
```

```
      s.setNeuronParameters(ach, 0.02f, 0.2f, -65.0f, 8.0f);

      //dopamine group
      int dopa=s.createGroup("dopa", N, EXCITATORY_NEURON);
      s.setNeuronParameters(dopa, 0.02f, 0.2f, -65.0f, 8.0f);

      //serotonin group
      int sero=s.createGroup("sero", N,  EXCITATORY_NEURON);
      s.setNeuronParameters(sero, 0.02f, 0.2f, -65.0f, 8.0f);


      //action neuron groups
      int openfield=s.createGroup("open_field",actionN,
EXCITATORY_NEURON);
      s.setNeuronParameters(openfield, 0.02f, 0.2f, -65.0f, 8.0f);

      int explore=s.createGroup("explore",actionN, EXCITATORY_NEURON);
      s.setNeuronParameters(explore, 0.02f, 0.2f, -65.0f, 8.0f);

      int wallfollow=s.createGroup("wall_follow",actionN,
EXCITATORY_NEURON);
      s.setNeuronParameters(wallfollow, 0.02f, 0.2f, -65.0f, 8.0f);

      int findhome=s.createGroup("find_home",actionN,
EXCITATORY_NEURON);
      s.setNeuronParameters(findhome, 0.02f, 0.2f, -65.0f, 8.0f);

      //event neuron input groups (battery, bump, home, object)
      int
ginbat=s.createSpikeGeneratorGroup("input_battery",eventN,EXCITATORY_N
EURON);
      int
ginbmp=s.createSpikeGeneratorGroup("input_bump",eventN,EXCITATORY_NEUR
ON);
      int
ginhome=s.createSpikeGeneratorGroup("input_home",eventN,EXCITATORY_NEU
RON);
      int
ginobj=s.createSpikeGeneratorGroup("input_object",eventN,EXCITATORY_NE
URON);

      //***CONNECTIONS***

      //serotonin --> dopamine, full connection
      s.connect(sero,dopa,"full", -1.0f, -1.0f, 1.0f, 1, 1, SYN_FIXED);

      //ACh/NE --> serotonin
      s.connect(ach,sero,"full", +1.0f, +1.0f, 1.0f, 1, 20,
SYN_PLASTIC);//non-distracted


      //ACh/NE --> dopamine
```

```
	s.connect(ach,dopa,"full", +1.0f, +1.0f, 1.0f, 1, 20, 
SYN_PLASTIC);//non-distracted


	//openfield/explore -->dopamine
	s.connect(openfield,dopa,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(explore,dopa,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);

	//wallfollow/findhome --> serotonin
	s.connect(wallfollow,sero,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(findhome,sero,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);

	//dopamine --> openfield/explore
	s.connect(dopa,openfield,"full", +5.0f, +5.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(dopa,explore,"full", +5.0f, +5.0f, 1.0f, 1, 1, 
SYN_FIXED);

	//serotonin --> wallfollow/findhome
	s.connect(sero,wallfollow,"full", +5.0f, +5.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(sero,findhome,"full", +5.0f, +5.0f, 1.0f, 1, 1, 
SYN_FIXED);

	//OFC-PFC all-to-all
	//openfield everything
	s.connect(openfield,explore,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(openfield,wallfollow,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(openfield,findhome,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);

	s.connect(openfield,explore,"full", +1.0f, +1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(openfield,wallfollow,"full", +1.0f, +1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(openfield,findhome,"full", +1.0f, +1.0f, 1.0f, 1, 1, 
SYN_FIXED);

	//explore everything
	s.connect(explore,openfield,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(explore,wallfollow,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
	s.connect(explore,findhome,"full", -1.0f, -1.0f, 1.0f, 1, 1, 
SYN_FIXED);
```

```
	s.connect(explore,openfield,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(explore,wallfollow,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(explore,findhome,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);

	//wallfollow everything
	s.connect(wallfollow,openfield,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(wallfollow,explore,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(wallfollow,findhome,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);

	s.connect(wallfollow,openfield,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(wallfollow,explore,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(wallfollow,findhome,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);

	//findhome everything
	s.connect(findhome,openfield,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(findhome,explore,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(findhome,wallfollow,"full", -1.0f, -1.0f, 1.0f, 1, 1,
SYN_FIXED);

	s.connect(findhome,openfield,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(findhome,explore,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);
	s.connect(findhome,wallfollow,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);

	//bump
	//s.connect(ginbmp,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_PLASTIC); //non-distracted
	s.connect(ginbmp,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED); //distracted
	s.connect(ginbmp,dopa,"full", +0.5f, +0.5f, 1.0f, 1, 1,
SYN_FIXED); //risk taking
	//s.connect(ginbmp,sero,"full", +0.5f, +0.5f, 1.0f, 1, 1,
SYN_FIXED); //risk aversive
	//s.setSpikeRate(ginbmp,&in);

	//home
	  //s.connect(ginhome,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_PLASTIC);//non-distracted
```

```
        s.connect(ginhome,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED); //distracted
     s.connect(ginhome,sero,"full", +0.5f, +0.5f, 1.0f, 1, 1,
SYN_FIXED);
     //s.setSpikeRate(ginlas,&in);

     //battery
     //s.connect(ginbat,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_PLASTIC);//non-distracted
     s.connect(ginbat,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);//distracted
     s.connect(ginbat,sero,"full", +0.5f, +0.5f, 1.0f, 1, 1,
SYN_FIXED);
     //s.setSpikeRate(ginbat,&in);

     //object
     //s.connect(ginobj,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_PLASTIC);//non-distracted
     s.connect(ginobj,ach,"full", +1.0f, +1.0f, 1.0f, 1, 1,
SYN_FIXED);//distracted
     s.connect(ginobj,dopa,"full", +0.5f, +0.5f, 1.0f, 1, 1,
SYN_FIXED);
     //s.setSpikeRate(ginex,&in);

//   // make random connections with 10% probability
//   s.connect(g2,g1,"random", -1.0f/100, -1.0f/100, 0.1f, 1, 1,
SYN_FIXED);
//   // make random connections with 10% probability, and random
delays between 1 and 20
//   s.connect(g1,g2,"random", +0.25f/100, 0.5f/100, 0.1f,  1, 20,
SYN_PLASTIC);
//   s.connect(g1,g1,"random", +6.0f/100, 10.0f/100, 0.1f,  1, 20,
SYN_PLASTIC);

//   // 5% probability of connection
//   s.connect(gin,g1,"random", +100.0f/100, 100.0f/100, 0.05f,  1,
20, SYN_FIXED);

//   float COND_tAMPA=5.0, COND_tNMDA=150.0, COND_tGABAa=6.0,
COND_tGABAb=150.0;
//
     s.setConductances(ALL,true,COND_tAMPA,COND_tNMDA,COND_tGABAa,COND
_tGABAb);

//   // here we define and set the properties of the STDP.
//   float ALPHA_LTP = 0.10f/100, TAU_LTP = 20.0f, ALPHA_LTD =
0.12f/100, TAU_LTD = 20.0f;
//   s.setSTDP(g1, true, ALPHA_LTP, TAU_LTP, ALPHA_LTD, TAU_LTD);

//   // show logout every 10 secs, enabled with level 1 and output to
stdout.
//   s.setLogCycle(10, 1, stdout);
//    // put spike times into spikes.dat
//    s.setSpikeMonitor(g1,"results/ROSSNN/spikes.dat");

//    // Show basic statistics about g2
//    s.setSpikeMonitor(g2);

//    s.setSpikeMonitor(gin);

      float COND_tAMPA=5.0, COND_tNMDA=150.0, COND_tGABAa=6.0,
COND_tGABAb=150.0;
      s.setConductances(ALL,true,COND_tAMPA,COND_tNMDA,COND_tGABAa,COND
_tGABAb);

      // here we define and set the properties of the STDP.
      float ALPHA_LTP = 0.10f/100, TAU_LTP = 20.0f, ALPHA_LTD =
0.12f/100, TAU_LTD = 20.0f;
      s.setSTDP(dopa, true, ALPHA_LTP, TAU_LTP, ALPHA_LTD, TAU_LTD);
      s.setSTDP(sero, true, ALPHA_LTP, TAU_LTP, ALPHA_LTD, TAU_LTD);
      s.setSTDP(ach, true, ALPHA_LTP, TAU_LTP, ALPHA_LTD, TAU_LTD);

      // put spike times into spikes.dat
      s.setSpikeMonitor(ach,"results/ROSSNN/ach_spikes.dat");
      s.setSpikeMonitor(sero,"results/ROSSNN/sero_spikes.dat");
      s.setSpikeMonitor(dopa,"results/ROSSNN/dopa_spikes.dat");

      s.setSpikeMonitor(openfield,"results/ROSSNN/openfield_spikes.dat"
);
      s.setSpikeMonitor(explore,"results/ROSSNN/explore_spikes.dat");
      s.setSpikeMonitor(wallfollow,"results/ROSSNN/wallfollow_spikes.da
t");
      s.setSpikeMonitor(findhome,"results/ROSSNN/findhome_spikes.dat");


      //setup some baseline input
      PoissonRate in(eventN);
      for (int i=0;i<eventN;i++) in.rates[i] = 1;

      PoissonRate nin(eventN);
      for (int i=0;i<(eventN);i++) in.rates[i] = 5;


      switch(event_value){

      case 0: //no inputs
      s.setSpikeRate(ginbat,&nin); //battery - 0
      s.setSpikeRate(ginhome,&nin); //home - 0
      s.setSpikeRate(ginbmp,&nin); //bump - 0
      s.setSpikeRate(ginobj,&nin); //object - 0
      break;

      case 1: //battery
```

```
s.setSpikeRate(ginbat,&in); //battery - 1
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&nin); //bump - 0
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 2: //beam
s.setSpikeRate(ginbat,&nin); //battery - 0
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&nin); //bump - 0
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 3: //beam+battery
s.setSpikeRate(ginbat,&in); //battery - 1
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&nin); //bump - 0
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 4: //bump
s.setSpikeRate(ginbat,&nin); //battery - 0
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&in); //bump - 1
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 5: //bump+ battery
s.setSpikeRate(ginbat,&in); //battery - 1
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&in); //bump - 1
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 6: //bump+beam
s.setSpikeRate(ginbat,&nin); //battery - 0
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&in); //bump - 1
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 7: //bump+beam+battery
s.setSpikeRate(ginbat,&in); //battery -1
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&in); //bump - 1
s.setSpikeRate(ginobj,&nin); //object - 0
break;

case 8: //object
s.setSpikeRate(ginbat,&nin); //battery - 0
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&nin); //bump - 0
```

```
s.setSpikeRate(ginobj,&in); //object - 1
break;

case 9: //object+battery
s.setSpikeRate(ginbat,&in); //battery - 1
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&nin); //bump - 0
s.setSpikeRate(ginobj,&in); //object - 1
break;

case 10: //object + beam
s.setSpikeRate(ginbat,&nin);  //battery - 0
s.setSpikeRate(ginhome,&in);//home - 1
s.setSpikeRate(ginbmp,&nin); //bump - 0
s.setSpikeRate(ginobj,&in); //object - 1
break;

case 11: //object+beam+battery
s.setSpikeRate(ginbat,&in);   //battery -1
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&nin);  //bump - 0
s.setSpikeRate(ginobj,&in);  //object - 1
break;

case 12: //object+bump
s.setSpikeRate(ginbat,&nin);   //battery - 0
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&in);  //bump - 1
s.setSpikeRate(ginobj,&in);  //object - 1
break;

case 13: //object+bump+battery
s.setSpikeRate(ginbat,&in);   //battery -1
s.setSpikeRate(ginhome,&nin); //home - 0
s.setSpikeRate(ginbmp,&in);  //bump - 1
s.setSpikeRate(ginobj,&in);  //object - 1
break;

case 14: //object+bump+beam
s.setSpikeRate(ginbat,&nin); //battery -0
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&in);  //bump - 1
s.setSpikeRate(ginobj,&in);  //object - 1
break;

case 15: //all inputs
s.setSpikeRate(ginbat,&in);  //battery -1
s.setSpikeRate(ginhome,&in); //home - 1
s.setSpikeRate(ginbmp,&in);  //bump - 1
s.setSpikeRate(ginobj,&in);  //object - 1
break;
```

```
      default:
      s.setSpikeRate(ginbat,&nin); //battery - 0
      s.setSpikeRate(ginhome,&nin); //home - 0
      s.setSpikeRate(ginbmp,&nin); //bump - 0
      s.setSpikeRate(ginobj,&nin); //object - 0
      break;
      }

//    s.setSpikeRate(ginbat,&nin); //battery
//    s.setSpikeRate(ginhome,&nin); //home
//    s.setSpikeRate(ginbmp,&nin); //bump
//    s.setSpikeRate(ginobj,&nin); //object


      //run for 10 seconds
      for(int i=0; i < 1; i++) {
            // run the established network for a duration of 1 (sec)
and 0 (millisecond), in CPU_MODE
            s.runNetwork(1, 0, GPU_MODE);
      }

      FILE* nid = fopen("results/ROSSNN/network.dat","wb");
      s.writeNetwork(nid);
      fclose(nid);

      return 0;
}

//FreeSpace
```

MATLAB CODE TO RUN AN IZHIKEVICH PULSE TRAIN

```
n = 1000;

t = 1:1:n;

v = ones(1,n);
u = ones(1,n);
I = [zeros(1,100),10*ones(1,900)];

a = 0.02*ones(1,n);
b = 0.2*ones(1,n);
c = -65*ones(1,n);
d = 8*ones(1,n);
%dv = [];
%du = [];
vma = [];
uma = [];


for e = 1:n

     if v(e) >= 30
        v(e) = 30;
        v(e+1) = c(e);
        u(e+1) = u(e) + d(e);
        vma = [vma v(e)];
        uma = [uma u(e)];
     else
        vma = [vma v(e)];
        uma = [uma u(e)];
        dv = (0.04*v(e)^2) + (5*v(e)) + 140 - u(e) + I(e);
        du = a(e)*(b(e)*v(e) - u(e));
        v(e+1) = v(e) + dv;
        u(e+1) = u(e) + du;
    end



end

subplot(2,1,1)
plot(t,vma)

subplot(2,1,2)
plot(t,uma)
```

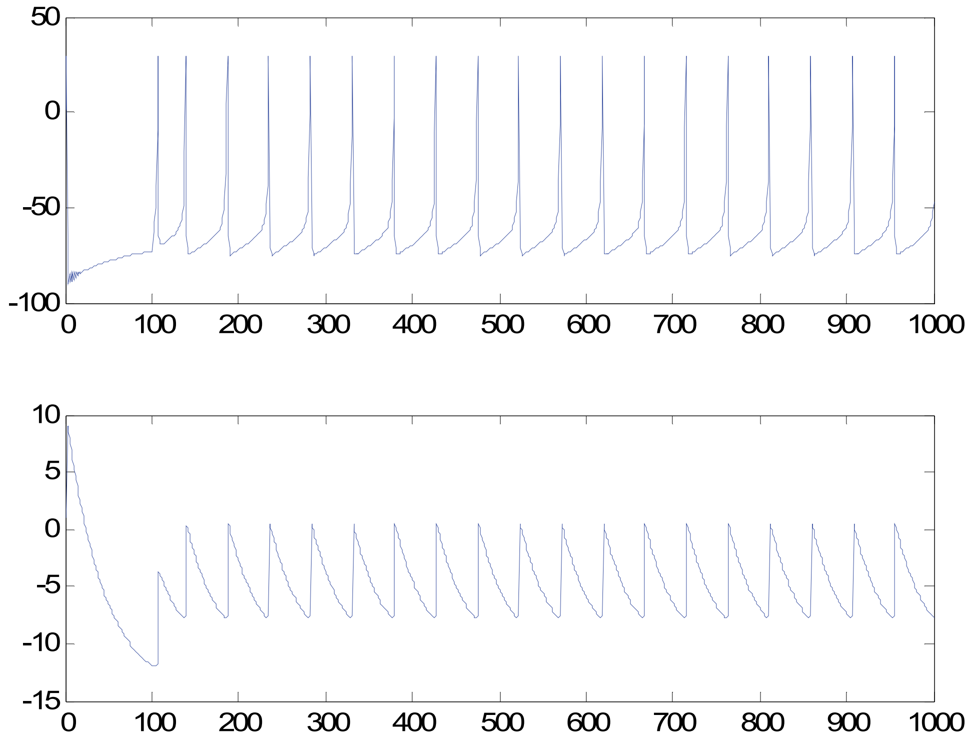

Figure A1. Top graph - action potentials, Bottom graph – recovery variable

Appendix B

PICTURES OF ROBOT MOTION

B.1: Sequence of robot motion in risk aversive mode:

B.2: Sequence of robot motion in risk taking mode:

B.3: Sequence of robot motion in distracted mode:

Appendix C

LIST OF PUBLICATIONS

Muhammad, Cameron and Biswanath Samanta. "Neuromodulation Based Control of an Autonomous Robot in a Safe Cloud Environment" AUVSI (Association for Unmanned Vehicle Systems International) Unmanned Systems 2014.

Muhammad, Cameron and Biswanath Samanta. "Control of Autonomous Robots using principles of Neuromodulation in ROS Environment" ASME 2014 International Mechanical Engineering Congress & Exposition. 2014.

Muhammad, Cameron and Biswanath Samanta. "Neuromodulation Based Control of Autonomous Robots in ROS Environment" IEEE Symposium Series on Computational Intelligence (IEEE SSCI)-Cognitive Algorithms, Mind, and Brain (CCMB) 2014.