Simplifying Neurorobtic Development with NCSTools

C. M. Thibeault[1,2,3,*], J. Hegie[2], L. Jayet Bray[2,3], and F. C. Harris Jr.[2,3]

[1] *Department of Electrical and Biomedical Engineering, University of Nevada, Reno. Reno, NV.*
[2] *Department of Computer Science, University of Nevada, Reno. Reno, NV.*
[3] *Brain Computation Lab, University of Nevada, Reno. Reno, NV.*

Abstract

The combination of biologically realistic neural simulations and robotic agents offers unique opportunities for both computational neuroscience and research in intelligent robotics. A concept that can provide insights into the cognitive developments involved in human robotic interaction as well as provide a pathway to developing truly intelligent agents. In this paradigm spiking neural models are coupled with physical or virtual entities in a closed-loop. The embodied agent provides stimulus to the neural model which in turn provides a filtered and processed view of that world. More often than not the complexity and corresponding computational burden of these models necessitates the use of high-performance computers that must be stored and maintained separate from the entity. In these instances, closing the loop can be an arduous task requiring intimate collaboration between neuroscientists and engineers. Presented here is a software package for simplifying those interactions while facilitating the communication between neural simulations and abstract remote entities.

Keywords: *Neurorobotics, Spiking Neural Network Simulation, Autonomous Agents, Intelligent Robotics.*

1 Introduction

Computational neuroscience enjoys a unique role in biological research. At one end it can help validate and quantify experimental results. While at the other, it provides a predictive mechanism for aspects of the nervous system that are unreachable by any other means. In addition to the importance mathematically modeling the nervous system has to physiological research is its value to artificial intelligence and autonomous agents.

By employing spiking neural models as the processing elements for robotic agents, researchers are attempting to explore theories that span the breadth and depth of robotics, AI and neuroscience. Understanding neurological processing often requires complex interactions with the real world (or a virtual one). This is the case in the fields of both social robotics [1, 2] and neurorobotics [3–5]. These theories generally involve the integration of several sensory modalities as well as complex commands between the agent and the neural architectures controlling it. This integration can be a complex task requiring either expertise in both computer engineering and neuroscience or collaborations between experts in each of these fields. The software, NCSTools, presented here was developed to ease the complexity of interfacing neural models with remote agents as well as abstract the neurological detail from the roboticists and the engineering detail from the neuroscientists.

This paper is laid out with the remainder of this section presenting a minimal background on spiking neural models and neurorobotics as well as introducing NCSTools. This is followed by Section 2, which illustrates the design choices made in this project as well as the basic software engineering behind its implementation. Section 3 provides a complete example of how NCSTools can be used to speed-up the task of interfacing with a spiking neural simulation. Finally, Section 4 concludes with some current applications where NCSTools has been utilized as well as some future directions.

The work presented here is an extension of a presentation given at the Computational Neuroscience Society Meeting in San Antonio, Texas, August 2010 [6].

1.1 Spiking Neural Models

Simulating excitable cells involves integrating a set of differential equations that describe the electrical activity along the membrane of the cell. The unique aspects of these cells is that once a threshold voltage has been reached, an all or nothing avalanche of electrical current occurs. This results in a spike of electrical activity, or an action potential, and initiates the communication between neurons. Its effect is felt by all neurons connected to the one that spiked, or fired. Additionally, repeated spike events can alter or grade the effect felt by those downstream neurons. This modification is referred to as synaptic level plasticity and is thought to play a major role in animal learning [7]. By combining the differential

*Corresponding Author. Corey@cmthibeault.com

equations describing membrane voltages with the synaptic communication, computational neuroscientists hope to reveal details of the nervous system unavailable with current experimental techniques.

1.2 Neurorobotics

The field of neurorobotics focuses on the coupling of neural systems with some form of physical actuation. An example of this is the concept of virtual neurorobotics (VNR). This is based around the interoperability of a neural model, a virtual robotic avatar and a human participant [8, 9]. Under all but the most basic scenarios this interoperability is accomplished through an organized network communication system. For this paper an emphasis is placed on robotic and automated agents; however, it should be noted that the tools described here are by no means limited to that application.

1.3 The NeoCortical Simulator

The NeoCortical Simulator (NCS) was developed at The University of Nevada, Reno by the Brain Computation Lab under the direction of Dr. Phillip Goodman. From its inception a heavy emphasis has been placed on parallelization and performance. In addition, mechanism for getting spiking information out and stimulus in was also extremely important. Despite the focus on performance, NCS provides a number of important biological models. For a review of what NCS has to offer refer to Wilson *et al.* [10, 11] and to see how NCS compares to other neural simulators see Brette *et al.* [12]. Its features can be summarized as:

- High-performance MPI-based parallel architecture
- Leaky Integrate-and-fire (LIF) neurons with conductance based synapses and Hodgkin-Huxley channels.
- Hebbian synaptic learning with Short-term plasticity, augmentation, and spike-timing dependent plasticity (STDP).

1.4 NCSTools

NCSTools is a bridge between neuroscientists and engineers using NCS for research. The strength of NCSTools lies in its configuration language. It provides a mechanism for defining interactions between the neural simulation and the agent. These interactions are described using plain text strings and developers simply need to agree on the strings to develop their components. Besides the benefit that neither end is required to know intricacies of the other, this also provides a level of reuse that can reduce development time. The only requirement is that the strings remain consistent. This greatly reduces the development time of a neurorobotic application.

2 Design

NCSTools was developed in C++ with a focus on object-oriented design principles. The motivation for its construction was driven by the need for a replacement to the previous software package, Brainstem [13]. Although successful as a proof-of-concept, Brainstem lacked the necessary extensibility and reliability required for rapid use by researchers. Based on these inadequacies, several non-functional requirements were identified before starting the development of NCSTools. These were

1. **Usability:** The use of NCSTools must be relatively simple. Although its intended users are scientists and engineers, its operation has to make sense based on the task.
2. **Extensibility:** The rapid pace of scientific research necessitates a system that can be readily extended to incorporate new ideas and concepts.
3. **Robustness:** The codebase must be invariant to the application and multitude of configurations.
4. **Reliability:** NCSTools is intended for researchers performing experiments in both neuroscience and robotics. If it is unreliable, meaningful results can be lost.

The component interfaces provide layered abstraction supporting an extensible yet robust code base. Similarly, the configuration language supports overall usability. Throughout this section numerical subscripts that correspond to the non-functional requirements are used to indicate how each design element supports the requirements.

Figure 1 presents the system level layout of NCSTools. The major components are described below.

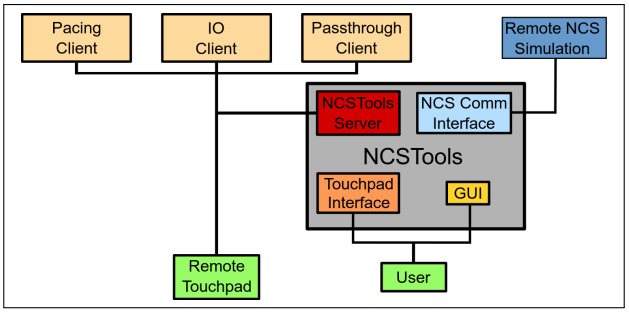

Figure 1: System layout. With the exception of the user interfaces, all connections are made using the various network clients described below.

```
input:
{
  NCSReportCollection1: {
    num_reports = 500;
    # The total period of counting, includes
    # report collection and recovery.
    period = 500;
    # The number of reports.
    NCSReports = 2;
    type = "STANDARD";
    NCSReport1: {
      connection = "to_PMC1";
      command = "point_left";
      };
    NCSReport2: {
      connection = "to_PMC2";
      command = "point_right";
    };
    # Setup the plots for this group.
    plot = "YES";
    plotType = "BAR";
    plotname = "Motor Activity";
    tabIndex = 1;
    plotIndex = 1;
  };
};
```

Listing 1: Input configuration example. This is a fixed window input container with a window of 500 spike reports from NCS. After that time the activity from the named connections "to_PMC1" and "to_PMC2" are compared. The command corresponding to the winning channel is sent to the connection clients. For instance, if the "to_PMC1" channel is more active the command "point_left" is sent to the connected agents. The last section is used to configure if, where, and how this activity will be presented on the user interface.

2.1 Configuration Language[1]

The flexibility of NCSTools lies in its configuration language. Most aspects are modifiable at runtime through the input configuration file. This includes control for the definable communication "language", the GUI, User IO and Data processing. There are two examples of the configuration language provided in Listings 1 and 2. The component of NCSTools that these correspond to are described in Sections 2.2 and 2.3 respectively.

2.2 Inputs[2,3]

In the context of this paper, inputs are the signals coming from the NCS simulation specified as reports. NCS publishes population level information about the simulation to each of the reports requested. This information can be spike-counts for individual neurons, sampled synaptic weights and neuron voltages.

Input Containers

The base class for all inputs is the input container. This provides the common functionality, such as initialization and communication, as well as the interface for all derived classes.

Windowed Input Containers

Windowed input containers allow the user to specify a window of time over which the spiking activity of any number of neuron populations can be compared in a winner-takes-all pattern. The window of time is determined by how often the NCS simulation sends spiking information. There are currently two types of derived window input containers, one uses a fixed amount of time and the other uses a moving window.

The fixed time container compares the spiking activity over a static window. After the defined period of NCS reports has elapsed the most active population (the one with the highest number of spikes) is selected. The command specified in the configuration file for that report channel is then sent to all connected clients. An example configuration for a fixed time container is presented in Listing 1.

The second input container uses a moving window of time. This window is fixed width but progresses in time as the simulation progresses. This continues until the most active population's activity is greater than the next most active by a user specified threshold. At that time the command associated with that report is sent out and the window is reset.

A key feature of both the inputs and the outputs, described in Section 2.3 below, is the ability to bind the same command string to multiple containers. This provides a mechanism for starting a coordinated series of events based on a single client command or simulation result.

In addition to sending commands to the connected agents these containers can also be used for plotting in the GUI as described below.

2.3 Outputs[2,3]

The NCSTools outputs define the signals traveling to the NCS simulations. These are stimulus sent to populations and can be triggered by text strings from the connected clients or by the built-in touchpad interface described below.

Static Output

The stimulus sent to the neuron population is fixed and defined by the configuration file.

Dynamic Output

The stimulus sent to the neuron population is sent by the client along with the string command.

Timed Output

Timed output containers are used to send the same stimulus a set number of times. Through the configuration file the user can specify how many times to send the

```
output:
{
  NCSStim1: {
    type = "TIMED_OUTPUT";
    command = "saw_red";
    connection = "from_VC1";
    # The static output to send.
    output = "0.2000";
    # How many ticks between stim inputs.
    frequency = 50;
    # The number of times to repeat the input.
    num_outputs = 10;;
  };
  NCSStim2: {
    type = "TIMED_OUTPUT";
    command = "saw_blue";
    connection = "from_childbot_VC1";
    output = "0.0000";
    frequency = 50;
    num_outputs = 10;;
  };
};
```

Listing 2: Output configuration example. Two outputs are configured here; each tied to a different command from the connected agents. These are both timed outputs where the stimulus, in this case a value of 0.2000, is sent every 50 simulation reports a total of 10 times. Where the stimulus is sent depends on the string received from the connected clients.

stimulus and the interval between successive stimulus. An example of this is presented in Listing 2.

2.4 Network Communication[3,4]

Aside from the user interface, the connections illustrated in Figure 1 represent one or multiple network communication mechanisms. NCSTools not only coordinates the connection to and from the NCS neural simulation but also provides a network server for handling client and remote agent connections. The individual components of the network communication provided by NCSTools are described below.

2.4.1 NCSTools Server[4]

NCSTools uses a simple POSIX socket server for client communication. The simplicity of the server helps ensure its reliability and the low-level components help guarantee that the performance of the server does not hinder the overall application.

2.4.2 Client Communication[2,3,4]

Several client implementations are provided for C++, Python and MATLAB/Java. These provide objects that appropriately abstract the interface from the implementation to support the extensibility of both NCSTools and client applications. Both blocking and non-blocking communication is supported.

IO Clients

The IO clients are used for most applications. These provide the input and output mechanisms required to interact with NCSTools.

Pacing Clients

As neural models increase in both size and complexity they often exceed the real-time capabilities of the hardware. Pacing clients are provided to ensure that remote clients do not overfill buffers or lose track of the simulation. These connect to NCSTools to maintain synchronization and receive heartbeats that are output at user specified intervals signaling the current time in the simulation.

Passthrough Clients

Passthrough clients connect to the NCSTools server and receive messages sent to or from other clients connected to the server, including the NCS simulation. These clients can provide users with a complete landscape of the neurorobotic experiment. These can also be used to create context aware clients that can modify their behavior based on the state of the system.

2.4.3 voServer

Communication with NCS is facilitated by the voServer package [14]. This is a minimal publish-subscribe server that provides both binary and ascii data streams. As part of this project a C++ client was developed similar to the NCSTools clients described above. This client is used by all of the IO modules of NCSTools.

2.5 Graphical User Interface

The Graphical User Interface (GUI) is an option given to users for visualizing aspects of the neural model in real-time. The GUI was written as a C++ library using Qt [15]. As with all aspects of NCSTools, it is completely configurable through the input file. The user can specify each tab and what information shows up on that tab. An example GUI is given in Figure 2.

Plot Types

The GUI is used to visualize the state of the NCS simulation. It provides three plot types:

Bar Plots

Bar plot are used to visualize the spiking activity of competing neuron populations. It uses the derived instances of the windowed input containers, described in Section 2.2.

Raster Plots

Raster plots present the precise spiking activity of the neurons within a population. The X axis is the time, in

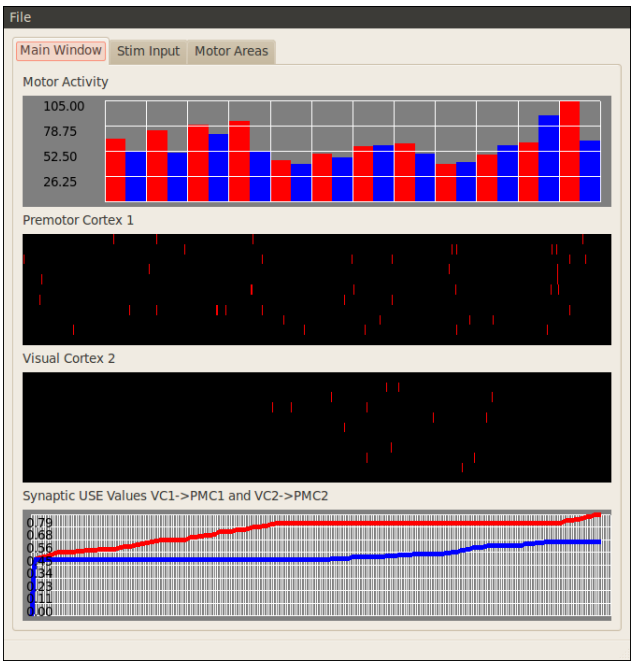

Figure 2: Example graphical user interface. This is a single tab that presents the three main types of plots. The tabs and plots are dynamically created based on the configuration file provided at runtime.

units of simulation reports, and the Y axis corresponds to the neuron index relative to the population.

Line Plots

Line plots are used to plot the synaptic efficacy over time, again this is in units of simulation time.

2.6 Touchpad Interface

The touchpad interface provides users with a way to bind keyboard inputs, either single keys or new-line terminated strings, to stimulus and control signals. The input is entered through the command line and is fully configurable through the input language. There are three major options for the touchpad that are described below. The touchpad signals can be directed to a named NCS input or to agents connected to the NCSTools server. Similar to the inputs and outputs described above, a particular key binding can be used for any number of different commands. With this a single keyboard input to control many different aspects of the neurorobotic interaction.

Instant: When touchpad bindings are defined as instant their associated actions will occur only once and as soon as the command is received.

Repeated: The repeated bindings are analogous to the time outputs described in Section 2.3. These provide a way to repeat an input stimulus over a configured period of time.

Timed: The timed bindings provide a mechanism for setting a period of silence before repeating the configured stimulus. This basically creates a window of stimulus followed by a rest period. This window is then repeated for the configured number of times.

Coupling With Input Signals

The unique aspect of the Repeated and Timed bindings is the ability to couple them to signals coming from the connected agents. When a configured signal is received from a connected agent the user is prompted for a numerical value by the touchpad interface. This value is used to determine the graded input values sent to the simulation. This is used in the example given in Section 3 below.

2.7 Control Interface

In addition to sending stimulus and receiving reports, NCSTools provides access for controlling and modifying a running neural simulation. Commands can be bound to strings from connected agents or to the built-in touchpad functionality. Some of the features this releases include the saving and loading of model states, modification of synapses, adding new stimulus paths and stopping the simulations.

3 Example Scenario

To illustrate NCSTool's role in neurorobotic research, a motivating example is presented in Figure 3. In this case the remote agent is a virtual robotic avatar and the interaction is with a camera and the touchpad interface. The steps are:

1. Camera captures image from user and dominant color is calculated.
2. The virtual environment sends the defined plain text statement ("saw red") to NCSTools through the server interface. This uses a static output described in Section 2.3.
3. Based on the configuration NCSTools will stimulate the appropriate regions of the remote NCS Model through the NCS network interface.
4. The activity of the two premotor regions in the model are monitored and compared as the simulation progresses. A windowed input describe in Section 2.2 monitors the activity.
5. In this case a winner-takes-all calculation is computed and the appropriate plain text statement is sent through the NCSTools server interface to the robotic avatar based on the most active brain region.

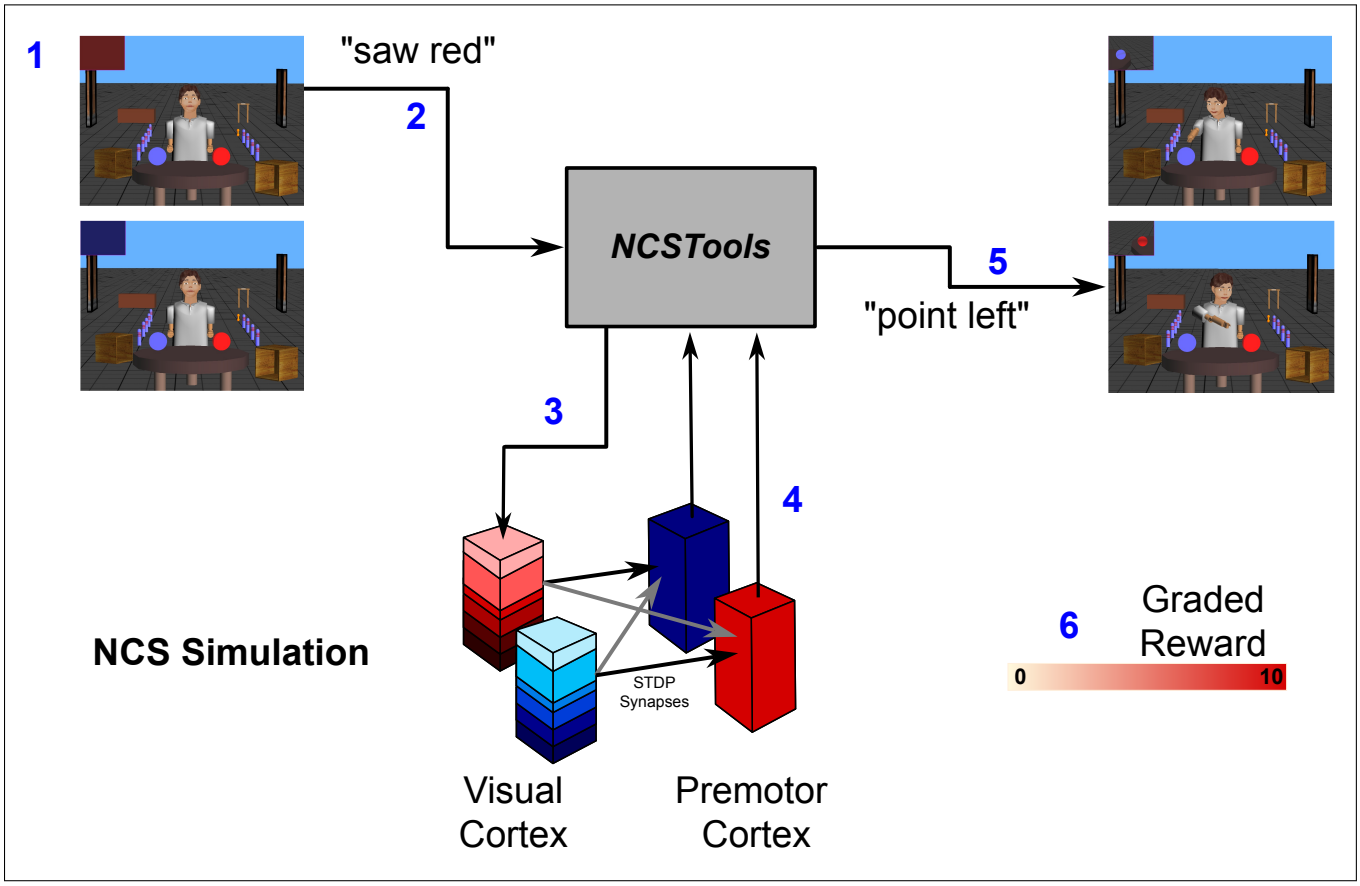

Figure 3: Example neurorobotic scenario.

6. The user is then given the opportunity to "reward" the robot if the correct color was identified. The reward is achieved by coupling the output from the agent with the touchpad interface as described in Section 2.6.

Although this is a simple example there is still a significant amount of coordination involved.

4 Discussion

NCSTools provides a dynamic interface to the neural simulation environment NCS. The abstraction between the neural models and the robotic interface is unique to this project and there have already been several projects that have successfully leveraged NCSTools.

Oxytocin Induced Trust

The work presented by Anumandla *et al.* [16] made extensive use of NCSTools. In this project a human participant interacted with a robotic avatar through a GPU based gabor processing application, NCSTools and a NCS simulation. The results of this work provided new theories on the role Oxytocin may play in establishing and stabilizing trust between mammals. These theories would not have been possible without a closed loop virtual neurorobotic system.

Emotional Speech Processing

Thibeault *et al.* [17] used NCSTools to coordinate the processing between a speech extraction package and a neural simulation. The speech processing algorithm successfully extracted the emotional aspects of a person's speech pattern to determine the reward stimulus to inject into the model.

Virtual Reality

As part of an unpublished proof-of-concept, NCSTools was utilized for the large-scale visualization of a running neural simulation. The neural model was constructed within NCS along with a corresponding X3D model. The visualization software created a virtual representation of the neuron populations. Through the network interface the voltages of the cells within the model were collected and as the simulation progressed the model neurons would change color to represent the voltage of the cell. In addition, 3D sound was used to signal when a spike was fired. The package was tested successfully on a 3D wall and a 6 sided Cave Automatic

Virtual Environment (CAVE).

Future Directions

There are several directions that have been identified for future development of NCSTools.

Real-time Structural Modifications

Providing a mechanism for users to modify aspects of the model including the type of neuron and the connectivity, will greatly increase the rate at which different models can be evaluated. In addition, this would provide a mechanism for actively modifying neurogenesis (the addition of new neurons), and synaptogenesis (the dynamic addition and removal of synaptic connections).

Cluster-Aware Version

The complexity of large-scale neural models generally requires distributed compute clusters. By creating a cluster-aware version of NCSTools, multiple instances can run in parallel while still coordinating communication between instances.

Dynamic Server Configuration

The current version is only modifiable on initialization. Allowing users to modify the configurable aspects of NCSTools will make it a more appealing tool for neurorobotics.

Acknowledgements

This work was supported in part by grants from the U.S. Defense Advanced Research Projects Agency (HR001109C001) and the U.S. Office of Naval Research (N000140110014).

References

[1] M. Scheutz, P. Schermerhorn, J. Kramer, and D. Anderson, "First steps toward natural human-like hri," *Autonomous Robots*, vol. 22, pp. 411–423, 2007. 10.1007/s10514-006-9018-3.

[2] K. Dautenhahn, "Socially intelligent robots: dimensions of humanrobot interaction," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 362, no. 1480, pp. 679–704, 2007.

[3] J. Krichmar, A. Seth, D. Nitz, J. Fleischer, and G. Edelman, "Spatial navigation and causal analysis in a brain-based device modeling cortical-hippocampal interactions," *Neuroinformatics*, vol. 3, pp. 197–221, 2005. 10.1385/NI:3:3:197.

[4] S. I. Wiener and A. Arleo, "Persistent activity in limbic system neurons: neurophysiological and modeling perspectives," *Journal of Physiology-Paris*, vol. 97, no. 4-6, pp. 547 – 555, 2003.

[5] N. Cuperlier, M. Quoy, and P. Gaussier, "Neurobiologically inspired mobile robot navigation and planning," *Frontiers in Neurorobotics*, vol. 1, no. 0, 2007.

[6] C. Thibeault, F. Harris, and P. Goodman, "Breaking the virtual barrier: real-time interactions with spiking neural models," *BMC Neuroscience*, vol. 11, no. Suppl 1, p. P73, 2010.

[7] D. Purves, G. J. Augustine, D. Fitzpatrick, W. C. Hall, A.-S. LaMantia, J. O. McNamara, and L. E. White, *Neuroscience*. Sinauer Associates, inc., 2008.

[8] P. H. Goodman, S. Buntha, Q. Zou, and S.-M. Dascalu, "Virtual neurorobotics (vnr) to accelerate development of plausible neuromorphic brain architectures," *Frontiers in Neurorobotics*, 2007.

[9] P. H. Goodman, Q. Zou, and S.-M. Dascalu, "Framework and implications of virtual neurorobotics," *Frontiers in Neuroscience*, p. 5, 2008.

[10] E. C. Wilson, "Parallel implementation of a large scale biologically realistic neocortical neural network simulator," Master's thesis, University of Nevada, Reno, August 2001.

[11] E. C. Wilson, P. H. Goodman, and F. C. Harris Jr., "Implementation of a biologically realistic parallel neocortical-neural network simulator," pp. 1–11, 2005.

[12] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, Jr., M. Zirpe, T. Natschlager, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: A review of tools and strategies," *Journal of Computational Neuroscience*, pp. 349–398, Nov. 2006.

[13] Q. Peng, "Brainstem: A neocortical simulator interface for robotic studies," Master's thesis, University of Nevada, Reno, December 2006.

[14] J. G. King, "Brain communication server: A dynamic data transferal system for a parallel brain simulator," Master's thesis, University of Nevada, Reno, May 2005.

[15] M. Dalheimer, *Programming with Qt (2nd ed.)*. O'Reilly Media, 2002.

[16] S. Anumandla, L. J. Bray, C. M. Thibeault, R. V. Hoang, S. Dascalu, F. Harris, and P. Goodman, "Modeling oxytocin induced neurorobotic trust and intent recognition in human-robot interaction," in *International Joint Conference on Neural Networks, IJCNN*, July 2011.

[17] C. M. Thibeault, O. Sessions, P. H. Goodman, and F. C. Harris Jr., "Real-time emotional speech processing for neurorobotics applications," in *Proceedings ISCA's 23rd International Conference on Computer Applications in Industry and Engineering (CAINE-2010), Las Vegas, NV.*, November 2010.