frontiers in
NEURAL CIRCUITS

METHODS ARTICLE
published: 18 January 2013
doi: 10.3389/fncir.2012.00098

Closed-loop, multichannel experimentation using the open-source NeuroRighter electrophysiology platform

Jonathan P. Newman[1], Riley Zeller-Townson[1], Ming-Fai Fong[1,2], Sharanya Arcot Desai[1,3,4], Robert E. Gross[3,4,5] and Steve M. Potter[1]*

[1] *Laboratory for Neuroengineering, Department of Biomedical Engineering, Georgia Institute of Technology and Emory University School of Medicine, Atlanta, GA, USA*
[2] *Department of Physiology, Emory University School of Medicine, Atlanta, GA, USA*
[3] *Department of Neurosurgery, Emory University School of Medicine, Atlanta, GA, USA*
[4] *Department of Neurology, Emory University School of Medicine, Atlanta, GA, USA*
[5] *Department of Biomedical Engineering, Georgia Institute of Technology and Emory University School of Medicine, Atlanta, GA, USA*

Edited by:
Eberhard E. Fetz, University of Washington, USA

Reviewed by:
Yang Dan, University of California, Berkeley, USA
Stavros Zanos, University of Washington, USA

****Correspondence:***
Steve M. Potter, Laboratory for Neuroengineering, Department of Biomedical Engineering, The Georgia Institute of Technology, 313 Ferst Drive, Atlanta, GA 30332, USA.
e-mail: steve.potter@bme.gatech.edu

Single neuron feedback control techniques, such as voltage clamp and dynamic clamp, have enabled numerous advances in our understanding of ion channels, electrochemical signaling, and neural dynamics. Although commercially available multichannel recording and stimulation systems are commonly used for studying neural processing at the network level, they provide little native support for real-time feedback. We developed the open-source NeuroRighter multichannel electrophysiology hardware and software platform for closed-loop multichannel control with a focus on accessibility and low cost. NeuroRighter allows 64 channels of stimulation and recording for around US $10,000, along with the ability to integrate with other software and hardware. Here, we present substantial enhancements to the NeuroRighter platform, including a redesigned desktop application, a new stimulation subsystem allowing arbitrary stimulation patterns, low-latency data servers for accessing data streams, and a new application programming interface (API) for creating closed-loop protocols that can be inserted into NeuroRighter as plugin programs. This greatly simplifies the design of sophisticated real-time experiments without sacrificing the power and speed of a compiled programming language. Here we present a detailed description of NeuroRighter as a stand-alone application, its plugin API, and an extensive set of case studies that highlight the system's abilities for conducting closed-loop, multichannel interfacing experiments.

Keywords: closed-loop, multichannel, real-time, multi-electrode, micro-electrode array, electrophysiology, open-source, network

1. INTRODUCTION

Multi-electrode neural interfacing systems, such as planar electrode arrays, silicon probes, and microwire arrays are commonly used to record spatially distributed neural activity *in vitro* and *in vivo*. Advances in nanoscale fabrication techniques have continued to push channel counts and electrode resolution (Du et al., 2011; Fiscella et al., 2012; Robinson et al., 2012), allowing for increasingly detailed measurements of network activity states. Because multi-electrode neural interfaces provide many parallel measurements, they can be used to rapidly estimate ensemble features of network activity (e.g., the population firing rate or network-level synchronization). This makes them well suited for real-time applications.

However, most commercial software interfaces for controlling multichannel hardware lack flexible support for real-time, bi-directional communication with neural tissue. Additionally, commercial software is often hard to integrate into complex multi-component experimental configurations. As a result, multichannel hardware has not been incorporated into closed-loop interfacing schemes to the degree of single-cell recording systems, such as voltage and dynamic clamp (Cole, 1949; Marmont, 1949; Hamill et al., 1981; Prinz et al., 2004; Arsiero et al., 2007; Kispersky et al., 2011). There are some exceptions to this trend (Jackson et al., 2006b; Azin and Guggenmos, 2011; Zanos et al., 2011). These systems are typically limited to low channel counts and/or low recording resolution in order to achieve embedded real-time processing at the recording site using a microcontroller or DSP. This approach has clear advantages for experiments on freely moving animals, but is limited in terms of input and output bandwidth, processing power to enable complex experimental protocols, and ease of programming. Neuroscience research would benefit from a multichannel acquisition platform that (1) enables bi-directional interaction with neuronal networks, (2) is practical for everyday use, (3) is straightforwardly extensible for complex closed-loop protocols, (4) works with a variety multi-electrode interfaces, (5) provides large channel counts and high recording resolution, and (6) is low cost. This type of system would be particularly applicable to three areas of neuroscience research:

- Feedback Control of Network Variables: Neuronal networks are complex systems with many recurrently interacting components. This often results in ambiguity in cause and effect

relationships between network variables (Rich and Wenner, 2007; Turrigiano, 2011). Feedback control can be used to parse variables of neural activation that are causally linked (Cole, 1949). Feedback control of network-level variables (e.g., population firing rate, neuronal synchronization, or neurotransmission levels) can potentially clarify their causal relationships (Wagenaar et al., 2005; Wallach et al., 2011).

- Artificial Embodiment: Dissociated neural cultures, slice preparations, and anesthetized or paralyzed animals allow stable electrophysiological access but cannot engage in natural behaviors with their environment. By artificially embodying reduced neuronal preparations using a virtual environment or a robot, experimental access is maintained while neural tissue is engaged in complex behaviors (Reger et al., 2000; DeMarse et al., 2001; Ahrens et al., 2012).
- Clinical Applications: Responsive (Morrell, 2011) or predictive (Mormann et al., 2007) application of neural therapies have the potential to improve the efficacy and safety of treatments that are currently used in open-loop. Examples include brain stimulation and local drug perfusion techniques that are used to treat movement disorders, clinical depression, chronic pain, and epilepsy. Additionally, electrical stimuli delivered to one region of motor cortex in response to spiking activity in another motor area has been shown to facilitate a functional reorganization of motor output, indicating a potential role for activity-dependent stimulation in rehabilitation therapy (Jackson et al., 2006a).

Here, we present substantial improvements to NeuroRighter, an open-source, multichannel neural interfacing platform which we designed specifically to enable bi-directional, real-time communication with neuronal networks (Rolston et al., 2009a, 2010). In the first half of the paper, we provide a description of NeuroRighter's capabilities, including an application programming interface (API) that facilitates the creation of custom real-time experiment protocols. In the second half of the paper, we demonstrate these features with a variety of case studies. Each case-study highlights a different aspect of NeuroRighter's abilities in the areas of network-level feedback control, artificial embodiment, and closed-loop control of aberrant activity states in freely moving animals.

2. THE NEURORIGHTER MULTICHANNEL ELECTROPHYSIOLOGY PLATFORM

NeuroRighter is an open-source, low-cost multichannel electrophysiology system designed for bi-directional neural interfacing (Rolston et al., 2009a, 2010). A complete system, including all necessary electronics and a host computer, can be assembled for less than $10,000 USD. The NeuroRighter software is free. Extensive documentation on the construction and usage of a NeuroRighter system is available online[1]. NeuroRighter's source code, the API reference, and demonstration closed-loop protocol code, are available from the NeuroRighter code repository[2]. Questions on NeuroRighter assembly and usage can be submitted to the NeuroRighter-Users forum[3]. Tutorials on API usage are provided in sections 1 and 2 of the Supplementary Material.

2.1. HARDWARE

Here we provide a summary of NeuroRighter's hardware building blocks. Hardware components can be used with neural interfaces designed for applications both *in vivo* and *in vitro*. Printed circuit board (PCB) performance specifications are provided in (Rolston et al., 2009a) and layouts are available online. A complete NeuroRighter system meets or exceeds the performance of commercial alternatives in terms of noise levels, stimulation channel count, stimulation recovery times, and flexibility (Rolston et al., 2009a). NeuroRighter's PCBs are designed to be modular: electrode interfacing and stimulation PCBs have identical footprints and use vertical headers to route power between boards. This allows interfacing PCBs to be stacked on top of one another for increased channel counts and the use of a single DC power supply (or set of batteries) for all hardware.

2.1.1. ADC/DAC boards

NeuroRighter uses National Instruments (NI; National Instruments Corp, Austin, TX, USA) data acquisition hardware driven with NI's hardware control library, DAQmx. NI PCI-6259, PCIe-6259, PCIe-6353, and PCIe-6363 16-bit, 1 M sample/sec data acquisition cards are currently supported. Each card supports 32 analog inputs (AI), 4 analog outputs (AO), and 48 I/O-configurable digital channels. NI SCB-68 screw-terminal connector boxes are used to interface each data acquisition card with external hardware. Up to 3 cards can be used in a single NeuroRighter system to meet channel count requirements.

2.1.2. Multichannel amplifier interfacing boards

NeuroRighter provides two types of PCB to interface the NI data acquisition cards with multi-electrode amplifier systems. For *in vivo* applications, a 16-channel filter module provides 1.6X signal buffering, anti-aliasing filtering (−3 dB point at 8.8 KHz), DC offset subtraction (−3 dB point at 1 Hz), and regulated power to the headstage. Up to four of these modules can be stacked together in order to meet channel count requirements. For *in vitro* applications, a 68 channel conversion board provides power and signal routing for planar electrode array amplifier systems, e.g., Multichannel Systems' 60 channel amplifiers (Multichannel Systems, Reutlingen, Germany), which have a manufacturer settable pass-band. Both boards interface with the SCB-68 connector boxes using 34-channel ribbon cables, wired as signal/ground pairs to reduce capacitive crosstalk between adjacent lines during stimulation.

2.1.3. Electrical micro-stimulation hardware

NeuroRighter includes all-channel (up to 64 electrodes) stimulation capabilities for both *in vivo* and *in vitro* systems. This system is based upon the circuits presented in (Wagenaar and Potter, 2004; Wagenaar et al., 2004) and includes two separate PCBs: (1) a voltage- or current-controlled signal generation PCB, and (2) a

[1] https://sites.google.com/site/neurorighter/
[2] http://code.google.com/p/neurorighter/
[3] http://groups.google.com/group/neurorighter-users

signal multiplexing and isolation PCB to select different electrodes for stimulation and isolate recording electrodes from stimulation cables between stimulus pulses.

(1) *Signal generation board.* The signal generation PCB is identical for all applications. This board provides both voltage controlled or constant current stimulation modes. It stacks into the amplifier interfacing board(s) and therefore does not require an additional power source. Aside from stimulus generation, this PCB can be used to perform electrode impedance measurements, which are useful for diagnosing the health of micro-electrodes and their insulated leads, and for electroplating (Desai et al., 2010). Only one signal generation PCB is required for up to 64 electrodes.
(2) *Signal multiplexing boards.* Stimulus multiplexing and isolation occurs at PCBs that piggyback directly on electrode pre-amplifiers. These PCBs are located close to the initial stages of electrode amplification so that the recording amplifier can be isolated from long electrical leads, which reduces capacitive pickup. Because recording amplifiers (e.g., headstages *in vivo* or multichannel amplifiers *in vitro*) come in many shapes and sizes, the design of the multiplexer PCBs is application dependent. For *in vivo* applications, we have designed multiplexer systems that use an 18-pin Omnetics Nano connector, which interfaces with headstages from Triangle Biosystems (Durham, NC), Tucker-Davis Technologies (Alachua, FL), and Neurolinc Corporation (New York, NY), among others (Rolston et al., 2009a). This board employs a single 1-of-16 multiplexer. For *in vitro* applications, four separate multiplexing modules, each of which houses two 1-of-8 multiplexers, plug directly into exposed 0.1″ pitch sockets of a 60 channel Multichannel Systems amplifier (Wagenaar and Potter, 2004). The creation of custom multiplexer boards or adapters for other systems is straightforward due to the simplicity of these PCBs (they generally consist of a single multiplexer integrated circuit).

2.1.4. Generic I/O

NeuroRighter provides 4 analog output channels and 32 bits of programmable digital I/O for controlling or recording digital signals from laboratory equipment. An auxiliary set of up to 32 analog input channels and 32 bits of digital I/O can also be used. Channel counts of generic I/O in a NeuroRighter system depend on the number of data acquisition cards in the user's system, and the amount of analog input channels reserved for the electrodes.

NeuroRighter's hardware serves as an adaptable interface between multi-electrode sensors and data acquisition cards for recording and microstimulation. There are many other options for routing signals to and from the acquisition cards. Therefore, except for the acquisition cards themselves, the hardware we present here is not required to make use of NeuroRighter's software.

2.2. SOFTWARE

The NeuroRighter software application was written in C# (pronounced "C-Sharp"). C# is a modern, general purpose, object-oriented programming language. The software is free and its source code is maintained on a publicly accessible repository[4]. For standard installations, NeuroRighter is distributed as an installation package for 32- or 64-bit Windows operating systems (Microsoft Corp., Redmond, WA). NeuroRighter installations contain two software components:

1. A stand-alone multichannel recording and stimulation application. This includes a graphical user interface (GUI) for data visualization, hardware configuration, data filtering, spike detection and sorting, all-channel stimulation, stimulus artifact rejection, and data recording (section 2.2.1).
2. An application programming interface (API) that allows NeuroRighter to be used as a real-time hardware interface and data server for user-coded protocols (section 2.2.2).

2.2.1. The NeuroRighter application

As a stand-alone application, NeuroRighter can be used for high-quality multichannel recordings (16-bit resolution, 31 k Samples/sec/channel) and all-channel stimulation protocols. NeuroRighter's graphical interface is organized into tabbed pages, each of which encapsulates a particular group of functions or visualization tools (**Figure 1**). In the following section, we discuss the main functional aspects of the stand-alone NeuroRighter application.

2.2.1.1. Main interface. The main NeuroRighter interface (**Figure 1C**) is an access point for all of the application's functionality. It facilitates user manipulation of hardware settings, online filter settings, data visualization windows, stimulation tools, and other features, which are discussed below. Additionally, some recording settings can be manipulated within the main interface itself:

Online acquisition settings. Many filter settings can be adjusted during data collection. This allows the user to fine tune acquisition settings while gaining visual feedback of the effect on incoming data streams. Bandpass, spike detection, and spike sorting parameters can be adjusted during a recording.

Data visualization. Data visualization tools in NeuroRighter use the Microsoft XNA game development framework. This ensures that online visualization does not consume CPU cycles by offloading plotting routines to a supported graphics card. Visualization tools are provided for single-unit activity, local field potentials (LFP), multiunit activity (MUA), electroencephalograph (EEG) traces, and auxiliary analog input streams. Additionally, overlay plots are used to display sorted spike waveforms for each channel (**Figure 1C**).

File saving. Data streams selected by the user are written to disk with a unique file extension that designates their type. These binary files can be read with MATLAB (Mathworks, Natick, MA) functions included with NeuroRighter installations.

[4]http://code.google.com/p/neurorighter/

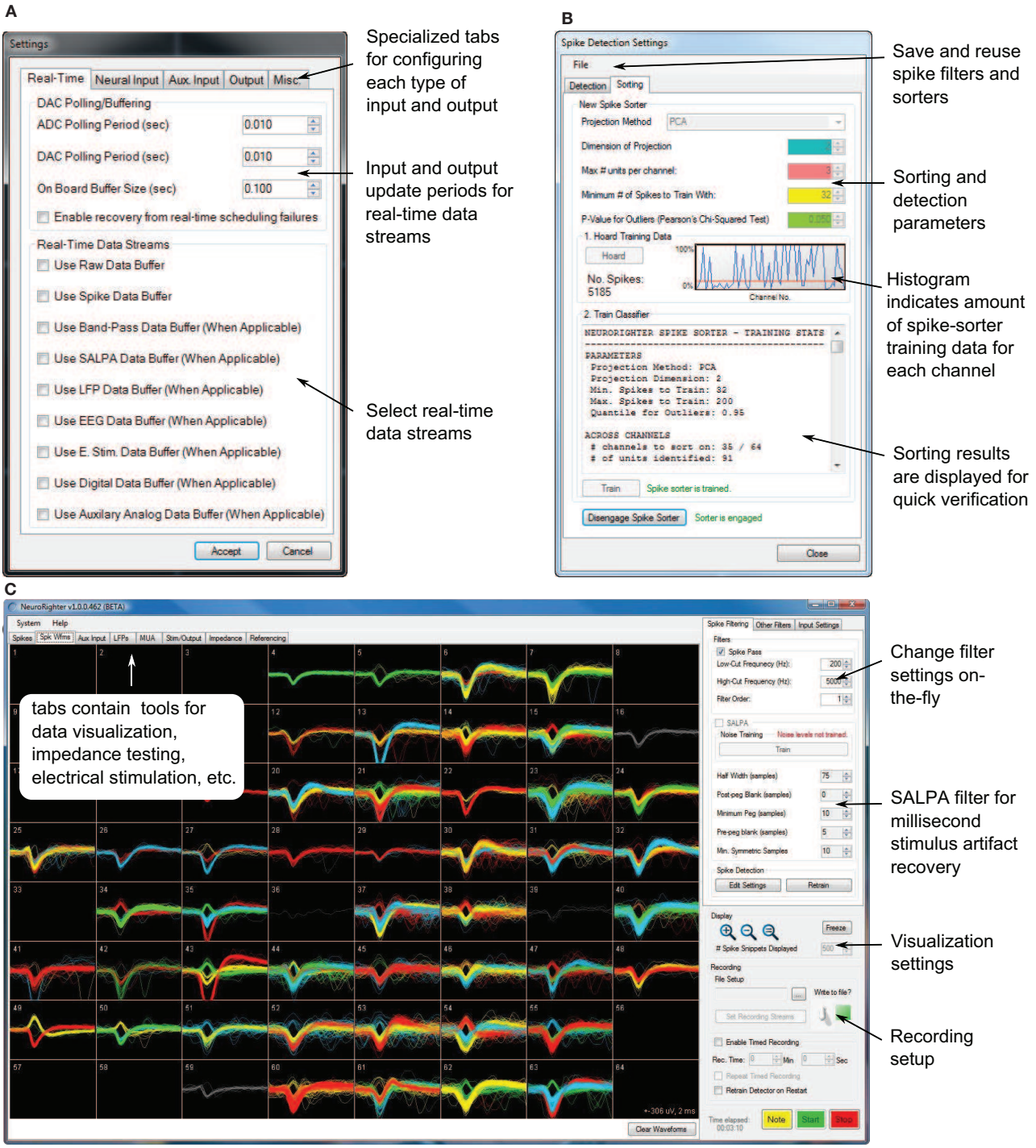

FIGURE 1 | Portions of NeuroRighter's graphical user interface. (A) The hardware settings interface. **(B)** The spike-detection filter and spike sorting interface. **(C)** The main application window. Sorted spike waveforms recorded from a 59-channel, planar electrode array are shown on the spike visualization tab of the main GUI. The position of each waveform corresponds to the position of the recording electrode on which it was detected.

2.2.1.2. ***Hardware configuration.*** Correctly specifying mixed digital and analog signal routing, clock synchronization, and trigger synchronization on a multi-board data acquisition system can be complicated. NeuroRighter simplifies this process using a graphical hardware settings interface (**Figure 1A**). Here, the user specifies the types of signals carried by the NI acquisition cards in his or her system, amplifier gain settings, auxiliary input and output channels, options for electrode impedance measurement,

signal referencing, and real-time data streaming options. Upon closing the settings dialog, NeuroRighter performs the required signal routing and clock synchronization. All NI cards are synchronized to a single clock oscillator using an NI real-time system integration bus (RTSI, **Figure 3**).

2.2.1.3. Time-series filtering. Incoming data from the A/D converters are passed through a cascade of digital filters to produce different neural data streams. First, channel voltages are passed through several linear filters to extract frequency bands for single-unit activity ($\simeq$200–5000 Hz) and LFP ($\simeq$1–500 Hz). MUA, which reflects the firing rate of neurons within the vicinity of the recording electrode, is extracted by rectifying and then low pass filtering the single-unit activity data stream (Supèr and Roelfsema, 2005).

In addition to traditional filtering methods, NeuroRighter provides several specialized filtering options. Common-mode noise sources such as AC mains pickup or movement artifacts in freely moving animals can corrupt neural recordings. NeuroRighter allows the mean or median of all recording electrodes (with appropriate scaling) to be subtracted from individual electrode voltage streams to combat common-mode interference (Rolston et al., 2009b). This is an effective method for reducing non-periodic common-mode interference, such as movement artifacts, where template subtraction methods are inappropriate. Finally, NeuroRighter includes an implementation of the SALPA filter (Wagenaar and Potter, 2002), which subtracts locally fit cubic splines from electrode traces following the application of a stimulus pulse. This removes the capacitive artifacts from non-saturated recording channels and allows online action potential detection within 2 ms after a stimulus pulse.

Sampling rates for different data streams can be set independently. Filter settings (pass-band and filter order) can be modified during data acquisition (**Figure 1C**). Raw data, as well as the result of each filtering stage, yield separate data streams (**Table 1**).

2.2.1.4. Spike filtering. Spike filtering in NeuroRighter is a three-step process: (1) detection, (2) validation, and (3) sorting. NeuroRighter detects spikes using a threshold criterion that compares individual voltage samples to the estimated RMS voltage on the corresponding electrode. Upon threshold crossing, a peak-aligned voltage "snippet" is extracted from the raw voltage stream. Each snippet is validated using a series of *ad hoc* criteria based upon waveform slope, width, and peak-to-peak amplitude. Finally, spikes can be sorted online using an automated Gaussian mixture modeling algorithm. Details of the spike detection and sorting algorithms used by NeuroRighter are provided in section 3 in the Supplementary Material.

The spike detection/sorting configuration is controlled through a child GUI (**Figure 1B**). All relevant spike detection, validation, and sorting parameters are under user control and are manipulated using the spike detection GUI. Because spike-detection settings are changed using a secondary GUI, the effects of parameter changes can be simultaneously monitored on the visualization tabs in the main interface while data collection occurs. A complete list of these parameters is shown in Table S1 in the Supplementary Material. Spike filters, including trained spike sorters, can be saved and reused.

2.2.1.5. Stimulation. NeuroRighter provides several options for delivering complex stimulus patterns to neural tissue either manually through the NeuroRighter application or using scripted protocols. Simple, periodic stimulation protocols, consisting of single or double phase, square, current- or voltage-controlled pulses on any electrode, can be performed directly from the main GUI. Stimuli can be triggered "on demand" in response to a mouse click or by using hardware-timed, periodic sequence of triggers.

Scripted protocols can be used to deliver complex, potentially non-periodic stimulus patterns and to access general purpose analog and digital output lines. Neurorighter uses a double-buffered output engine, called `StimSrv` (**Table 2**), to produce arbitrary, hardware-timed stimulation, analog-output, or digital output signals (**Table 1**, bottom). `StimSrv` can be accessed on-the-fly using NeuroRighter's API (section 2.2.2) or with user-written scripts. The schematic in **Figure 2A** demonstrates how `StimSrv` delivers uninterrupted output. First, a block of the NI cards' memory is reserved and divided into two sections, each of which comprises a single output buffer. At a given instant, one buffer is reserved for sample generation and one is available for writing. When the all samples in the read buffer are exhausted, the buffers switch roles, allowing seamless delivery of constantly varying output signals. This allows the delivery of complex, aperiodic stimulation patterns and the orchestration of experimental apparatuses using analog and digital output lines. All output is clock-synchronized to input data streams, allowing *a priori* specification of stimulus delivery times, relative to the start of the experiment, with single-sample precision. Stimulation scripts can be created with a set of MATLAB functions that are included with NeuroRighter installations (see section 1 in the Supplementary Material).

Figure 2B demonstrates the use of a scripted stimulation protocol to deliver spatio-temporal patterns of electrical stimuli. One-second trials of spatially uniform, and temporally Poisson random stimulus pulses were delivered to a dissociated cortical network. Each trial consisted of either a new, random stimulus realization or a single repeated realization. Each type of stimulus sequence was interleaved with no delay between adjacent trials. **Figure 2Bi** shows stimulus raster plots for 100 trials each stimulus type, with a gray-scale indicating the stimulus trial. For repeated stimuli, individual trials cannot be seen since the recording and stimulation subsystems are clock-synchronized and every repeated stimulus sequence occupies the same set of samples relative to the start of a trial. **Figure 2Bii** shows spiking patterns in response to random and repeated stimuli for 4 units across trials. The delivery of repeated stimuli to the network results in extremely reproducible spiking patterns, and non-repeated, random stimuli probe the variability of population spiking response. This type of stimulus protocol is commonly used to estimate the mutual information between a stimulation process and the population spiking response (Strong et al., 1998; Yu et al., 2010).

2.2.2. NeuroRighter's application programming interface

NeuroRighter installations include an API that facilitates the creation of real-time protocols. The API comprises a set of tools for interacting with NeuroRighter's input and output streams. Protocols written using the API are externally compiled libraries that can "plug in" to the NeuroRighter application in order to impart

Table 1 | Overview of NeuroRighter's input and output streams.

Input	Source	Server (DataSrv)	Buffer type	Max. channel count
	Raw electrodes	RawElectrodeSrv	Circular double[][]	64
	SALPA Filter	SalpaSrv	Circular double[][]	64
	Spike-band filter	SpikeBandSrv	Circular double[][]	64
	Spike filter	SpikeSrv	List <SpikeEvent>	64 or No. units
	LFP filter	LFPSrv	Circular double[][]	64
	EEG filter	EEGSrv	Circular double[][]	64
	MUA filter	MUASrv	Circular double[][]	64
	Electrical stimuli	ElecStimuliSrv	List <SpikeEvent>	64
	Auxiliary analog	AuxAnalogSrv	Circular double[][]	32
	Auxiliary digital	AuxDigitalSrv	List <DigitalEvent>	32 bits
Output	**Source**	**Server (StimSrv)**	**Buffer type**	**Max. channel count**
	Electrical stimuli	StimOut	List <StimulusEvent>	64
	Analog output	AnalogOut	List <AnalogEvent>	4
	Digital output	DigitalOut	List <DigitalEvent>	32 bits

Each stream is accessed using a dedicated server that includes functions for reading from, or writing to, its data buffer.

Table 2 | Packages included with NeuroRighter's Plugin API.

Package	Component	Description
Server	DataSrv	Contains input server objects (**Table 1**, top)
	StimSrv	Contains output server objects (**Table 1**, bottom)
Datatypes	MultiChannelBuffer	Circular buffer for time series data
	SpikeEvent	Spike event type (time, channel, waveform, unit)
	DigitalEvent	Digital event type (time, 32-bit port state)
	StimulusEvent	Stimulus event type (time, channel, waveform)
	AuxEvent	Auxiliary voltage event (time, channel, voltage)
NeuroRighterTask	NRTask	Abstract class for real-time NeuroRighter interfacing
Log	Logger	Used for debugging real-time protocols

real-time and closed-loop functionality. The software packages included with the API are shown in **Table 2**. Each package contains different set of tools for interacting with NeuroRighter's data streams. Here we discuss the contents and usage of each of these tools. Additionally, a detailed API reference is available online[5].

2.2.2.1. NeuroRighterTask. User-defined protocols employ the NeuroRighter application as a real-time data server. These protocols are inherited from a base component called `NRTask`, which belongs to the NeuroRighterTask package. Closed-loop protocols created with the plugin API are derived from `NRTask` (see section 2 in the Supplementary Material for details). Three functions included in `NRTask` can then be accessed to impart real-time functionality.

1. `NRTask.Setup()`: This function is called when the base `NRTask` component is instantiated. It allows one-time setup operations to take place, such as the declaration of variables, allocation of internal buffers, file streaming setup, GUI initialization, etc.
2. `NRTask.Loop()`: This function is executed periodically by a hardware-timed clock. Execution periods of 1 to 150 ms are allowed and can be set from the Hardware Settings GUI in the main application (**Figure 1A**). To achieve closed-loop functionality, code within the `Loop` function should access other components of the API, most importantly components from the Server and DataTypes packages (**Table 2**). These packages provide access to incoming neural data streams and output buffers and can be used to form a bi-directional interface with neural tissue. Output can be sent from within the `Loop` function using the StimSrv package (**Table 2**) or through natively supported communication interfaces such as TCP/IP ports, serial ports, or USB communication.
3. `NRTask.Cleanup()`: This function is called a single time when the protocol is stopped from the NeuroRighter GUI. It allows the deconstruction of GUIs, the closure of file streams that may have been created during the execution of the plugin, and other cleanup routines.

[5]https://potterlab.gatech.edu/main/neurorighter-api-ref/

A

```
/// StimSrv-based plugin
using NeuroRighter.NeuroRighterTask;
using NeuroRighter.DataTypes;
using NeuroRighter.DatSrv;
using NeuroRighter.StimSrv;
namespace Example
{
  public class MyTask : NRTask
  {
 // Called once at plugin start
    protected override void Setup(){
    }
    // Called by output buffer
    protected override void Loop(){
      data = NRDataSrv.SpikeSrv.Read();
      if (myUnit member of data)
      {
        NRStimSrv.Write();
      }
    }
    // Called upon plugin termination
    protected override void Cleanup(){
    }
  }
}
```

B

```
/// NewData-based plugin
using NeuroRighter.NeuroRighterTask;
using NeuroRighter.DataTypes;
using NeuroRighter.DatSrv;

namespace Example
{
  public class MyTask : NRTask
  {
    // Called once at plugin start
    protected override void Setup(){
      SpikeSrv.NewData += NewData_Hander();
    }
    // Called on NewData event
    private void NewData_Hander(){
      if(myUnit member of data)
      {
        NI Card sends output;
      }
    }
    // Called upon plugin termination
    protected override void Cleanup(){
    }
  }
}
```

LISTING 1 | Code structure for two types of real-time plugin implemented with the API. (A) Pseudocode for a StimSrv-based real-time plugin. **(B)** Pseudocode for real-time plugin triggered by NewData events.

Listing 1A and **1B** provide pseudocode for a two real-time plugins that both respond to a spike produced by a particular detected unit. A real-time protocol written using the API will follow the structure of one of these code skeletons, regardless of its complexity. First, the user references the required packages from the API. Next, the plugin is designated to be a child of `NRTask`, which provides the protocol with automatic access to NeuroRighter's data servers. Finally, the `Setup()`, `Loop()`, and `Cleanup()` functions are overridden (**Listing 1A**), or a `NewData` event is subscribed to (**Listing 1B**), to impart real-time functionality. After it is compiled (either using Visual Studio or Mono[6]), the plugin can be executed through NeuroRighter's GUI. Plugin protocols executed through NeuroRighter operate on a high-priority thread to decrease closed-loop response latency. The diagram shown in **Figure 3** shows the interaction between a plugin created using the API, the NeuroRighter executable, and hardware. Functional examples of plugin protocols are provided in section 5 of the Supplementary Material.

2.2.2.2. ***Server.*** Components derived from `NRTask` have automatic access to NeuroRighter's input and output servers, which belong to the Server package. There are two banks of data servers: (1) `DataSrv`, which can be used to read NeuroRighter's input streams (**Table 1**, top) and (2) `StimSrv`, which can be used to write to output streams (**Table 1**, bottom). `DataSrv` and `StimSrv` objects encapsulate isolated data servers, each of which handles a particular data stream. Each server includes methods for reading the hardware clock, reading from and writing to its own data buffer, and accessing stream metadata. Because input and output servers are simultaneously accessible from within a user-defined `NRTask`, sending output signals (e.g., stimuli) contingent on recorded input is straightforward. The user can select which data streams are sent to `DataSrv` or available for writing on `StimSrv` using the Hardware Settings GUI (**Figure 1A**).

[6]http://www.mono-project.com/Main_Page

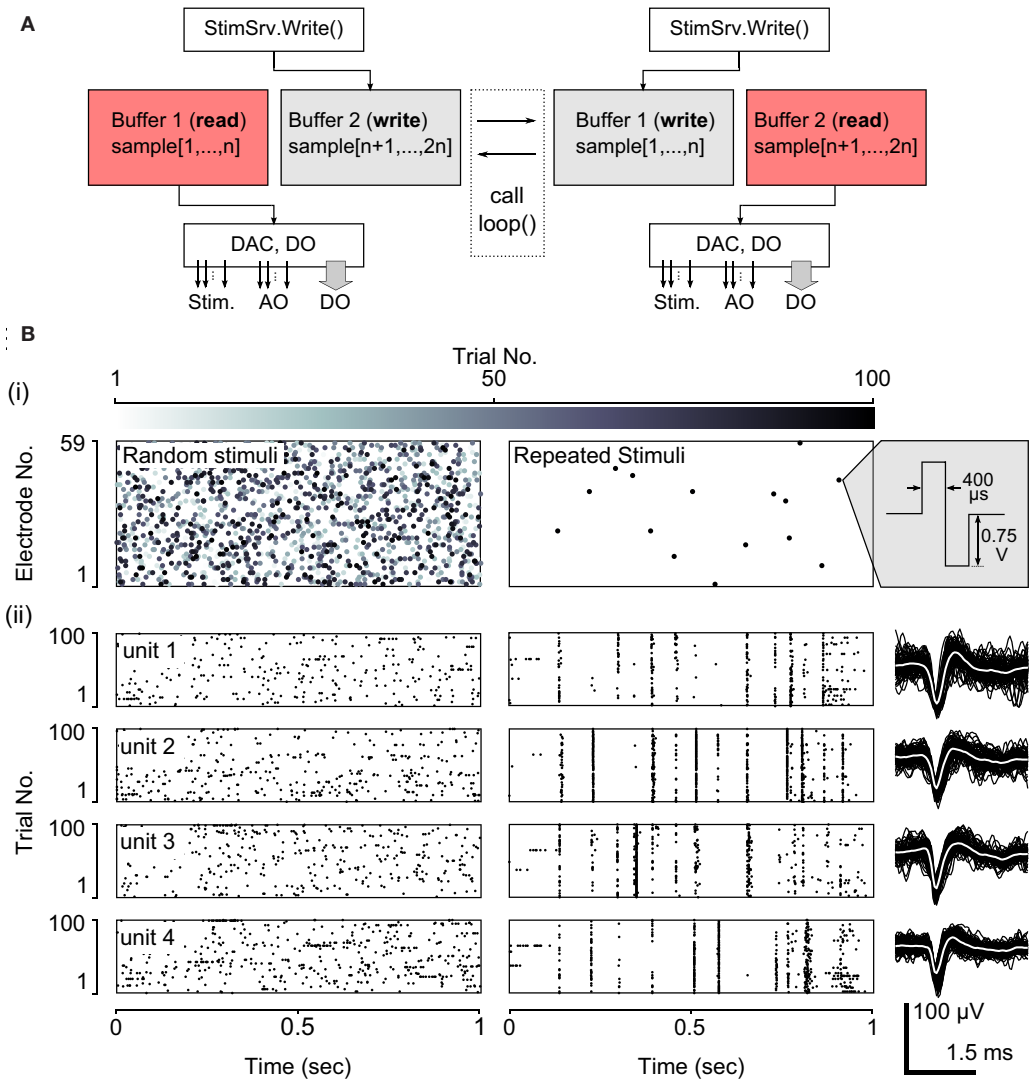

FIGURE 2 | NeuroRighter's StimSrv subsystem. (A) To deliver complex, non-periodic stimuli, NeuroRighter uses a double-buffering system. This allows samples to be generated and written to the NI cards' analog and digital outputs simultaneously. At a given instant, one buffer is reserved for reading (pink) and one from writing (gray). When the all samples in the read buffer are generated, the buffers switch roles, allowing seamless delivery of constantly varying stimulus patterns and generic analog and digital signals. When using `StimSrv` for closed-loop protocols, the `loop()` function is called at the instant of a buffer switch. **(B)** Example open-loop stimulus protocol using `StimSrv`. (i) 100, 1 s Poisson sequences of electrical stimuli (left) and a single repeated Poisson sequence (right), were delivered to a dissociated cortical network (biphasic, voltage controlled, ±0.75V, 800 μs period). Stimulus rasters are shown using a gray-scale to indicate the trial number. For repeated stimuli, stimulus points are overlaid since stimulus delivery is clock-synchronized with the acquisition subsystem. (ii) Rastergrams of 4 units are shown below each stimulus raster, across trials. Example waveforms for each of the 4 units are shown to the right.

A final important feature of each data server within `DataSrv` is a `NewData` event. A `NewData` event is fired for a given stream each time it receives new data for the A/D card or a digital filter. Functions within a plugin can subscribe to these events so that feedback processing only occurs when new data is acquired. This reduces computational overhead and the latency of the closed-loop response. Plugins that use `NewData` events to generate feedback are not required to include a `Loop()` function or to use `StimSrv` to send output signals. Instead, standard calls to the National Instrument driver library (DAQmx) can be used to access the NI cards' directly. Alternatively, output can be generated using natively supported external communication protocols (USB, TCP/IP, UDP, serial, etc.). **Listing II. B.2(b)** provides pseudocode for a real-time protocol analogous to **Listing II.B.2(a)**, but using the `NewData` event to trigger a response. This type of plugin provides a lower response latencies but is less capable of producing complex, precisely timed output signals. A functional example of a `NewData`-based plugin is provided in section 5.2 in the Supplementary Material.

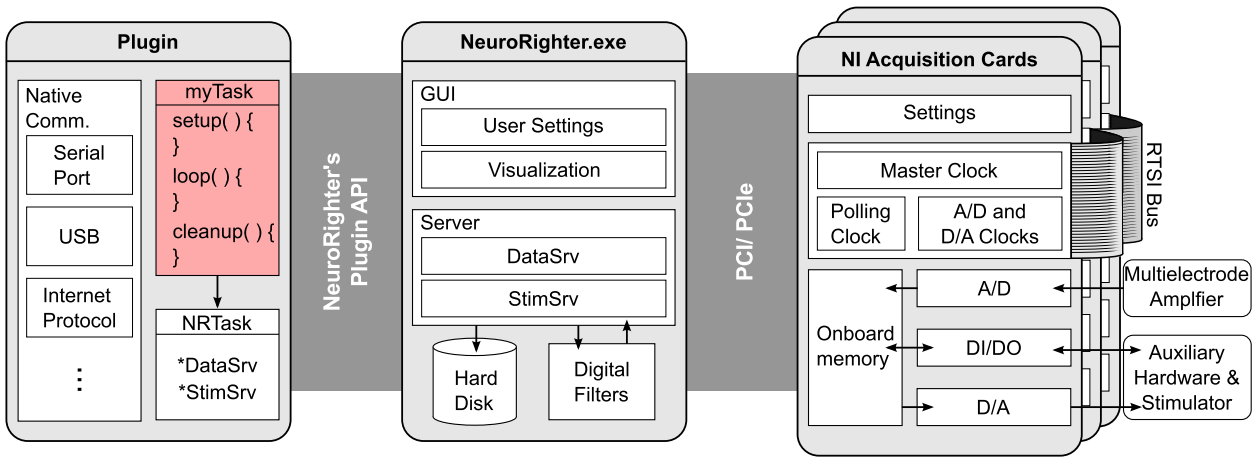

FIGURE 3 | Conceptual schematic of NeuroRighter's hardware and software elements. NeuroRighter serves as a high-level interface between hardware and custom user-written protocols (pink box). NeuroRighter simplifies hardware level programming by using datatypes and methods that are specialized for multichannel neural recording and stimulation. This facilitates the creation of low-latency, closed-loop protocols. Neural signals and secondary data streams are fed into the NI cards' analog and digital inputs where they are digitalized and stored temporarily in on-board memory. NeuroRighter periodically transfers data from the acquisition cards' FIFO memory to RAM using direct memory access. Data is then pushed to NeuroRighter's `DataSrv` server object. `DataSrv` serves data to NeuroRighter's visualization tools, filtering algorithms, and externally compiled plugins. The plugin API provides functions for safe interaction with `DataSrv` so that custom operations can be performed on incoming data streams. User-written plugins can interact with any of the computer's native communication ports, or write data back to `StimSrv` in order to control external hardware as a function of recorded neural signals.

2.2.2.3. Datatypes. NeuroRighter's input and output servers operate on high-level data types that encapsulate different forms of multichannel input and output data. These include multichannel buffers for continuous data streams (such as raw electrode voltages or LFP recordings) and discrete event types (such a detected spikes or stimulation events). Extensive documentation on each of these data types is provided in the API reference.

2.2.2.4. Log. The Log package provides accesses to a data logging tool that operates within the NeuroRighter executable, but can be invoked from a user protocol. This tool can be used to write information to a log file using a separate, low-priority thread. This is useful in the development of real-time protocols because core NeuroRighter operations (such as the timing of hardware reads, writes, and other triggers) are logged to this file as well, providing context for messages written from the plugin.

3. CASE STUDIES

NeuroRighter's abilities for orchestrating closed-loop experiments are best demonstrated through example. Here we present five case studies in which protocols created with the API were used to measure NeuroRighter's closed-loop reaction-time, clamp network firing levels in dissociated cultured cortical networks, react to seizures in freely moving animals with multi-electrode electrical stimulation, and control robots serving as artificial embodiments. Experimental methods, and plugin examples are provided in the section 4 in the Supplementary Material. The plugin code used in these case studies is available for download on NeuroRighter's code repository. [7]. Additionally, we provide all code used in the reaction-time case study in section 5 in the Supplementary Material.

3.1. LOW-LATENCY CONTROL OF REAL-TIME HARDWARE

Rapid response times are critical for maintaining a tight feedback loop in which features of incoming data streams (e.g., spikes, EEG, temperature, or animal motion) are used to trigger or adjust the delivery of stimuli. To benchmark the response speed of protocols written using the API, we wrote a protocol that generated output signals in response to recorded action potentials. We picked two sorted units from a dissociated neural culture to serve as triggers for hardware activation. When one of these units fired, it triggered the output of a digital word encoding the identity of the detect unit. These signals serve as a generic stand-in for a stimulation pattern or any other hardware control signal that might be used in a feedback control scheme. Output signals were then recorded using NeuroRighter's digital input port. The delay between action potential detection and signal generation could then be measured using the same sample clock. A diagram of the experimental protocol is shown in **Figure 4A**. We wrote protocols to test three hardware options for generating the required digital output:

1. StimSrv: Buffered manipulation of the NI cards using NeuroRighter's native stimulation server (**Figure 2** and **Listing II.B.2(a)**).
2. NewData: Unbuffered manipulation of the NI cards whenever new data enters NeuroRighter's spike server (**Listing II.B.2(b)**).
3. Arduino: An Arduino ATmega2560-based microcontroller board[8] communicating via serial port (RS-232).

[7] http://code.google.com/p/neurorighter/source/browse/NR-ClosedLoop-Examples/

[8] http://www.arduino.cc/

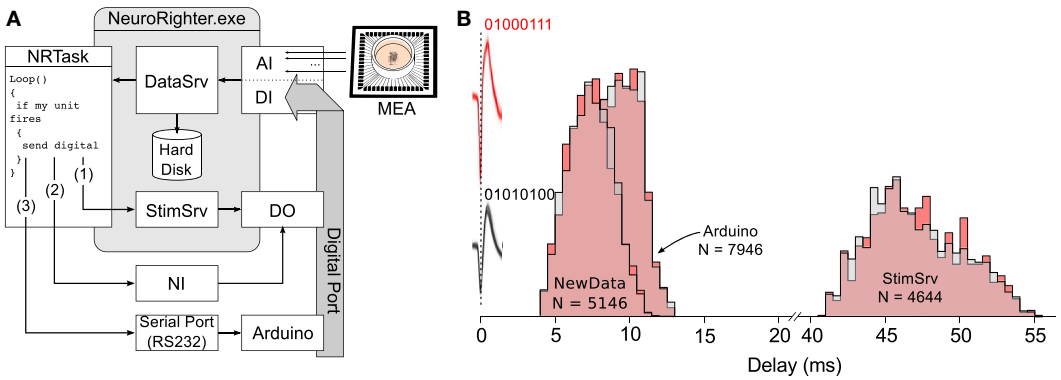

FIGURE 4 | Estimated loop times for bi-directional communication using different hardware configurations. (A) Schematic of experiment used to test reaction delays for different real-time hardware options. Spikes detected and sorted from 59-channel planar electrode array were passed to the real-time plugin. The plugin determined if a spike originated from one of two units of interest. In the case that a spike was produced by one of the two units, the plugin triggered the generation of a digital word encoding the detected unit using either StimSrv, unbuffered digital output triggered by a `NewData` event, or an Arduino board. Digital signals were then, recorded though NeuroRighter's digital input port. **(B)** Normalized histogram of time delays from spikes produced by the two units of interest (action potential waveforms are shown in pink and gray and occur at 0 ms) to the recorded digital signals produced by the plugin to encode the units (01000111 or 01010100). Delay histograms are shown for each unit (pink and gray) and the three different hardware options. N is the number of spikes recorded for each hardware option.

The response latency, calculated from the time of an action potential peak to the corresponding change in the digital port was calculated for each hardware option (**Figure 4**). Mean response latencies were 46.9 ± 3.1 ms for rb StimSrv, 7.1 ± 1.5 ms for `NewData`, and 9.2 ± 1.3 ms for the Arduino board. Latencies where measured while NeuroRighter performed bandpass filtering, spike detection, spike sorting, data streaming, and data saving for 64 electrode inputs, each sampled at 25 kHz. Experiments were conducted on a desktop computer using an Intel Core i7 processor (Santa Clara, California, USA.) and running running 64-bit Windows Vista.

The differences in reaction latency for different hardware options are a result of both the method used to communicate with the hardware and the how the input sent from NeuroRighter is interpreted and transformed into a physical output signal. The differences in response times for NewData and Arduino are largely attributable to the different communication protocols and command interpretation by the client device. For instance the Arduino used a RS-232 serial interface where as NewData communicates with the NI cards via PCIe. StimSrv's long latency in comparison to other options is a result of its double buffering system, which requires a relatively long time period between updates to the NI D/A's output buffer. While StimSrv is slow in comparison to the NewData and microcontroller options, it provides an interface that is easier to use and allows the uninterrupted delivery of arbitrary complex singal outputs. On the other hand, the Arduino and NewData methods can only respond by generating finite-sample or periodic control signals. We have found that StimSrv is fast enough for most of our closed-loop requirements. For this reason, we used StimSrv to generate physical outputs for the remainder of the case studies. However, as demonstrated above, the API's modularity allows the use of faster hardware options with little change in coding complexity.

3.2. MULTICHANNEL POPULATION FIRING CLAMP

The population firing rate is a building block of the neural code. The ability to precisely control population firing in the face of experimental perturbations can be used to understand its role in network function. To demonstrate NeuroRighter's ability to control the network firing rate, we implemented the feedback controller presented in Wagenaar et al. (2005) to control the firing activity in dissociated cortical cultures grown on 59-channel micro-electrode arrays. This algorithm adjusts the stimulation amplitude of voltage controlled, biphasic pulses on 10 electrodes to desynchronize population firing and force the network firing rate to track target values. The control law is given by

$$v_k[t + \Delta T] = v_k[t] - \alpha v_k[t] \left(\frac{\langle f_u[t] \rangle}{f^*} - 1 \right), \qquad (1)$$

where v_k is the stimulation voltage on electrode k, $\langle f_u[t] \rangle$ is the average firing rate across sorted units detected with the 59 electrode array extending over a 2 s window into the past, f^* is the target firing rate, ΔT is the update period of the feedback loop (as defined within NeuroRighter's Hardware Settings GUI), and α defines the time constant of the feedback controller as

$$\tau_{FB} = \Delta T / \alpha. \qquad (2)$$

We used $\Delta T = 10$ ms and $\alpha = 0.002$ so that $\tau_{FB} = 5$ s. Electrodes were stimulated at a 10 Hz aggregate frequency (1 Hz per electrode for 10 electrodes) in a random, repeating sequence. Additionally, individual electrode voltages were multiplied by a tuning factor that was inversely proportional to the number of spikes that occurred within 30 ms following a stimulus pulse on that electrode, as described in Wagenaar et al. (2005). This factor equalizes each electrode's ability to evoke a spiking response, and is critical for achieving the desynchronizing effect of the controller on population activity.

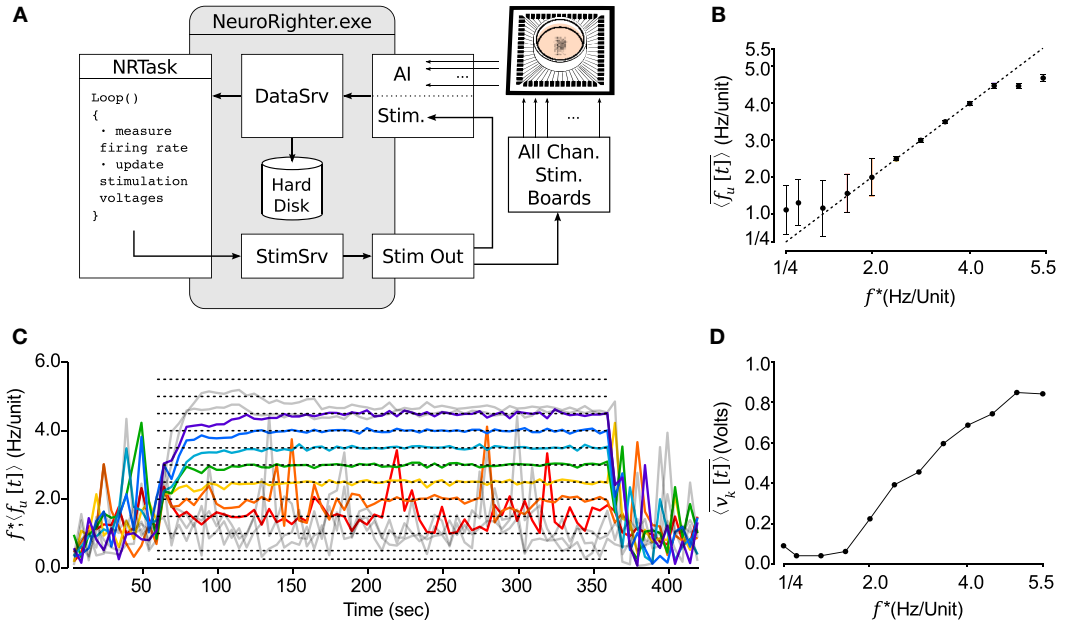

FIGURE 5 | NeuroRighter can be used to clamp population firing rates *in vitro* using closed-loop electrical stimulation. (A) Schematic of the multi-electrode population firing clamp. **(B)** Step tracking performance is shown for a range of target firing rates, f^* (dotted lines). The average neuronal firing rate across detected units, $\langle f_u[t]\rangle$ (colored lines), is shown for each step in f^*. Tracking failures are colored gray. **(C)** Time averaged neuronal firing rate for the last 2.5 min of each 5 min protocol compared to the reference signal, f^*. The dotted line is identity. **(D)** The mean control voltage across the stimulating electrodes over the final 2.5 min of each step protocol at different values of f^*.

We used the controller to clamp network firing at target rates for 5 min epochs. These results are shown in **Figure 5**. The controller was able to achieve target rates within the range of $f^* = 1.5$–4.5 Hz/Unit. An animation of neural activity before and during firing-rate clamping is provided in the Supplementary Material.

The monotonically increasing relationship between the mean stimulation voltage $\overline{\langle v_k[t]\rangle}$, and target firing rate f^* (**Figure 5D**) might indicate that knowledge of the stimulation voltage versus firing rate relationship is sufficient to design an open-loop controller capable of holding network firing rates. To test this, we clamped firing at $f^* = 3.0$ Hz/Unit over 10 min epochs for 15 trials. Five minutes into each 10 min protocol, we stopped updating stimulation voltages on the ten stimulating electrodes, but continued multi-electrode stimulation in open-loop mode (**Figure 6**). Although the desired mean firing rate was achieved fairly consistently, the open-loop control scheme could not react to the rapid changes in excitability that are typical of cultured cortical networks (Wagenaar et al., 2006b). This variability is reflected in the large range of control signals required to track the target rate over the first 5 min of each trial. As a result the RMS error of $\langle f_u[t]\rangle$ about f^* increased by a factor of 5.1 for open-loop compared to closed-loop epochs. The variance of firing during open-loop stimulation is comparable to that of spontaneous (non-evoked) firing behavior that was recorded before the controller was switched on (**Figure 6**, top).

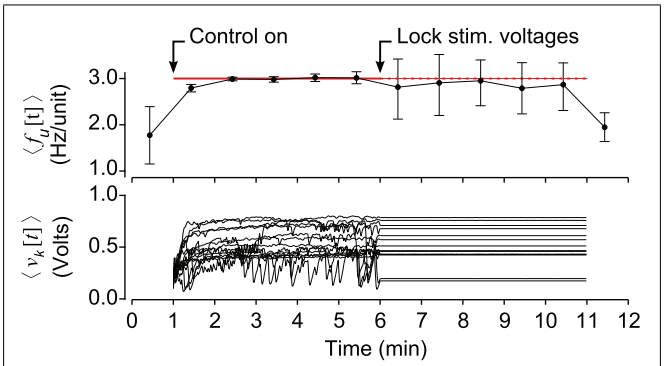

FIGURE 6 | Closed-loop stimulation is required to robustly clamp population firing. (Top) The average neuronal firing rate over 1 min periods across 15 trials. Half-way through a multichannel population clamp protocol, real-time voltage updates stop and microstimulation is applied in open-loop. Error bars are ± standard deviation. (Bottom) The mean electrode stimulation voltage across 10 stimulating electrodes, for each of the 15 trials.

3.3. LONG-TERM POPULATION FIRING CLAMP WITH SYNAPTIC DECOUPLING

3.3.1. Experiment 1

In vitro neural preparations allow continuous experimental access to neural tissue over very long time scales (Potter and DeMarse,

2001), and therefore serve as important models for understanding slowly occurring developmental processes (Turrigiano et al., 1998; Minerbi et al., 2009; Gal et al., 2010). To demonstrate that NeuroRighter is capable of stable closed-loop neural interfacing over long time scales, we used the multi-electrode feedback controller used in section 3.2 for 6 h epochs. This protocol started with a 1 h recording of spontaneous activity. Then, the controller was engaged to clamp population firing to $f^* = 3.0$ Hz/Unit for 6 h. Following the clamping protocol, spontaneous network activity was recorded for an additional hour.

Figure 7A shows the resulting multichannel stimulation signal (**Figure 7Ai**), neuronal firing rate in relation to f^* (**Figure 7Aii**), individual unit firing rates (**Figure 7Aiii**), and zoomed rastergrams before, during, and after multi-electrode stimulation was applied (**Figure 7Aiv**). The controller achieved the $f^* = 3.0$ Hz/Unit tracking over the duration of the 6 h protocol. Additionally, network activity was desynchronized through most of the control epoch, but occasionally the controller allowed bouts of synchronized network activity (Wagenaar et al., 2006b).

3.3.2. Experiment 2

Spiking and neurotransmission have a strong reciprocal influence on one another, making their individual effects on network development difficult to quantify (Turrigiano, 2011). For instance, *N*-methyl-D-aspartate (NMDA)-ergic neurotransmission plays a large role in sustained network recruitment (Nakanishi and Kukita, 1998). For this reason, long-term changes in the state of *in vitro* networks following the application synaptic blockers (e.g., changes in firing rate, spiking patterns, or synaptic-strength) is difficult to attribute directly to effects on neurotransmission because of secondary, confounding effects on network activity levels. However, the closed-loop population clamp provides a solution to this problem. A firing rate controller has the potential to compensate for changes in network excitability induced by the application of a drug, removing its confounding effect on network activity.

To test this, we used the multichannel population clamp during the bath application of d(−)-2-amino-5-phosphonopentanoic acid (AP5), a competitive antagonist of NMDA receptor. This protocol proceeded identically to experiment 1 except that at 1-h following the start of closed-loop stimulation, NeuroRighter triggered the perfusion of 50 μm AP5 into the culturing medium using a syringe pump and a custom, gas-permeable perfusion lid (Potter and DeMarse, 2001; Figure S5 in the Supplementary Material). Four hours after AP5 was applied, NeuroRighter triggered the pump a second time to perform a series of washes with normal culturing medium that removed AP5 from the bath.

Time-series results of this protocol are shown in **Figure 7B**. The contents of these plots are analogous to **Figure 7A** but have arrows to indicate when AP5 was added to, and removed from, the culturing chamber. The controller was able to successfully compensate for changes in network excitability caused by the addition of AP5. Changes in network dynamics were reflected in the control signal, which became smoother in the presence of the AP5 (**Figure 7Bi**).

3.3.3. Comparing Experiments 1 and 2

Figure 7C shows the average, pair-wise firing rate correlation functions (Tchumatchenko et al., 2010) for 30 randomly selected units from experiment 1 (black lines) and experiment 2 (red lines). **Figures 7Ci,iii** show the correlation functions of spontaneous network activity before and after the controller was engaged, respectively. **Figure 7Cii** shows correlation functions for epochs during the clamping phase (which included the AP5 treatment for experiment 2). The periodicity of this correlation function follows the 10 Hz aggregate stimulation frequency during the clamping period.

Intriguingly, although the pair-wise spiking correlations for experiments 1 and 2 were very similar for epochs of spontaneous activity before and during multichannel stimulation (**Figures 7Ci,ii**), they were remarkably different once the stimulator was turned off (**Figure 7Ciii**). When AP5 was not present during the clamping phase (experiment 1), the firing correlation between units appeared to be enhanced following multichannel stimulation. In contrast, pair-wise correlations were almost non-existent following the a population clamp in which AP5 was present (experiment 2). Because the firing statistics (firing rate and correlation structure) during the 6-h clamping period were nearly identical for the both experiments 1 and 2, this effect on the correlation structure of network activity can not be due to effects on firing activity, but required blocking NMDAergic transmission. Without the closed-loop controller in place, AP5 would have affected network activity levels, obfuscating the mechanism of AP5's effect.

This case study demonstrates the ability of the closed-loop controller to quickly adapt to drug-induced changes in network excitability, to decouple network variables that are normally causally intertwined, and to operate robustly over many hours. Additionally, this case study demonstrates NeuroRighter's ability control peripheral equipment aside from electrical stimulators.

3.4. REAL-TIME SEIZURE INTERVENTION IN FREELY MOVING RATS

Aside from *in vitro* recording hardware, NeuroRighter can interface with many different types of neural probes, including those designed to record from and stimulate freely moving animals. To demonstrate this, we performed electrical micro-stimulation in response to paroxysmal activity of hippocampal recordings taken from a rat with induced temporal lobe epilepsy. Many studies have shown potentially therapeutic effects of electrical stimulation on epileptic brain tissue, which could serve as an alternative to pharmacological or surgical treatment methods. For instance, electrical stimulation triggered by characteristic field potential abnormalities can potentially abrogate seizures and lead to a decreased frequency of behavioral symptoms (Mormann et al., 2007; Morrell, 2011; Nelson et al., 2011).

We used the plugin API to create a closed-loop protocol that could detect temporal lobe seizures in freely moving rats and react with multi-electrode stimulation (**Figure 8A**). This control scheme is similar to that of the NeuroPace responsive neurostimulation system (Sun et al., 2008) (NeuroPace Inc., Moutain View, CA, USA), with the exception that we used multi-micro-electrode stimulation instead of driving a single macroelectrode.

Rats were rendered epileptic using focal injections of tetanus toxin into the right-dorsal hippocampus (Hawkins and Mellanby, 1987; see section 4C in the Supplementary Material). LFPs were recorded from CA1 and CA3 regions of the hippocampus using

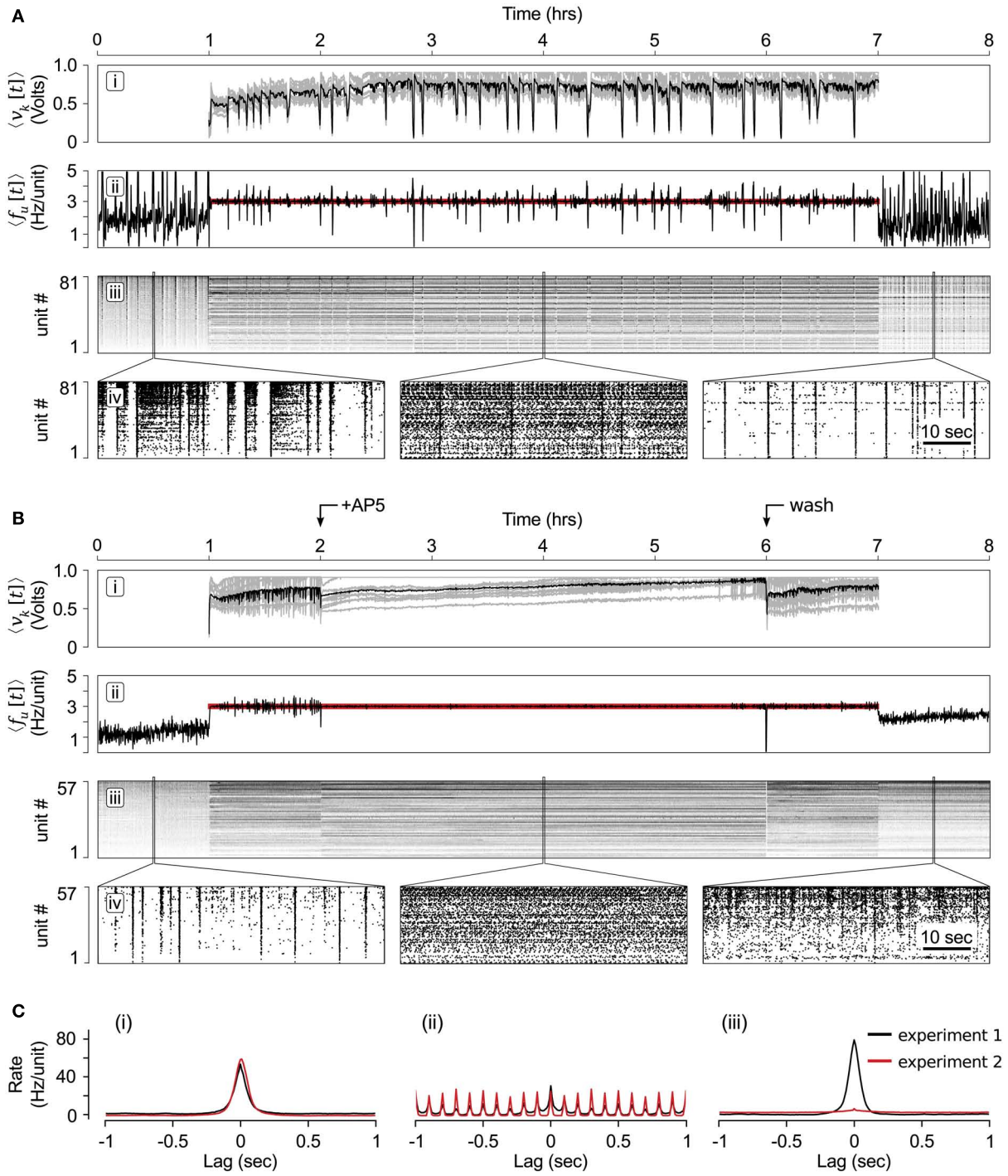

FIGURE 7 | Long-term population clamp. (A) (i) The mean stimulation voltage (black) and individual electrode stimulation voltages (gray) over the course of the 6-h clamping protocol. (ii) The neuronal firing rate (black) compared to the target rate (red line). (iii) Individual unit firing rates, sorted in order of increasing rate during the 1 h period prior to the start of closed-loop control. (iv) Zoomed rastergrams showing short time scale network spiking before, during and after the controller was engaged. **(B)** Same as **(A)** except that AP5 was added 1 h after the start of the closed-loop controller and removed 4 h later. This is indicated by the arrows at the top of the figure. **(C)** Average pair-wise correlation functions between units for experiments with and without AP5 application (red and black lines, respectively). Cross-correlations were created from spiking data (i) during spontaneous activity before the closed-loop controller was engaged, (ii) half-way through the closed-loop-control period, and (iii) during spontaneous network activity following closed-loop control. The data used to create the correlation functions is centered about locations used to create the rastergrams shown in **(A**iv**)** and **(B**iv**)**. To create the correlation functions, unit firing rates were calculated using 10 ms time bins.

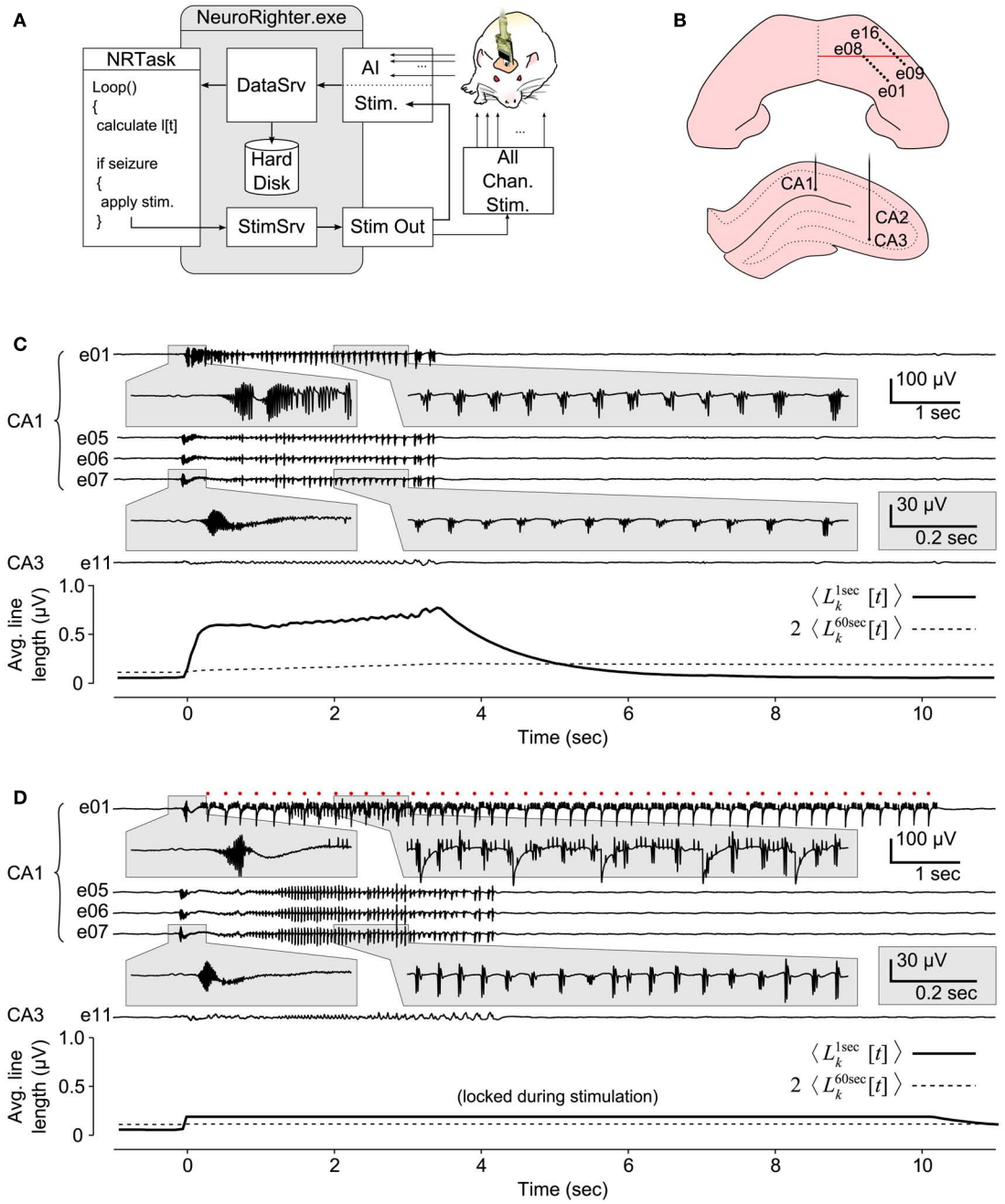

FIGURE 8 | Closed-loop seizure intervention in a freely moving rat. **(A)** Schematic of the closed-loop seizure intervention protocol. A 16-channel microwire array, with two rows of 8 electrodes, were used to record LFP signals in the CA1 and CA3 regions of the hippocampus of a epileptic rat. Paroxysmal activity in CA1 triggered the application of multichannel electrical stimulation through the recording electrodes via a stimulation multiplexing board (green). **(B)** Implantation sites of the microwire array. Top view shows the electrode penetration sites (black dots) in the right-dorsal hippocampus. The red line indicates position of the coronal view shown below. **(C)** A 12 s epoch of hippocampal LFPs during a seizure event. Electrodes 1–8 were located in CA1 and 9-16 in CA3. The line length measures, averaged across channels, are shown below the LFP traces. Seizure detection occurs at 0 s. **(D)** Same as **(C)** except with closed-loop stimulation engaged. Electrical stimulation was applied on electrode 1 along with nine other electrodes (not shown). Red dots indicate stimulation times for e01 and stimulation artifacts appear on the LFP trace. e05–e07 and e11 were not used for stimulus application.

a chronically implanted 16-channel microwire array (Tucker-Davis Technologies, Alachua, FL; **Figure 8B**). The microwire array consisted of two rows of electrodes, with 8 electrodes per row. Multi-electrode stimulation was triggered in response to detected seizures while the rat moved around its cage. To accomplish this, a "line length" measure on each LFP channel, which has been shown

to be effective for threshold based seizure detection, was calculated online (Esteller et al., 2001). A line length increment for a single LFP channel is defined as absolute difference between successive samples of the LFP,

$$l_k[t] = |x_k[t] - x_k[t - T_s]| \tag{3}$$

where $x_k[t]$ is the LFP value on the kth channel at time t, and T_s is the LFP sampling period of 500 μs. $l_k[t]$ was passed through a first order averaging filter,

$$L_k^{\tau_{filt}}[t + T_s] = l_k[t] + \exp\left(\frac{-T_s}{\tau_{filt}}\right) \cdot (L_k^{\tau_{filt}}[t] - l_k[t]) \tag{4}$$

where τ_{filt} is the filter time constant. For each recording channel, we calculated $L_k^{\tau_{filt}}[t+T_s]$ using two values of τ_{filt}, 1 and 60 s, which resulted in short and long time averages that could be compared to detect rapidly occurring trends in $l_k[t]$. Specifically, seizures were defined as events for which the criterion

$$L_k^{1sec}[t] > 2 \cdot L_k^{60sec}[t] \tag{5}$$

was met on at least 4 of the 16 recordings channels. Upon seizure detection, 10 randomly chosen electrodes were stimulated sequentially at 45 Hz (aggregate frequency) for 10 s using biphasic, 1 V, 400 μs per phase, square waves. **Figures 8C,D** shows seizure events without and with closed-loop stimulation engaged. During stimulus application, $L_k^{\alpha}[t]$ values were frozen to prevent stimulation artifacts from affecting the line length averages.

There was no easily discernible effect of microstimulation on seizure duration or intensity during this pilot experiment. However, this proof of concept demonstrates the API's utility in experiments conducted on freely moving animals and to modulate aberrant neural activity states. These features are useful for testing stimulation algorithms that do not just react to a seizure occurrence, but *predict* oncoming seizures ahead of time in order to apply a preventative action, which has proven a difficult goal to achieve (Mormann et al., 2007).

3.5. SILENT BARRAGE AND ROBOTIC EMBODIMENT

The complexity of neural systems often necessitates intricate experimental protocols for proper investigation. To meet this requirement, the plugin API can be used to integrate NeuroRighter with complicated configurations external hardware and software. Working in collaboration with the SymbioticA group at the University of Western Australia, we used NeuroRighter for intercontinental neural control of a robotic system. This project was part of an art-science collaboration called Silent Barrage (Zeller-Townson et al., 2011), in which a dissociated cortical culture in Atlanta, Georgia, USA, was embodied with a remote array of robotic drawing machines situated in an interactive art gallery[9]. This system is an extension of the MEART project (Bakkum et al., 2007).

Figure 9A shows an illustration of the Silent Barrage system. Using the plugin API, a protocol was written to communicate between NeuroRighter and a custom web server running on the same computer. The web server in turn communicated with a client computer controlling a robotic body consisting of 32 independent robots. Each robot had a rotating actuator capable of climbing up and down a vertical column (**Figure 9C**). Columns were arranged in a grid that reflected the electrode layout of the MEA (**Figures 9A,B**). The height of each rotating actuator at a given moment was determined by the instantaneous firing rate detected on two adjacent electrodes from the 59-channel MEA. As the actuators traveled up and down, they periodically marked their positions on the vertical poles using an ink pen. Over time, this resulted in a visual record of spatiotemporal activity of the culture inscribed on each column (**Figure 9C**).

Silent Barrage was exhibited in the United States (New York), Spain (Madrid), Brazil (Sao Paolo), Ireland (Dublin), and China (Beijing). Visitors to the exhibitions were encouraged to mingle amongst the robotic embodiment and they were observed using overhead cameras (**Figures 9A,B**). The resulting video feed was processed on site to extract features of audience movement (Horn and Schunck, 1981) and these data were streamed back to NeuroRighter's web server in Atlanta. Audience movement measures were then used to adjust stimulation patterns delivered through NeuroRighter's all-channel stimulator. The relationship between incoming video data and electrical stimulation varied from exhibit to exhibit, from simple single-electrode rate coding schemes to more complex multi-electrode schemes where artificial neural networks were used to deliver certain stimulus pattern based upon learned features of incoming video data. Electrical stimulation modulated the activity state of the culture's firing patterns, thus closing the loop around the dissociated culture, robotic body, and audience members separated by thousands of kilometers. While on exhibit in the National Art Museum of China, Silent Barrage was perhaps the Earth's largest behaving "organism."

4. DISCUSSION

Closed-loop electrophysiology systems are powerful tools for neuroscience research because they can be used to parse recurrent systems into independently manipulable components. Voltage clamp techniques use feedback control to separate membrane potential from the recurrent influence of voltage-dependent ionic conductances (Marmont, 1949). Seminal experiments using voltage clamp have fostered our understanding of ion channels, neuronal excitability, and synaptic transmission. More recently, dynamic clamp has been used to deliver artificial transmembrane or synaptic conductances into living neurons (Prinz et al., 2004; Kispersky et al., 2011). Using these approaches, feedback control transforms dynamic features of *individual neurons* into controlled experimental variables. Similarly, closed-loop multichannel systems like NeuroRighter can transform features of *neural networks* into controlled experimental variables (Arsiero et al., 2007). NeuroRighter is a powerful tool for controlling network variables, improving upon currently available systems in terms of cost, usability, accessibility, extensibility, and hardware standardization (Wagenaar et al., 2006a; Stirman et al., 2011; Wallach et al., 2011; Ahrens et al., 2012). We have this demonstrated NeuroRighter's power in conducting basic and translational neuroscience research through a variety of case studies.

[9]http://silentbarrage.com/

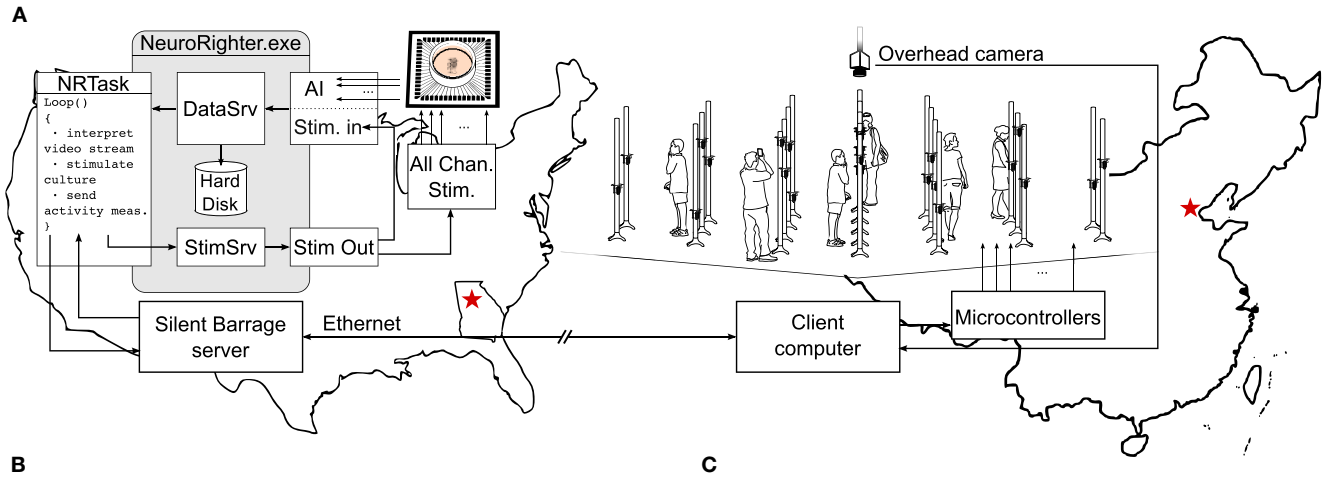

FIGURE 9 | The Silent Barrage robotic embodiment. (A) Illustration of the Silent Barrage "organism" during its exhibition at the National Art Museum of China (NAMOC), in Beijing. Spatial patterns of action potentials recorded from a dissociated cortical culture are used to drive the robotic body. A video stream of visitors to the exhibition are interpreted by NeuroRighter's plugin protocol and used to control multichannel electrical stimulation though the MEA, closing the loop around audience members, robotic system, and neural tissue over thousands of kilometers. **(B)** Audience members viewing the exhibition at NAMOC. Simultaneously, NeuroRighter translated the overhead video feed to stimulation patterns delivered to the culture and then translated resulting neuronal activity patterns to robotic actuation at the exhibit. **(C)** Photograph of an individual robot and the traces it produced during the NAMOC exhibition.

Altered gene expression, synaptic input, or environmental conditions can induce changes in spiking activity, which in turn trigger activity-dependent processes. Because of this, it becomes difficult to distinguish the role these factors play in shaping network dynamics and neural plasticity independent of firing rate. Closed-loop multichannel feedback systems provide an opportunity to render the population firing rate a controlled experimental variable and enable study of cellular and network processes as a function of a defined activity state. We used Neurorighter to clamp the firing rate of a living neural network to user-defined setpoints over both short and long timescales (Sections 3.2, 3.3). Further, we were able to control population firing rate during prolonged application of the NMDA receptor antagonist, AP5 (Section 3.3). Our controller compensated for the loss of NMDA-mediated excitation and maintained network spiking at the target firing rate. Therefore, the effects of AP5 could be deduced through comparison with a control culture that underwent an identical clamping protocol but with intact synaptic transmission. In most studies that use long-term drug application, the individual roles of spiking and excitatory neurotransmission on plasticity are ambiguous (Turrigiano, 2011). By using a real-time multichannel feedback system, we have begun to unravel the independent effects of spiking and NMDAergic transmission on network behavior. This approach could also be used to more directly study the effects of altered genetic or environmental factors on network activity.

In addition to better controlled experimental variables, real-time feedback can be used to improve the relevance of experiments using reduced neural preparations in studies of behavior. Implicit to animal behavior is the interplay between motor output and sensory perception (e.g., head movement affects the visual input stream and vice-versa). While reduced neural preparations or immobilized animals provide excellent experimental accessibility, their major weakness is that they do not preserve a functional sensory-motor loop. We have demonstrated that Neurorighter is well-equipped for performing closed-loop experiments that restore the sensory-motor loop by interfacing living neural networks with artificial bodies (Section 3.5). The advantages of this approach over traditional open-loop techniques are twofold. First,

neural systems can engage in "motor" behaviors without sacrificing delicate optical (Ahrens et al., 2012) or electrophysiological (Harvey et al., 2009) access due to actual motion. Secondly, the experimenter has complete control over the mapping between a recorded neural signal and its resulting "motor" effect (DeMarse et al., 2001; Ahrens et al., 2012). For example, Ahrens et al. (2012) recently examined optomotor adaptation in paralyzed larval zebrafish by embedding them in a virtual environment. Visual stimuli in the virtual environment provided a perception of motion, and induced fictive motor-nerve activity. Recorded motor-nerve activity was used to drive motion of the virtual environment. Changes in sensory-motor feedback gain could be achieved by adjusting the efficacy by which fictive motor patterns propelled the fish through its virtual world. All the while, full brain activity was recorded through single-cell resolution imaging, which would be nearly impossible to achieve in a freely moving animal. This study highlights how closed-loop interfaces between artificial bodies or environments and a living neural system allows excellent experimental access during behaviors requiring an intact sensory-motor loop.

Aside from basic research, closed-loop multichannel electrophysiology has possible medical applications. Predictive application of drugs or electrical stimulation has the potential to increase the efficacy and safety of treatments for various neurological disorders (Mormann et al., 2007; Rosin et al., 2011) and improve neural rehabilitation procedures (Jackson et al., 2006a). For example, a reliable seizure prediction algorithm would open the possibility for targeted interventions that abort seizures before they occur. Mormann et al. (2007) provide an extensive comparison of different methods for seizure prediction. Unfortunately, the clinical applicability of these algorithms remains quite pessimistic and future studies will require a high-throughput validation system to test robustness of seizure prediction algorithms under a variety of circumstances. We have demonstrated that NeuroRighter can be used for this purpose (Section 3.4). The stimulation algorithm we used is very similar to a method called responsive neurostimulation (NeuroPace Inc., Mountain View, CA, USA) that recently showed very promising results in a large, double-blind, pivotal clinical trial (Morrell, 2011). This form of closed-loop seizure modulation is not truly predictive as it was triggered on the occurrence of "unequivocal seizure onset" (Litt and Echauz, 2002). However, the API provides a means for easy reconfiguration in order to test alternative, predictive methods to abort seizures before they begin, using multichannel electrical stimulation or the local application of an anti-convulsive drug. Additionally, a plugin could be reconfigured for closed-loop modulation of other pathological neuronal activities or to facilitate motor rehabilitation (Jackson et al., 2006a).

Tools that enable closed-loop interaction with neural tissue at the network level have great potential to advance experimental neuroscience. Historically, open-source projects have been extremely good at adapting equipment and code designed for a singular purpose to other uses. For this reason, we envision a large role for open-source software and open-access hardware communities in the development of technologies for closed-loop eletrophysiology systems. Rapid improvements in microprocessor performance, embedded computer systems, on-chip multichannel signal processing, and A/D conversion technology must be matched by projects that can expose their powerful features for researchers with little or no background in embedded systems or computer science. NeuroRighter is one of several open-source hardware/software projects that are enabling more labs to carry out sophisticated electrophysiology with less money and more experimental flexibility[10].

ACKNOWLEDGMENTS

This work was supported by NSF COPN grant 1238097 and NIH grant 1R01NS079757-01, NSF GRFP Fellowship 08-593 to Jonathan P Newman, NSF GRFP Fellowship 09-603 to Ming-fai Fong, and NSF IGERT Fellowship DGE-0333411 to Jonathan P Newman and Ming-fai Fong. Sharanya Arcot Desai was supported by a Faculty for the Future fellowship, provided by the Schlumberger Foundation. We thank Guy Ben-Ary, Phil Gamblen, Peter Gee, Stephen Bobic, and Douglas Swehla for their contributions to the Silent Barrage robotic embodiment project. We thank J. T. Shoemaker for performing tissue harvests. We thank Ted French for his work creating the gas-permeable perfusion caps for delivery of AP5. Finally, we gratefully acknowledge all those who have contributed to NeuroRighter's hardware forums and supplied bug reports to the NeuroRighter code repository. Jon Erickson and Neal Laxpati have been especially helpful in this regard.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at http://www.frontiersin.org/Neural_Circuits/10.3389/fncir.2012.00098/abstract

[10]http://www.its.caltech.edu/~daw/meabench/
http://code.google.com/p/arte-ephys/
http://open-ephys.com/
http://www.backyardbrains.com/Home.aspx

REFERENCES

Ahrens, M. B., Li, J. M., Orger, M. B., Robson, D. N., Schier, A. F., Engert, F., et al. (2012). Brain-wide neuronal dynamics during motor adaptation in zebrafish. *Nature* 485, 471–477.

Arsiero, M., Lüscher, H. R., and Giugliano, M. (2007). Real-time closed-loop electrophysiology: towards new frontiers in in vitro investigations in the neurosciences. *Arch. Ital. Biol.* 145, 193–209.

Azin, M., and Guggenmos, D. J. (2011). A battery-powered activity-dependent intracortical microstimulation IC for brain-machine-brain interface. *IEEE J. Solid-State Circuits* 46, 731–745.

Bakkum, D. J., Gamblen, P. M., Ben-Ary, G., Chao, Z. C., and Potter, S. M. (2007). MEART: The Semi-Living Artist. *Front. Neurorob.* 1:5. doi:10.3389/neuro.12.005.2007

Cole, K. S. (1949). Dynamic electrical characteristics of the squid axon membrane. *Annu. Rev. Physiol.* 3, 253–258.

DeMarse, T. B., Wagenaar, D. A., Blau, A. W., and Potter, S. M. (2001). The neurally controlled animat: biological brains acting with simulated bodies. *Auton. Robots* 11, 305–310.

Desai, S. A., Rolston, J. D., Guo, L., and Potter, S. M. (2010). Improving impedance of implantable microwire multi-electrode arrays by ultrasonic electroplating of durable platinum black. *Front. Neuroeng.* 3:5. doi:10.3389/fneng.2010.00005

Du, J., Blanche, T. J., Harrison, R. R., Lester, H., and Masmanidis, S. C. (2011). Multiplexed, high density electrophysiology with nanofabricated neural probes. *PLoS ONE* 6:e26204. doi:10.1371/journal.pone.0026204

Esteller, R., Echauz, J., and Tcheng, T. (2001). Line length: an efficient feature for seizure onset detection. *Conf. Proc. IEEE Eng. Med. Biol. Soc.* 2, 1707–1710.

Fiscella, M., Farrow, K., Jones, I. L., Jäckel, D., Müller, J., Frey, U., et al. (2012). Recording from defined populations of retinal ganglion cells using a high-density CMOS-integrated microelectrode array with real-time switchable electrode selection. *J. Neurosci. Methods* 211, 103–113.

Gal, A., Eytan, D., Wallach, A., Sandler, M., Schiller, J., and Marom, S. (2010). Dynamics of excitability over extended timescales in cultured cortical neurons. *J. Neurosci.* 30, 16332–16342.

Hamill, O. P., Marty, A., Neher, E., and Sakmann, B. (1981). Improved patch-clamp techniques for high-resolution current recording from cells and cell-free membrane patches. *Pflugers Arch.* 391, 85–100.

Harvey, C. D., Collman, F., Dombeck, D. A., and Tank, D. W. (2009). Intracellular dynamics of hippocampal place cells during virtual navigation. *Nature* 461, 941–946.

Hawkins, C. A., and Mellanby, J. H. (1987). Limbic epilepsy induced by tetanus toxin: A longitudinal electroencephalographic study. *Epilepsia* 28, 431–444.

Horn, B. K. P., and Schunck, B. G. (1981). Determining optical flow. *Artif. Intell.* 17, 185–203.

Jackson, A., Mavoori, J., and Fetz, E. E. (2006a). Long-term motor cortex plasticity induced by an electronic neural implant. *Nature* 444, 56–60.

Jackson, A., Moritz, C. T., Mavoori, J., Lucas, T. H., and Fetz, E. E. (2006b). The Neurochip BCI: towards a neural prosthesis for upper limb function. *IEEE Trans. Neural Syst. Rehabil. Eng.* 14, 187–190.

Kispersky, T. J., Economo, M. N., Randeria, P., and White, J. A. (2011). GenNet: a platform for hybrid network experiments. *Front. Neuroinform.* 5:11. doi:10.3389/fninf.2011.00011

Litt, B., and Echauz, J. (2002). Prediction of epileptic seizures. *Lancet Neurol.* 1, 22–30.

Marmont, G. (1949). Studies on the axon membrane. *J. Cell. Comp. Physiol.* 34, 351–381.

Minerbi, A., Kahana, R., Goldfeld, L., Kaufman, M., Marom, S., and Ziv, N. E. (2009). Long-term relationships between synaptic tenacity, synaptic remodeling, and network activity. *PLoS Biol.* 7:e1000136. doi:10.1371/journal.pbio.1000136

Mormann, F., Andrzejak, R. G., Elger, C. E., and Lehnertz, K. (2007). Seizure prediction: the long and winding road. *Brain* 130, 314–333.

Morrell, M. J. (2011). Responsive cortical stimulation for the treatment of medically intractable partial epilepsy. *Neurology* 77, 1295–1304.

Nakanishi, K., and Kukita, F. (1998). Functional synapses in synchronized bursting of neocortical neurons in culture. *Brain Res.* 795, 137–146.

Nelson, T. S., Suhr, C. L., Freestone, D. R., Lai, A., Halliday, A. J., Mclean, K. J., et al. (2011). Closed-loop seizure control with very high frequency electrical stimulation at seizure onset in the Gaers model of absence epilepsy. *Int. J. Neural Syst.* 21, 163.

Potter, S. M., and DeMarse, T. B. (2001). A new approach to neural cell culture for long-term studies. *J. Neurosci. Methods* 110, 17–24.

Prinz, A. A., Abbott, L. F., and Marder, E. (2004). The dynamic clamp comes of age. *Trends Neurosci.* 27, 218–224.

Reger, B. D., Fleming, K. M., Sanguineti, V., and Alford, S. (2000). Connecting brains to robots: an artificial body for studying the computational properties of neural tissues. *Artif. Life* 324, 307–324.

Rich, M. M., and Wenner, P. (2007). Sensing and expressing homeostatic synaptic plasticity. *Trends Neurosci.* 30, 119–125.

Robinson, J. T., Jorgolli, M., Shalek, A. K., Yoon, M.-H., Gertner, R. S., and Park, H. (2012). Vertical nanowire electrode arrays as a scalable platform for intracellular interfacing to neuronal circuits. *Nat. Nanotechnol.* 7, 180–184.

Rolston, J. D., Gross, R. E., and Potter, S. M. (2009a). A low-cost multielectrode system for data acquisition enabling real-time closed-loop processing with rapid recovery from stimulation artifacts. *Front. Neuroeng.* 2:12. doi:10.3389/neuro.16.012.2009

Rolston, J. D., Gross, R. E., and Potter, S. M. (2009b). Common median referencing for improved action potential detection with multielectrode arrays. *Conf. Proc. IEEE Eng. Med. Biol. Soc.* 2009, 1604–1607.

Rolston, J. D., Gross, R. E., and Potter, S. M. (2010). Closed-loop, open-source electrophysiology. *Front. Neurosci.* 4:31. doi:10.3389/fnins.2010.00031

Rosin, B., Slovik, M., Mitelman, R., Rivlin-Etzion, M., Haber, S. N., Israel, Z., et al. (2011). Closed-loop deep brain stimulation is superior in ameliorating parkinsonism. *Neuron* 72, 370–384.

Stirman, J. N., Crane, M. M., Husson, S. J., Wabnig, S., Schultheis, C., Gottschalk, A., et al. (2011). Real-time multimodal optical control of neurons and muscles in freely behaving Caenorhabditis elegans. *Nat. Methods* 8, 153–158.

Strong, S. P., Koberle, R., de Ruyter Van Stevenvick, R. R., and Bialek, W. (1998). Entropy and information in neural spike trains. *Phys. Rev. Lett.* 80, 197–200.

Sun, F. T., Morrell, M. J., and Wharen, R. E. (2008). Responsive cortical stimulation for the treatment of epilepsy. *Neurotherapeutics* 5, 68–74.

Supèr, H., and Roelfsema, P. R. (2005). Chronic multiunit recordings in behaving animals: advantages and limitations. *Prog. Brain Res.* 147, 263–282.

Tchumatchenko, T., Geisel, T., Volgushev, M., and Wolf, F. (2010). Signatures of synchrony in pairwise count correlations. *Front. Comput. Neurosci.* 4:1. doi:10.3389/neuro.10.001.2010

Turrigiano, G. (2011). Too many cooks? Intrinsic and synaptic homeostatic mechanisms in cortical circuit refinement. *Annu. Rev. Neurosci.* 34, 89–103.

Turrigiano, G. G., Leslie, K. R., Desai, N. S., Rutherford, L. C., and Nelson, S. B. (1998). Activity-dependent scaling of quantal amplitude in neocortical neurons. *Nature* 391, 892–896.

Wagenaar, D. A., Madhavan, R., Pine, J., and Potter, S. M. (2005). Controlling bursting in cortical cultures with closed-loop multi-electrode stimulation. *J. Neurosci.* 25, 680–688.

Wagenaar, D. A., Pine, J., and Potter, S. M. (2004). Effective parameters for stimulation of dissociated cultures using multi-electrode arrays. *J. Neurosci. Methods* 138, 27–37.

Wagenaar, D. A., Pine, J., and Potter, S. M. (2006a). An extremely rich repertoire of bursting patterns during the development of cortical cultures. *BMC Neurosci.* 7:11. doi:10.1186/1471-2202-7-11

Wagenaar, D., DeMarse, T. B., and Potter, S. M. (2006b). "MEABench: a toolset for multi-electrode data acquisition and on-line analysis," in *International IEEE EMBS Conference on Neural Engineering*, Washington, 518–521.

Wagenaar, D. A., and Potter, S. M. (2002). Real-time multi-channel stimulus artifact suppression by local curve fitting. *J. Neurosci. Methods* 120, 113–120.

Wagenaar, D. A., and Potter, S. M. (2004). A versatile all-channel stimulator for electrode arrays, with real-time control. *J. Neural. Eng.* 1, 39–45.

Wallach, A., Eytan, D., Gal, A., Zrenner, C., and Marom, S. (2011). Neuronal response clamp. *Front. Neuroeng.* 4:3. doi:10.3389/fneng.2011.00003

Yu, Y., Crumiller, M., Knight, B., and Kaplan, E. (2010). Estimating the amount of information carried by a neuronal population. *Front. Comput. Neurosci.* 4:10. doi:10.3389/fncom.2010.00010

Zanos, S., Richardson, A. G., Shupe, L., Miles, F. P., and Fetz, E. E. (2011). The Neurochip-2: an autonomous head-fixed computer for recording and stimulating in freely behaving monkeys. *IEEE Trans. Neural Syst. Rehabil. Eng.* 19, 427–435.

Zeller-Townson, R., Ben-Ary, G., and Gamblen, P. (2011). "Silent barrage: interactive neurobiological art," in *The Proceedings of the 8th ACM Conference on Creativity and Cognition*, Atlanta, 407–408.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 01 August 2012; accepted: 18 November 2012; published online: 18 January 2013.

*Citation: Newman JP, Zeller-Townson R, Fong M-F, Arcot Desai S, Gross RE and Potter SM (2013) Closed-loop, multichannel experimentation using the open-source NeuroRighter electrophysiology platform. Front. Neural Circuits **6**:98. doi: 10.3389/fncir.2012.00098*

SUPPLEMENTARY MATERIAL - Closed-loop, multichannel experimentation using the open-source NeuroRighter electrophysiology platform

Jonathan P Newman,[1] Riley Zeller-Townson,[1] Ming-fai Fong,[2,1] Sharanya Arcot Desai,[1,3] Robert E. Gross,[3,4] and Steve M Potter[1]
[1]*Laboratory for Neuroengineering, Dept. of Biomedical Engineering, Georgia Institute of Technology and Emory University School of Medicine, Atlanta, Georgia, 30332, USA*
[2]*Dept. of Physiology, Emory University School of Medicine, Atlanta, Georgia, 30322, USA*
[3]*Dept. of Neurosurgery and Neurology, Emory University School of Medicine, Atlanta, Georgia, 30322, USA*
[4]*Dept. of Biomedical Engineering, Georgia Institute of Technology and Emory University School of Medicine, Atlanta, Georgia, 30332, USA*

(Dated: January 1, 2013)

Keywords: closed-loop, multichannel, real-time, multi-electrode, micro-electrode array, electrophysiology, open-source, network

CONTENTS

I. Scripted Output 2

II. Using the real-time API 4
A. Using a pre-compiled real-time plugin 4
B. Writing a real-time plugin 4
C. Event-based methods 8
D. Reading data within a plugin 8
E. Generating output from a plugin 9

III. Spike detection and sorting with NeuroRighter 10
A. Spike detection and validation 10
B. Spike sorting 11

IV. Experimental methods 12
A. Tissue culture 12
B. In-vitro MEA electrophysiology 12
C. Animal surgery, recordings, and multielectrode stimulation 14

V. Latency measurement plugin code 15
A. StimSrv-based real-time loop 15
B. NewData-based real-time loop 18
C. Arduino-based real-time loop 21

Cited in Supplementary Material 23

I. SCRIPTED OUTPUT

Real-time plugins have on-the-fly access to NeuroRighter's output servers. However, NeuroRighter's output servers can also be controlled in open-loop mode using predefined stimulation scripts, referred to here as 'scripted output'. Scripted output allows the generation of continuous, non-periodic, uninterrupted output streams on all of NeuroRighter's available output channels (Fig. 2 and table I of the main text). Using scripted output, manipulation of electrical stimulation lines, analog outputs, and digital outputs occurs with 10 μs precision relative to the start of a clock-synchronized recording session. Scripts can be generated within MATLAB(R) (Mathworks, Natick, MA) or Octave[1] using three functions provided with a standard NeuroRighter installation: `makestimfile.m`, `makeauxfile.m`, and `makedigfile.m`. Each of these functions contains documentation in their file header, which can be accessed using the '`help`' command.

In the following example, we generate two output files that control NeuroRighter's analog and digital output streams, respectively. The first file produces 10 changes in the voltage of analog output channel 0 (AO.0). The second file encodes the voltage changes on AO.0 using the first 8 bits of the digital output port 0 (P0.0-7). Changes in the voltage of AO.0 and port state of P0.0-7 will occur once a second relative to the start of the recording, on the same 100 kHz clock edge.

```
t = (1:10)';              % 1,2,3,...,10 seconds
a = rand(10,1);           % 10 random voltages
c = zeros(size(a));       % use channel AO.0
d = ceil(a*255);          % encode 'a' with 8-bit resolution on P0.0-7
```

[1] `http://www.gnu.org/software/octave/`

```
makeauxfile('myAuxTest',t,c,a);
makedigfile('myDigTest',t,d);
```

This MATLAB script produces two files, `myAuxTest.olaux` and `myDigTest.olaux`, which can be loaded into NeuroRighter's GUI for execution. The user must enable analog and digital output lines in NeuroRighter's hardware settings in order for these scripts to function (section II.B.1 of the main text and Fig. S1(c) of this document). This example illustrates an advantage of having direct access to the analog and digital output lines: because all physical inputs and outputs to and from NeuroRighter are synchronized to the same hardware clock, the digital output that encodes changes in analog voltage AO.0 can be recorded instead of AO.0's raw voltage. Since only changes in the state of the digital port are recorded, this vastly reduces the amount of disk space required to encode the delivered output stream.

Aside from the manipulation of the NI cards' analog and digital output lines, scripted protocols can be created to drive NeuroRighter's all-channel electrical micro-stimulation board. Below, we provide a MATLAB script that produces the scripted stimulation protocol used to generate Fig. 2 of the main text.

```
% Stimulation protocol with 1 second epochs of a new or repeated
% realization of spatially uniform random, temporally Poisson,
% voltage-controlled square wave stimuli. New and repeated stimulation
% trials are interleaved in time with no down time between trials.

lambda = 15;                        % Poisson rate parameter (1/seconds)
num_trials = 100;                   % Number of random realizations
start_time = 60;                    % Protocol start time (seconds)

tot = start_time;                   % Total time;
T = [];                             % Stimulation times
C = [];                             % Stimulation channels

% Uniform random points over defined time epoch is equivalent
% to Possionian point process (exp dist. of times between stimuli).
rept = rand(lambda,1);
repc = 59*rand(size(rept));

for i = 1:num_trials

    % New random realization
    T = [T;tot(end) + rand(lambda,1)];
    C = [C;59*rand(size(rept))];
    tot = tot + 1; % 1 sec. increment

    % Repeated realization
    T = [T;tot(end) + rept];
    C = [C;repc];

end

% Waveform is +-0.75 volt, 400 us/phase, postive-first, square-wave.
% This can also be used as a current waveform by adjusting NR's
% hardware settings to enable current-controlled stimulation.
fs = 1e5;
w = [zeros(1,5) 0.75*ones(1,40) -0.75*ones(1,40) zeros(1,5)];
W = w(ones(size(T)),:);

% Write the file
makestimfile('pois-stim-protocol',T,C,W)
```

II. USING THE REAL-TIME API

A real-time plugin is an externally compiled class library, written in C♯ or some other .NET supported language, that can be used for a wide variety of tasks. For instance, plugins allow on-the-fly manipulation NeuroRighter's electrical stimulation board as well as analog and digital output lines, with 10 μs precision. Plugins can also be used to change NeuroRighter's outputs as a function of incoming data, such as neural recordings, auxiliary analog inputs, or digital input channels in order to close the loop around neural tissue. Since plugins are externally compiled programs, they can be created without editing NeuroRighter's source code, and can reference third party libraries, such as those available for sending information over the Internet or for executing MATLAB code. In the following section we demonstrate the creation and use of real-time plugins with NeuroRighter.

A. Using a pre-compiled real-time plugin

A compiled plugin is a class library, or, in Microsoft parlance, a 'Dynamic-linked Library' (DLL) file. A DLL is a type of Windows file which contains compiled code that cannot be executed by itself (like an executable or '*.exe' file), but can be referenced or used by executable code. To start, we will detail the usage of a plugin that has already been complied and exists as a DLL file. The compiled DLLs that were used in the case-studies in the main text (section III) are available online[2], an can be used as described in the following paragraphs.

After downloading a pre-compiled plugin, we need to configure NeuroRighter's hardware settings to match the details of our recording system (see section II.B.1 and table I of the main text). Figure S1 outlines this process for a 64 channel MEA1060-Up amplifier (Multichannel Systems, Reutlingen, Germany) with a 1200X pass-band gain. First we enter the 'Real-time' tab (Fig. S1(a)) where we can specify the A/D and D/A polling periods (which determine the loop speed of the plugin), and where we select which data servers we want to expose to the plugin. For example, to run the closed-loop reaction time case-study (section III.A of the main text), we only needed to select the 'Spike Data' buffer. Data servers are inactive by default, in order to decrease computational overhead and memory requirements. Next, in the 'Neural Input' tab (Fig. S1(b)), we supply our amplifier gain and select the NI boards being used to route neural signals. Finally, in the 'Output' tab, we can define how to deliver electrical stimuli and whether we want to use generic digital or analog output lines.

After configuring hardware and streaming settings, we can load the plugin library. To do this, we open NeuroRighter's 'Stim/Output' tab (Fig. S2(a)) and then navigate to the 'Real-time Plugin' box (Fig. S2(b)). A plugin library is a collection of plugin classes that are all stored in the same file. To load a library, we click the '...' button in the Real-Time Plugin box and select the DLL file that we have downloaded or compiled. After we select a library, the drop down list underneath the load button is populated with the names of the individual real-time plugin classes that are stored inside the selected library.

After selecting the plugin we want to execute, we click the 'Start' button in the 'Real-Time Plugin' box to activate the plugin and begin data acquisition. The plugin can be interrupted at any time by clicking the 'Stop' button in the 'Real-Time Plugin' box. A final detail worth mentioning is that plugins can use stimulus waveforms that are designed within the NeuroRighter GUI (Fig. S2(c)) if the plugin references this `PredefinedWaveform` in its code. Referencing this GUI-defined waveform is a way to quickly configure stimulation parameters without writing any code or recompiling the plugin library.

B. Writing a real-time plugin

The source-code for all plugins used as case-studies in the main text are available online[3]. Additionally, the code used to produce the first case-study from the main text (section III.A) is provided at the end of this document, in section V. In this section, we walk through the steps required to write a custom real-time plugin for NeuroRighter.

[2] `http://code.google.com/p/neurorighter/downloads/list`

[3] `http://code.google.com/p/neurorighter/source/browse/NR-ClosedLoop-Examples/`

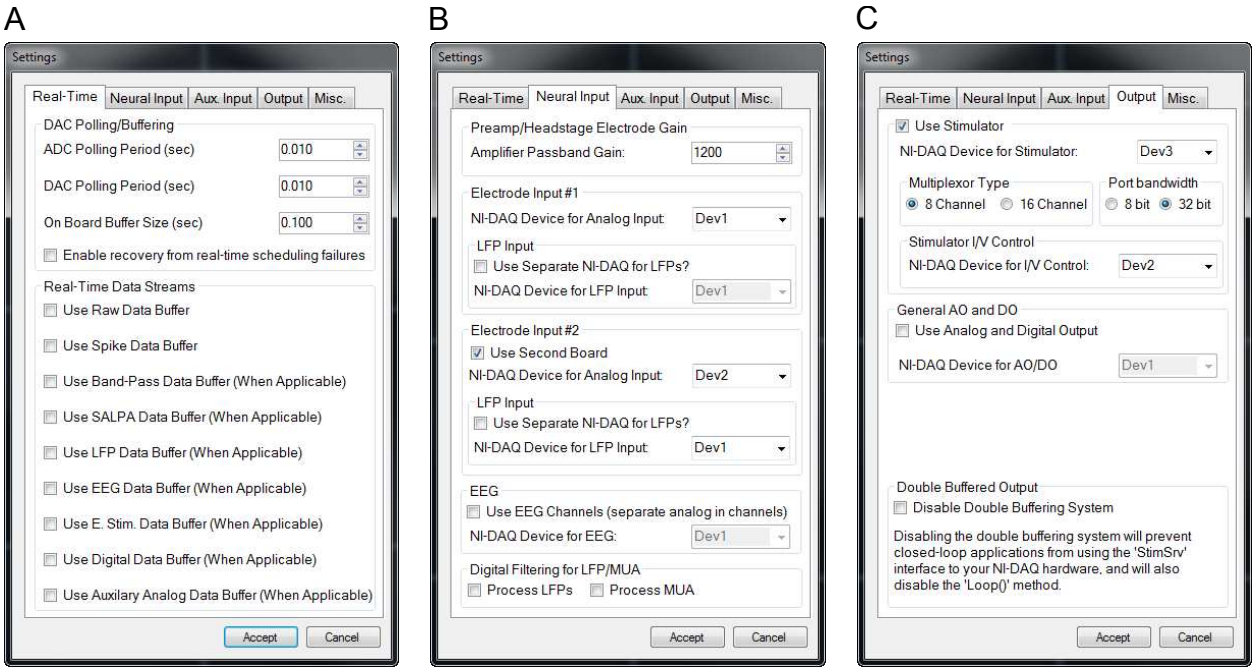

FIG. S1: Configuring hardware settings for a real-time plugin. **(A)** The 'Real-Time' tab is used to adjust the hardware polling period and configure NeuroRighter's data servers. **(B)** The 'Neural Input' tab is used to configure multichannel amplifier settings and specifies which NI boards are used to acquire raw, LFP, and EEG voltages. **(C)** The 'Output' tab is used to configure electrical stimulation as well as generic analog and digital output lines.

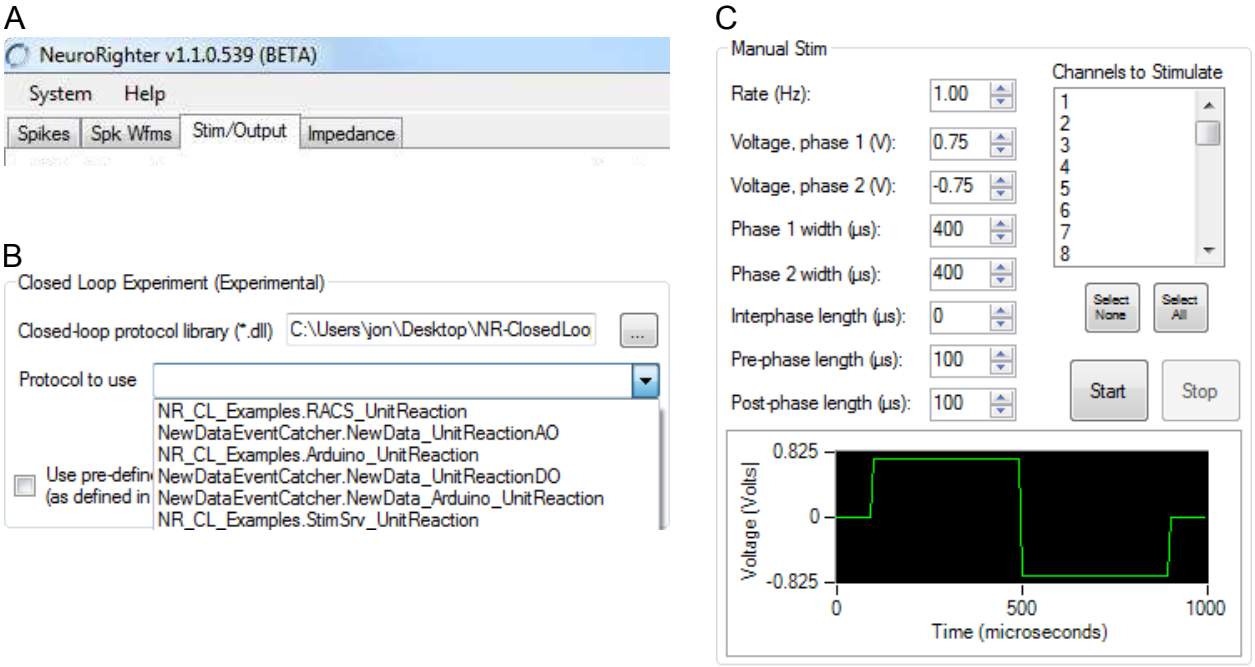

FIG. S2: Loading a plugin library. **(A)** Plugin libraries are loaded through the 'Stim/Output' tab on the main NeuroRighter interface. **(B)** Once a plugin library is loaded, all the classes it encapsulates are exposed for execution. **(C)** Simple stimulus waveforms can be designed in NeuroRighter's GUI and accessed from a closed-loop plugin.

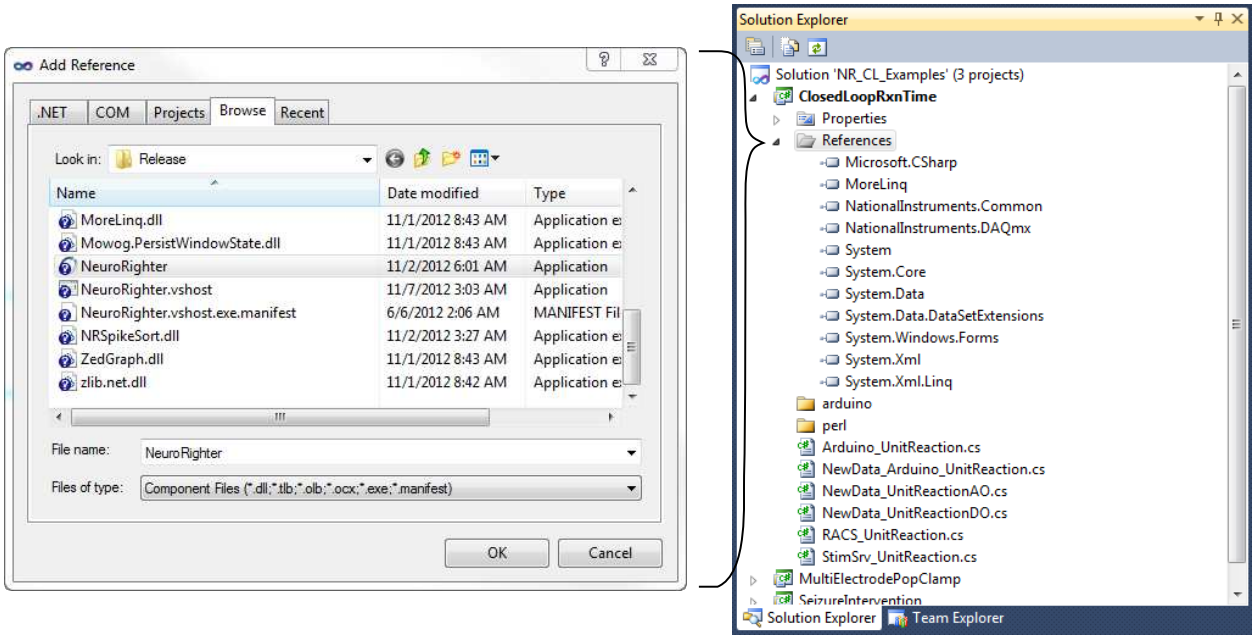

FIG. S3: Referencing the NeuroRighter executable to create a real-time plugin.

Creating a real-time **plugin library** consists of creating a class library, writing a **plugin class** within that library, and adding special methods to the plugin class that will be activated by NeuroRighter. All plugin classes follow a similar code template, regardless of their complexity and have automatic access to NeuroRighter's data servers as well as NeuroRighter's stimulation servers. Different integrated development environments (IDEs) can be used to write a plugin library. Two good development options are Microsoft Visual Studio (Microsoft, Redmond, WA) or the open-source and free MonoDevelop[4]. The steps required for creating a plugin do not change significantly for different IDE options.

Our first step to write a new plugin is to create a new project that will contain our plugin library. This can be done by creating a new project of the type 'Class Library' in your IDE of choice (this should be something similar to: file→ new→ project→ class library). In general, the difference between an executable and a class library is that a class library lacks a generic entry point (nominally a '`main()`' method) from which to start code execution. Instead, it requires a parent executable to determine how the library code should be entered and executed at runtime. In our case, the parent executable is NeuroRighter, so we must make our newly created class library aware of NeuroRighter's existence. To do this, we reference the NeuroRighter executable from our library by adding NeuroRighter.exe to the 'References' list associated with our project (Fig. S3). This exposes NeuroRighter's public namespaces to our library, which then can be used by individual classes with the '`using`' keyword. These namespaces are described in table II of the main text (and referred to their as 'packages' to avoid programming jargon) and are reiterated here.

NeuroRighter.NeuroRighterTask: This namespace must be referenced in every real-time plugin class, as it contains the `NRtask` abstract class that all plugin classes must implement to integrate with NeuroRighter while it is running.

NeuroRighter.Server: This namespace provides access to NeuroRighter's `NRDataSrv` data server object and `NRStimSrv` output server object.

NeuroRighter.DataTypes: This namespace provides several data types that are used by `NRDataSrv` and `NRStimSrv` (e.g. `SpikeEvent`, `AuxOutEvent`, etc.).

[4] `http://monodevelop.com/`

NeuroRighter.dbg: This namespace provides access to a debugging tool that allows user messages to be logged to a text file along with timing information that is generated by NeuroRighter during plugin execution.

Next, we create a new class that will define our plugin. A plugin class inherits functionality from the base-class `NRTask` which provides automatic access to the following features.

- Overrideable methods for setting up (the `Setup()` method), executing (the `Loop()` method), and cleaning up (the `Cleanup()` method) a real-time loop within NeuroRighter.
- Access to NeuroRighter's server objects `NRDataSrv` and `NRStimSrv`, which are used to read from and write to hardware. Manipulation of the objects requires including the `Server` namespace, as mentioned above.
- Information concerning the state of the NeuroRighter executable, such as whether or not it is currently recording data and the file-paths being used to save data.

The basic structure of a plugin class mimics high-level programming languages for micro-controllers, such as Arduino[5]. Below, we provide an example of a very simple plugin.

```
using NeuroRighter.Output
namespace NRpluginExample
{
  public class HelloWorld: NRtask
  {
    override void Setup()
    {
      Console.WriteLine("Hello, World!");
    }

    override void Loop()
    {
      Console.WriteLine("I'm still here, World!");
    }

    override void Cleanup()
    {
      Console.WriteLine("Goodbye, World!");
    }
  }
}
```

In this example, `NRpluginExample` is the name of the library that is being created, and `HelloWorld` is the name of a single plugin class within that library. When we are finished writing our class, we can compile the class library. In our example, this would result in NRpluginExample.dll, which can be loaded into the NeuroRighter GUI as described above.

Once loaded into NeuroRighter, the 'Start' button is clicked which starts the plugin's execution along with data acquisition in NeuroRighter. When start is clicked, the `Setup()` method is called a single time. In our example, this would result in "Hello, World" being printed to the console (System→ Show Console, from within NeuroRighter). After the `Setup()` method finishes, NeuroRighter's input and output servers are activated, and the `Loop()` method is called periodically (Fig. 2 of the main text), at a frequency specified by the DAC polling period (Fig. S1). This would result in "I'm still here, World!" being printed to the console many times, in rapid succession. Execution would continue until the user clicks the 'Stop' button, which calls the `Cleanup()` method. At this point our example plugin would display the message "Goodbye, World!" to the console and NeuroRighter's input and output servers would shut down.

[5] `http://arduino.cc/en/`

C. Event-based methods

Note that, in contrast to the example above, a plugin is not required to override any of the three base methods supplied by the `NRTask` class. For example, instead of using the `Loop` function, which is executed periodically by NeuroRighter's output servers, we can execute code upon the capture of new data into NeuroRighter's input servers using `NewData` events (see section II.B.2 of the main text and section V.B of this document). For example, the following code snippet will write the line "New spikes captured" to the console immediately after new spikes are pushed to the Spike server data stream. As shown in Fig. 4 of the main text, this method provides the lowest feedback latency.

```
override void Setup()
{
  // Subscribe to the NewData event on the SpikeSrv data server
  NRDataSrv.SpikeSrv.NewData +=
    new EventDataSrv<NeuroRighter.DataTypes.SpikeEvent>.
      NewDataHandler(SpikeSrv_NewData);
}

// This method is called every time the SpikeSrv.NewData event fires
private void SpikeSrv_NewData()
{
  Console.Writeline("New spikes captured!");
}
```

D. Reading data within a plugin

Reading data streams within a plugin is performed using the `NRDataSrv` object. `NRDataSrv` contains a set of servers that buffer the most recent data collected by NeuroRighter. `NRDataSrv` encapsulates two general types of data, continuous data and packet data. **Continuous data** is defined by an NxM dimensional array, where N is the channel count, and M is the sampling frequency multiplied by the buffer's duration. Continuous data servers buffer raw voltage values from electrodes, auxiliary analog streams, or filtered outputs from the spike, SALPA, LFP and EEG streams. NeuroRighter's continuous data servers are listed in the top section of table SI. **Packet data** is defined by asynchronous physical events, that are not necessarily temporally periodic (e.g. detected spikes). Like continuous data buffers, packet buffers contain data over the time period specified in hardware settings but, since they are not sampled continuously, the amount of data in these buffers can fluctuate. For example, if the spike detection rate increases, the number of spikes in `NRDataSrv.SpikeSrv` will increase. NeuroRighter's packet data servers are listed in the bottom section of table SI.

All data servers are null by default in order to decrease computational overhead and memory requirements. They can be activated using the hardware settings GUI (Fig. S1(a)). The amount of data that is buffered by each server object can also be set in the hardware settings GUI. The default buffer history is 1 second. This means that at any point in time, the plugin has access to data that is up to 1 second old. If a plugin needs access to older data, hardware settings must be configured to allow for a longer buffer or the plugin must implement its own buffer to save past data values.

To read from continuous- or packet-based data servers, the `ReadFromBuffer` method is used. The `ReadFromBuffer` method is an attribute of each server object that takes start and stop sample indices as input arguments. For example, to read all the spikes that occurred between times 10 seconds and 12 seconds relative to the start of the recording, we could use the following code snippet:

TABLE SI: NeuroRighter's data servers. All servers are null unless activated using the hardware settings GUI.

	Server Name	Description
Continuous Data Servers	`NRDataSrv.AuxAnalogSrv`	Provides access to data from auxiliary analog channels
	`NRDataSrv.EEGSrv`	Provides access to data from EEG channels
	`NRDataSrv.LFPSrv`	Provides access to data from LFP channels
	`NRDataSrv.RawElectrodeSrv`	Provides access to raw electrode voltages.
	`NRDataSrv.FilteredElectrodeSrv`	Provides access to data produced by the 'spike filter' which is used to pass signals from ∼100 to 5000 Hz.
	`NRDataSrv.SalpaElectrodeSrv`	Provides access to data processed by the SALPA filter, which is used for rapid electrical artifact subtraction (Wagenaar and Potter, 2002)
Packet Data Servers	`NRDataSrv.SpikeSrv`	Provides access to recently detected spikes (containing time, channel, waveform, unit number, etc.)
	`NRDataSrv.AuxDigitalSrv`	Provides access to the state changes recently detected on the auxiliary digital input lines (time and port state)
	`NRDataSrv.StimSrv`	Provides access to recently recorded electrical stimuli (time, channel, voltage, etc.)

```
// Convert required read start and stop times to samples
ulong start = (ulong) (10*NRDataSrv.SpikeSrv.SampleFrequencyHz);
ulong stop = (ulong) (12*NRDataSrv.SpikeSrv.SampleFrequencyHz);

// Read samples from the buffer
EventBuffer<SpikeEvent> recordedSpikes;
recordedSpikes = NRDataSrv.SpikeSrv.ReadFromBuffer(start, stop);
```

At this point, all the spikes recorded between seconds 10 and 12 are assigned to the `recordedSpikes` object (assuming they were available in the buffer when the read call was made). To find what samples are currently available, the `EstimateAvailableTimeRange()` method can be used. Alternatively, as described above, the data server can inform listening processes, like our plugin, whenever new data is available using a `NewData` event.

E. Generating output from a plugin

The easiest way to change NeuroRighter's outputs from within a plugin is to use the `NRStimSrv` object, which is automatically provided to classes that are derived from `NRTask`. As noted in the section II.B.2 of the main text, this is not the only option to produce output with NeuroRighter. Standard communication protocols (TCP/IP, USB, RS232, etc.) as well as direct communication with the NI boards can also be used (see, for instance, section V.B and V.C). Analogous to `NRDataSrv`, `NRStimSrv` encapsulates server objects that can be used to generate physical outputs from within a plugin. These servers are:

NRStimSrv.AuxOut: This server is used to write values to the analog auxiliary channels.

NRStimSrv.DigitalOut: This server is used to write digital values out to the auxiliary digital port.

NRStimSrv.StimOut: This server is used to write electrical stimuli.

Writing to these servers is accomplished using the `WriteToBuffer` method. For example, if we wanted a plugin to generate electrical stimuli at times 1.2, 1.5, 1.7 seconds, on channels 3, 7, and 9, using the waveform specified in the 'Manual Stim' GUI (Fig. S2(c)), we could use the following code segment within our plugin class:

```
// Define stimulus times and channels
double[] times = {1.2,1.5,1.7};
int[] channels = {3,7,9};

override void Setup()
{
  // Here, we access the waveform created within NR's 'Manual Stim' GUI
  double[] waveform = PredefinedWaveform;

  // Grab the output sampling frequency
  double fs = NRStimSrv.SamplingFrequencyHz;

  List<StimulusOutEvent> stimuli = new List<StimulusOutEvent>();
  for (int i = 0; i < times.Length; i++)
  {
    stimuli.Add(new StimulusOutEvent(channels[i],(ulong)(times[i]*fs), waveform));
  }

  // Add the stimuli to the buffer - one time write since we are in Setup()
  NRStimSrv.StimOut.WriteToBuffer(stimuli)
}
```

Like when reading data from input servers, the timing of these outputs are measured in samples since the experiment began. All output servers operate with a hard-coded 10 μs precision. To find the next sample available for writing on a particular output buffer during a real-time protocol, the `GetTime()` or `GetCurrentSample()` methods can be used.

For more information regarding the usage and creation of real-time plugins with NeuroRighter, visit NeuroRighter's website and examine the example code provided in section V of this document.

III. SPIKE DETECTION AND SORTING WITH NEURORIGHTER

A. Spike detection and validation

Putative spikes are detected as voltage samples, $v_k[t]$ for which

$$|v_k[t]| > \gamma V_k^{\mathrm{RMS}}, \tag{1}$$

where γ is a user-defined coefficient (table SII). When the detection criterion is met, $v_k[t]$ is searched for a voltage peak or trough following the threshold crossing. V_k^{RMS} is defined as the average of lowest 10% of RMS values calculated from a group of 100 millisecond data windows of voltage values taken from channel k. This method prevents overestimates of V_k^{RMS} by excluding windows with large amounts of spiking activity from the RMS calculation (Wagenaar et al., 2006a). V_k^{RMS} can be a fixed value, based upon the first 10 seconds (100 windows) of a recording or the 10 seconds after the user clicks the 'Retrain' button. Alternatively, V_k^{RMS} can be calculated adaptively using a sliding 10 second window to adapt to changes in channel noise levels. Following detection, a short voltage 'snippet', aligned at the absolute voltage peak, is then extracted from the raw trace. The number of voltage samples included in each snippet is user-defined (table SII).

Following detection, spike snippets are validated as true action potentials using a series of tests based on waveform slope, width and peak-to-peak amplitude (Figure S4, table SII). If a spike is successfully validated, a detection pause is enforced on the channel of origin to prevent multiple detections of a single spike that contains multiple peaks. Following spike validation, the spike snippet and associated information (time of occurrence, channel, etc.) is pushed to the `SpikeSrv.Buffer` data stream (table I of the main text).

TABLE SII: User defined parameters for spike detection, validation, and sorting.

	Parameter	Description	Typical Value
Detection	Threshold (γ)	RMS multiplier defining detection threshold	4-7
	Pre-spike time	Time before spike to store in snippet	0.5 msec
	Post-spike time	Time following spike to store in snippet	1.5 msec
	Noise estimation algorithm	Method used to estimate RMS noise	adaptive/fixed
	Spike alignment algorithm	Method used to align spikes	align by peak
Validation	Min. spike width	Minimum allowable spike width	80 μsec
	Max. spike width	Maximum allowable spike width	1500 μsec
	Max spike amplitude	Maximum allowable peak-to-peak amplitude	500 μV
	Min. spike slope	Minimum average absolute spike slope	2-5 μV/samp.
	Dead time	Detection pause following validation	0.5-1 msec
Sorting	Projection type	Method to project spike snippets into feature space	PCA
	Projection dimension	Dimensionality of feature space projection	2
	Max. K	Maximal number of units per recording channel	4
	P-value	P-value to consider a classified spike an outlier	0.01

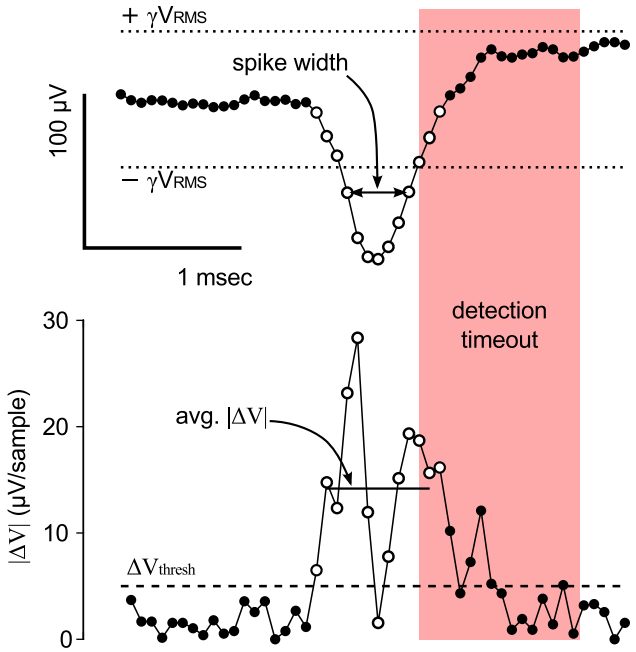

FIG. S4: Spike detection and validation in NeuroRighter is a multi-step process. Spikes are detected as voltage events that exceed the estimated RMS noise level for a given channel (Top). Putative spikes are then validated using *ad hoc* tests based on the spike width and rectified slope (Bottom).

B. Spike sorting

NeuroRighter uses an automated Gaussian mixture modeling algorithm to classify spikes based on low-dimensional features of their waveform shape (Xu and Wunsch, 2009). The algorithm is implemented using the Aforge.NET and Accord.NET libraries for machine learning[6]. The following explanation of the spike sorting algorithm applies to a single recording channel. During an actual recording, spike sorting is performed for all recording electrodes that have been enabled in the hardware settings GUI.

First, a training data set is collected. This consists of a set of spike snippets, $\{X_n\} \in \mathbb{R}^D$ where D is the number of voltage samples in each waveform. After a user-defined number of spikes have been collected, the classifier can be trained. To do this, waveforms are first projected into a low dimensional feature space, $\{Y_n\}_{n=1}^N \in \mathbb{R}^M$, $M < D$ using one of several available methods (principle component analysis, take the

[6] `http://accord-net.origo.ethz.ch/` and `http://code.google.com/p/aforge/`

peak voltage value, or take the peak voltage and and after-polarization amplitude). Next, a mixture of K Gaussians is fit to $\{Y_n\}_{n=1}^N$ using the expectation maximization algorithm (EM) (Dempster and Laird, 1977), initialized using K-means clustering. Following EM convergence, the minimum description length (MDL) is calculated as

$$\mathrm{MDL}(K) = \frac{1}{2} N_p \log(N * M) - \log p_y(y|K, \theta), \tag{2}$$

where N_p is the number of free parameters in θ and $\log p_y(y|K, \theta)$ is the maximized log-likelihood of the training set $\{Y_n\}_{n=1}^N$ according to the Gaussian mixture parametrized by θ. MDL therefore weighs model complexity against the goodness-of-fit of a given mixture (Xu and Wunsch, 2009). The value of K is iteratively decremented, and the model fitting and calculation of the MDL proceeds for each decreased model order. The value of K, and corresponding mixture, that minimizes the MDL is selected for online classification. The starting value of K is user-settable.

Following training, spikes are classified online. The unit number of each projected datum is defined as,

$$\kappa_n = \arg\max_K p_y(y_n|K, \theta). \tag{3}$$

The Mahalanobis distance between the classified datum, y_n and its putative component distribution, $f_\kappa(y)$, is then calculated as

$$d_n = [(y_n - \mu_\kappa)^\intercal C_\kappa (y_n - \mu_\kappa)]^{1/2}. \tag{4}$$

Here, (μ_κ, C_κ) are the (mean, covariance) pair for $f_\kappa(y)$. A Pearson's χ^2-test is performed to detect outliers in comparison with a χ^2 distribution fit to the Mahalanobis distances derived from the training data (Filzmoser, 2004). Outliers remain unsorted (they are assigned to $\kappa_n = 0$).

IV. EXPERIMENTAL METHODS

A. Tissue culture

Our culturing methods are described and demonstrated elsewhere (Hales et al., 2010) and here we provide a breif overview. MEAs (59 electrode + common ground, 200 uM electrode spacing, 30 um electrode diameter, titanium-nitride conductor, with silicon nitride insulation) were obtained from Multichannel Systems (Reutlingen, Germany). MEAs were sterilized using 70% ethanol and exposure to UV light, and coated with polyethyleneimine and laminin to promote surface hydrophilicity and cell adhesion, respectively. All dissections were carried out in accordance with the National Research Council's Guide for the care and use of laboratory animals using a protocol approved by the Georgia Tech IACUC. Whole neocortex was isolated from E18 rats under sterile conditions and stored in Hibernate-E medium (Invitrogen, Carlsbad, California, USA) for up to two hours before plating. The cortical tissue was digested in a solution containing 20 U$\cdot$ml^{-1} papain (Sigma-aldrich, St. Louis, Missouri, USA) in a culturing medium described in (Hales et al., 2010; Potter and DeMarse, 2001) without antibiotics or antimycotics. Cells were dissociated mechanically using 3-10 trituration passes through a 1 mL conical pipette tip. The cell suspension was diluted to 2500 cells/μL in culturing medium. Cells were centrifuged and strained to remove small and large debris. Fifty thousand cells in a 20 uL drop were plated at onto a 2 mm diameter area on precoated MEAs, which results in 2500 cells/mm^2 on the culturing surface (Wagenaar et al., 2006b). The culturing well of each MEA was sealed with a fluorinated ethylene-propylene (FEP) membrane (Potter and DeMarse, 2001) to prevent infection and changes in osmolarity due to evaporation (fig.S5). After a 30 minute adhesion period, culturing wells were flooded with 1 mL culturing media, adapted from (Jimbo et al., 1999), but without antibiotics. 0.75 mL of fresh culturing medium was exchanged every 3 days.

B. In-vitro MEA electrophysiology

All experiments and culture storage were carried out in an incubator regulated to 35°C, 5% CO_2, 65% relative humidity, which is a safe for MEA recording and stimulation electronics. Electrode voltages were

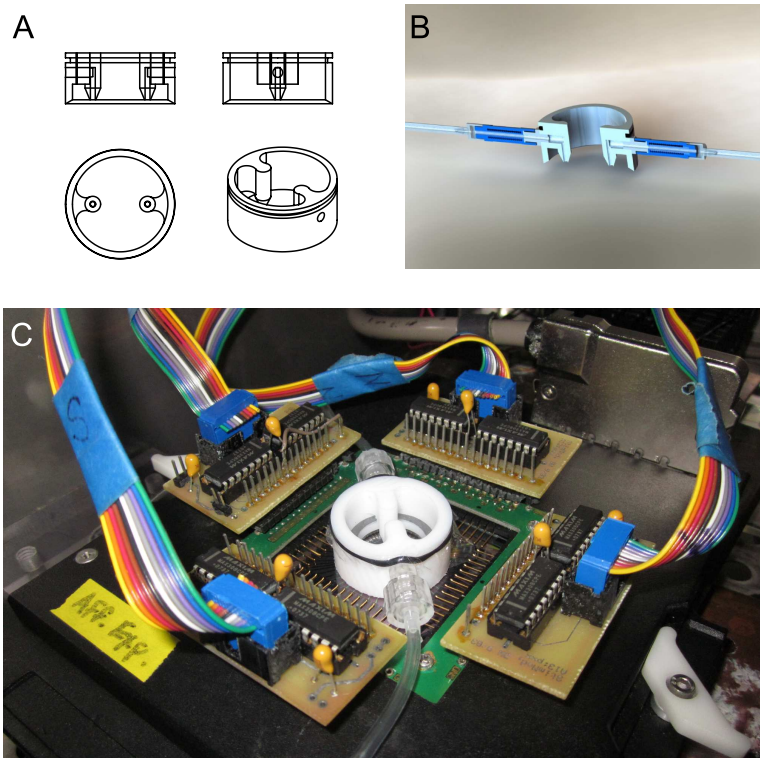

FIG. S5: Gas permeable perfusion system for bath application of drugs during multichannel recording and stimulation, *in-vitro*. **(A)** Drawing of perfusion cap design. The cap is machined from Teflon and fits tightly over the planar MEA's culturing well. The groove at the top of the cap holds a sheet of gas-permeable, water- and contaminant-impermeable FEP in place over the culturing well using an o-ring. FEP allows gas exchange between the incubator's regulated atmosphere and the culturing well, while preventing contamination and evaporation, which can lead to harmful changes in osmolarity and pH (Potter and DeMarse, 2001). Two perfusion ports allow input and output of culturing medium via a syringe pump. **(B)** Three dimensional rendering of the cap showing needle-less septa interfacing with the cap using Luer taper connectors. This allows the removal of input and output tubing without exposing the culturing well to outside air. **(C)** Photograph of a functioning perfusion lid, integrated with an MEA1060-Up amplifier (black) and stimulus multiplexing boards (brown with multicolored ribbon cables) (Wagenaar and Potter, 2004)

amplified 1200X and bandpass filtered between 10 Hz and 10 kHz using a 60 channel analog amplifier (MEA1060-Up; Fig. S5(c)). All MEA recording and stimulation were performed using the NeuroRighter multichannel electrophysiology platform[7]. Within NeuroRighter, amplified electrode voltages were digitally filtered using a 3rd order Butterworth design with a passband of 300 to 5000 Hz. Spike events were detected as events exceeding 5 times V_{RMS} noise on a given channel. Spike waveforms were collected as 2 millisecond snippets about the peak voltage inflection following a threshold crossing and validated by the slope of the waveform. A 1 millisecond detector pause following the peak of each waveform during which no spiking events can be detected was enforced (Fig. S4). Spike waveforms were sorted according to the methods described in section III.B and outliers were rejected using a p-value of 0.005.

Multichannel stimulation was delivered using the NeuroRighter stimulus generation boards along with 4 stimulation multiplexing boards to route electrical pulses to any of the 59 recording electrodes (Wagenaar and Potter, 2004). Stimulation multiplexer boards are shown in figure S5(c).

For experiments involving perfusion of d(-)-2-amino-5-phosphonopentanoic acid (AP5), 50 μM dilution of AP5 in culturing media was used. NeuroRighter triggered perfusion of the AP5 solution to the culture

[7] `https://sites.google.com/site/neurorighter/`

through a custom, gas-permeable perfusion cap via a kdScientific (Holliston, MA, USA) model 780262 syringe pump running at 1 mL/minute for 5 minutes (Fig. S5). AP5 was washed from the culture in the same way, using 10 mL of normal culturing media at 1 mL/minute. The perfusion cap and attached FEP membrane were autoclaved prior to use.

C. Animal surgery, recordings, and multielectrode stimulation

All animal procedures were conducted in accordance with the National Institutes of Health Guide for the Care and Use of Laboratory Animals and approved by the Emory University Institutional Animal Care and Use Committee. A 300-gram male Sprague-Dawley rat was anaesthetized with 1.5-3% inhaled isoflurane. A craniotomy was made over the right dorsal hippocampus. Seven 69 nL injections of tetanus toxin (concentration 50 ng/μL) were made into the right dorsal hippocampus at coordinates -3.3 AP, - 3.2 ML and -3.1 DV over 4 minutes. A microelectrode array (MEA) with 16 electrodes (each electrode with 33 μm diameter; Tucker-Davis Technologies) was implanted with 8 electrodes targeted at the CA1 and 8 electrodes targeted at the CA3 cell layers (Fig. 8(b)). The microelectrode array had a row separation of 1 mm and the electrodes within each row were separated by 175 μm. 5 smaller craniotomies were made for skull screws. The reference for microelectrode electrode array recording was tied to the skull screw over the cerebellum and the ground was tied to the remaining skull screws. Single unit recording performed during the implantation process guided the final positioning of the microelectrode array. Dental acrylic was used to seal the craniotomy and secure the MEA. The rat was allowed to rest for 6 days before electrical recording and stimulation experiments began.

All recording and stimulation was performed using the NeuroRighter multichannel electrophysiology platform. A recording head-stage from Triangle BioSystems (Durham, NC, USA) was used for signal amplification (100 X) and impedance matching. A custom stimulator board with a 1:16 multiplexer was used for sending electrical stimulation to a selected the electrode (design available on the NeuroRighter website[8]). By cascading the stimulus multiplexer board and the recording head-stage, electrical recording and stimulation could be performed simultaneously. Electrode voltages were digitally filtered using two 1st order Butterworth filters to generate two recording streams: 1-500 Hz for LFP and 500-5000 Hz for single and multiunit activity.

[8] `https://sites.google.com/site/neurorighter/download-neurorighter-pcbs`

V. LATENCY MEASUREMENT PLUGIN CODE

Below we provide the code to produce the three NeuroRighter plugins used to measure NeuroRighter's closed-loop response latency in section III.A of the main text. Each example is a C♯ class file that is derived from the `NRTask` base class. After making the proper API references and inheriting methods from `NRTask`, calls can be made to NeuroRighter's input and output servers. Each of these classes follows one of the basic structures outlined in code listing II.B.2 of the main text and was created using the steps detailed in section II of this document. Aside from these examples, the code used to produce all case-studies presented in the main text (with the exception of the Silent Barrage robotic embodiment) are available on NeuroRighter's code repository[9].

A. StimSrv-based real-time loop

```
/// <summary>
/// This class uses StimSrv to produce a series of 32-bit digital pulses in
/// response to spikes produced by two, pre-specified units. Each pulse encodes
/// the unit number that produced the spike and the time (32-bit sample integer)
/// that the spike occurred. This method uses double buffering and therefore
/// results in a large reaction latency.
/// </summary>

// References
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NeuroRighter.NeuroRighterTask;
using NeuroRighter.DataTypes;
using NeuroRighter.Server;
using NeuroRighter.Network;

namespace NR_CL_Examples
{
    class StimSrv_UnitReaction : NRTask
    {
        // Unit that we will react to with a digital pulse
        private int[] units = { 1, 2 };

        // Internal variables
        private ulong lastSampleRead = 0;
        protected EventBuffer<SpikeEvent> newSpikes;
        ulong nextAvailableSample;

        // When using StimSrv, most setup tasks are taken care of automatically.
        protected override void Setup()
        {
            nextAvailableSample = 0;
        }

        // Loop is called periodically by the double buffering system when the
        // read-head has exhausted all the samples in one of the buffers and
        // that buffer is made available for writing.
```

[9] http://code.google.com/p/neurorighter/source/browse/NR-ClosedLoop-Examples/

```
protected override void Loop(object sender, EventArgs e)
{
    // First, figure out what history of spikes we have
    ulong[] spikeTimeRange =
            NRDataSrv.SpikeSrv.EstimateAvailableTimeRange();

    // Is there any new data yet?
    if (spikeTimeRange[1] > lastSampleRead)
    {
        // Try to get the number of spikes within the available time range
        newSpikes =
                NRDataSrv.SpikeSrv.ReadFromBuffer(lastSampleRead, spikeTimeRange[1]);

        // Update the last sample read
        lastSampleRead = spikeTimeRange[1];
    }
    else
    {
        return;
    }

    // Is one of my units in here?
    List<SpikeEvent> unitGSpikes =
            newSpikes.Buffer.Where(x => x.Unit == units[0]).ToList();
    List<SpikeEvent> unitTSpikes =
            newSpikes.Buffer.Where(x => x.Unit == units[1]).ToList();

    // Get the current buffer sample and make sure that we are going
    // to produce stimuli that are in the future
    ulong currentLoad =
            NRStimSrv.StimOut.GetNumberBuffLoadsCompleted() + 1;
    nextAvailableSample =
            currentLoad * (ulong)NRStimSrv.GetBuffSize();

    // Create the output buffer
    List<DigitalOutEvent> DigitalOutBuffer =
            new List<DigitalOutEvent>();

    for (int i = 0; i < unitGSpikes.Count; i++)
    {
        // Use the native digital output server to send digital change
        DigitalOutBuffer.Add(
                new DigitalOutEvent(nextAvailableSample, 71));
        SpikeEvent sG = unitGSpikes[0];
        DigitalOutBuffer.Add(
                new DigitalOutEvent(nextAvailableSample + 10,
                                    (uint)sG.SampleIndex));
    }
    for (int i = 0; i < unitTSpikes.Count; i++)
    {
        // Use the native digital output server to send digital change
        DigitalOutBuffer.Add
                (new DigitalOutEvent(nextAvailableSample, 84));
        SpikeEvent sT = unitTSpikes[0];
        DigitalOutBuffer.Add
```

```
                        (new DigitalOutEvent(nextAvailableSample+10,
                                                                (uint)sT.SampleIndex));
            }

            if (DigitalOutBuffer.Count > 0)
                NRStimSrv.DigitalOut.WriteToBuffer(DigitalOutBuffer);
        }

        // Shutdown StimSrv etc.
        protected override void Cleanup()
        {
            Console.WriteLine("Terminating protocol...");
        }
    }
}
```

B. NewData-based real-time loop

```
/// <summary>
/// This class produces a series of 32-bit digital pulses in response to
/// spikes produced by two, pre-specified units. Each pulse encodes the unit
/// number that produced the spike and the time (32-bit sample integer)
/// that the spike occurred.
/// </summary>

// References
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NeuroRighter.NeuroRighterTask;
using NeuroRighter.DataTypes;
using NeuroRighter.Server;
using NeuroRighter.Log;
using NationalInstruments.DAQmx;
using NationalInstruments;
using System.Windows.Forms;
using System.ComponentModel;
using MoreLinq;

namespace NewDataEventCatcher
{
    class NewData_UnitReactionDO : NRTask
    {
        // Unit that we will react to with a digital pulse
        int[] units = { 1, 2 };

        // Internal variables
        int numberOfSpikesReactedTo = 0;
        uint spkSamp; // The sample of the latest spike detection

        // NI Stuff
        Task DOTask;
        DigitalSingleChannelWriter DOWriter;

        // Take the first 32-bit port
        string DOChannel1 = "/Dev1/Port0/line0:31";

        // The sample frequency of the output channel
        double DOSampleFreqHz = 10000.0;

        protected override void Setup()
        {
            // Subscribe to the NewData event on the spikes input server
            NRDataSrv.SpikeSrv.NewData +=
                    new EventDataSrv<NeuroRighter.DataTypes.SpikeEvent>.
                    NewDataHandler(SpikeSrv_NewData);

            // Setup an digital on demand output line. Make sure that you don't
            // reserve this line through the Hardware Settings GUI in
            // NeuroRighter for digital input or output. Also, make sure that
```

```
    // double-buffering is disabled.
    try
    {
        // Create and configure tasks. This is unbuffered, on-demand
        // output to reduce response latency.
        DOTask = new Task("DOTask");
        DOTask.DOChannels.CreateChannel(
                DOChannel1,
                "DOChannel1",
                ChannelLineGrouping.OneChannelForAllLines);
        DOTask.Timing.SampleTimingType = SampleTimingType.OnDemand;

        // Verify tasks and reserve the port and clock lines
        DOTask.Control(TaskAction.Verify);
        DOTask.Control(TaskAction.Reserve);

        // Create and configure writers
        DOWriter = new DigitalSingleChannelWriter(DOTask.Stream);

        // Clear the port
        DOWriter.WriteSingleSamplePort(true, 0);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// There is no loop function needed when doing closed loops based on
// NewData events. In this case, the NewData Event will trigger the
// NewData Event Handler to react with whatever code you want.
// Here is the NewData event handler:
private void SpikeSrv_NewData
        (object sender, NewEventDataEventArgs<SpikeEvent> eArgs)
{
    // Is my unit in here? (eArgs automatically contains the new data on
    // the spike server. No read is required.)
    List<SpikeEvent> unit1Spikes =
            eArgs.NewDataBuffer.Buffer.Where(x => x.Unit == units[0]).ToList();
    List<SpikeEvent> unit2Spikes =
            eArgs.NewDataBuffer.Buffer.Where(x => x.Unit == units[1]).ToList();

    // In the case that the first unit spiked
    if (unit1Spikes.Count > 0 && unit2Spikes.Count == 0)
    {
        // Write a digital pulse representing the detected unit and
        // the time it occurred
        spkSamp =
                (uint)unit1Spikes.MaxBy(x => x.SampleIndex).SampleIndex;
        WriteDO(units[0]);
        numberOfSpikesReactedTo++;
    }
    // In the case that the second unit spiked
    if (unit1Spikes.Count == 0 && unit2Spikes.Count > 0)
    {
```

```
                // Write a digital pulse representing the detected unit and
                // the time it occurred
                spkSamp =
                        (uint)unit2Spikes.MaxBy(x => x.SampleIndex).SampleIndex;
                WriteDO(units[1]);
                numberOfSpikesReactedTo++;
            }

            // In the case that both spiked in the new data buffer
            if (unit1Spikes.Count > 0 && unit2Spikes.Count > 0)
            {
                // Write a digital pulse representing the detected unit and
                // the time it occurred
                uint spkSamp1 =
                        (uint)unit1Spikes.MaxBy(x => x.SampleIndex).SampleIndex;
                uint spkSamp2 =
                        (uint)unit2Spikes.MaxBy(x => x.SampleIndex).SampleIndex;
                uint[] spkSamps = {spkSamp1,spkSamp2};
                spkSamp = spkSamps.Max();
                WriteDO(units[0] + units[1]);
                numberOfSpikesReactedTo++;
            }
        }

        // This method interacts with the NI Card to produce digital pulses
        public void WriteDO(int whichUnit)
        {
            // Write a sample that says which unit fired
            DOWriter.WriteSingleSamplePort(true, whichUnit);

            // Write a sample that says when the unit fired
            DOWriter.WriteSingleSamplePort(true, spkSamp);

            // Clear the port
            DOWriter.WriteSingleSamplePort(true, 0);
        }

        // Dispose the National Instruments virtual channel objects
        protected override void Cleanup()
        {
            DOTask.Stop();
            DOTask.Dispose();
        }
    }
}
```

C. Arduino-based real-time loop

```
/// <summary>
/// This class uses an RS232 serial communication protocol to generate digital
/// pulses using an Arduino micro-controller board in response to spikes produced
/// by particular units. The ascii_response.ino or digital_reaction.ino script
/// must be running on the Arduino prior to executing this protocol.
/// </summary>

// References
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NeuroRighter.NeuroRighterTask;
using NeuroRighter.DataTypes;
using NeuroRighter.Server;
using System.IO.Ports;

namespace NR_CL_Examples
{
    class Arduino_UnitReaction : NRTask
    {
        // Unit that we will react to with a digital pulse
        private int[] units = { 1, 2 };

        // I/O variables
        private ulong lastSampleRead = 0;
        protected EventBuffer<SpikeEvent> newSpikes;

        // Serial Port
        private SerialPort serialPort1;

        // Initialize the plugin
        protected override void Setup()
        {
            try
            {
                // Serial port object and properties (RS232)
                System.ComponentModel.IContainer components
                    = new System.ComponentModel.Container();
                serialPort1 = new SerialPort(components);
                serialPort1.PortName = "COM3";
                serialPort1.BaudRate = 9600;
                serialPort1.Open();

                if (!serialPort1.IsOpen)
                {
                    Console.WriteLine("Failed to connect to device.");
                    return;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
```

```
        }

        // Turns on the serial port
        serialPort1.DtrEnable = true;

        // Callback for text coming back from the Arduino
        serialPort1.DataReceived += OnReceived;

        // Give it 2 secs to start up the sketch
        System.Threading.Thread.Sleep(2000);

        Console.WriteLine("Serial-communication established.");
    }

    protected override void Loop(object sender, EventArgs e)
    {
        // First, figure out what history of spikes we have
        ulong[] spikeTimeRange =
        NRDataSrv.SpikeSrv.EstimateAvailableTimeRange();

        // Is there any new data yet?
        if (spikeTimeRange[1] > lastSampleRead)
        {
            // Get the number of spikes within the available time range
            newSpikes =
                NRDataSrv.
                SpikeSrv.
                ReadFromBuffer(lastSampleRead,spikeTimeRange[1]);

            // Update the last sample read
            lastSampleRead = spikeTimeRange[1];
        }
        else
        {
            return;
        }

        // Is my unit in here?
        List<SpikeEvent> unitGSpikes =
                newSpikes.Buffer.Where(x => x.Unit == units[0]).ToList();
        List<SpikeEvent> unitTSpikes =
                newSpikes.Buffer.Where(x => x.Unit == units[1]).ToList();

        for (int i = 0; i < unitGSpikes.Count; i++)
        {
            // Use the serial port to send a command to the Arduino
            serialPort1.Write(new byte[] { 1 }, 0, 1);
        }
        for (int i = 0; i < unitTSpikes.Count; i++)
        {
            // Use the serial port to send a command to the Arduino
            serialPort1.Write(new byte[] { 2 }, 0, 1);
        }
    }
```

```
        // Callback from Arduino
        private void OnReceived(object sender, SerialDataReceivedEventArgs c)
        {
            try
            {
                // write out text coming back from the Arduino
                Console.Write(serialPort1.ReadExisting());
            }
            catch (Exception exc)
            {
                Console.Write(exc.Message);
            }
        }

        // Shut down the serial port on protocol termination
        protected override void Cleanup()
        {
            serialPort1.Close();
        }
    }
}
```

CITED IN SUPPLEMENTARY MATERIAL

Dempster, A. P. and Laird, N. M. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society.*, 39(1):1–38.

Filzmoser, P. (2004). A multivariate outlier detection method. In *Proc. of the Seventh International Conference on Computer Data Analysis and Modeling*, number 1989, pages 18–22.

Hales, C. M., Rolston, J. D., and Potter, S. M. (2010). How to culture, record and stimulate neuronal networks on micro-electrode arrays. *J. Visualized Exp.*, (39).

Jimbo, Y., Tateno, T., and Robinson, H. P. (1999). Simultaneous induction of pathway-specific potentiation and depression in networks of cortical neurons. *Biophys. J.*, 76(2):670–8.

Potter, S. M. and DeMarse, T. B. (2001). A new approach to neural cell culture for long-term studies. *J. Neurosci. Methods*, 110(1-2):17–24.

Wagenaar, D., DeMarse, T. B., and Potter, S. M. (2006a). MEABench: A toolset for multi-electrode data acquisition and on-line analysis. In *Proc. 2nd Int. IEEE EMBS Conf. on Neural Eng.*, pages 518–521.

Wagenaar, D. A., Pine, J., and Potter, S. M. (2006b). An extremely rich repertoire of bursting patterns during the development of cortical cultures. *BMC Neurosci.*, 7(11).

Wagenaar, D. A. and Potter, S. M. (2002). Real-time multi-channel stimulus artifact suppression by local curve fitting. *J. Neurosci. Methods*, 120:113–120.

Wagenaar, D. A. and Potter, S. M. (2004). A versatile all-channel stimulator for electrode arrays, with real-time control. *J. Neural. Eng.*, 1(1):39–45.

Xu, R. and Wunsch, D. C. (2009). *Clustering*. John Wiley and Sons, Inc., Hoboken, N. J.