Closing the loop in the HiDens system

Benjamin Naecker

July 2, 2018

Purpose This document is intended to sketch the outlines of a closed-loop experiment in the HiDens array system. Here, "closed-loop" means an experiment in which data is collected, a new electrode configuration is chosen and set, and more data is collected. It assumes familiarity with the rest of the recording software system, as well as familiarity with the tools used to generate and set chip configurations.

Introduction

One of the major advantages of the high-density CMOS MEA systems acquired from our collaborator Andreas Hierlemann is the ability to dynamically connect up to 126 of the chips more than 11K electrodes. We can generate almost any configuration of actual recording electrodes we desire, spread in any fashion over the ≈ 2.5mm chip surface. Here, I'll describe how one might write a closed-loop expriment from scratch, using the existing HiDens tools as well as the current recording software system.

Generating and setting chip configurations

Chip configurations are generated using the exectuable `neuro-dish-router`, which is part of the HiDens software suite. It is currently compiled and installed on `granite.stanford.edu`, the main HiDens recording computer. The program is usually run as a GUI; one clicks on desired electrodes on a model of the chip, and the program uses that as the starting point for generating a configuration.

Specifying configurations with text files

However, the program can also be run from the command line, without a graphical interface. In this case, the program uses so-called "Neuro Placement" files (their term), which have extension `.neuropos.nrk`. These are text files which define "interesting" locations (positions). The full format is detailed on this internal ETH Wiki page (requires login).

Each line in the file gives a location to consider. These lines have the format:

```
[*]Neuron NAME: X/Y, SX/SY[, srSR][, cCOST][, elCNT][, stim]
```

Parameters in brackets (`[]`) are optional.

`NAME` is an arbitrary string to assign to this location. `X` and `Y` are the position in microns. These need not be actual positions of any electrode, but must be in the range of the size of the chip, i.e., $0 <= X <= 2.5$mm. `SX,Y` are the width and height of the location. They currently need to be the same.

`SR` refers to the sampling rate, which should be ignored.

The routing system solves an optimization problem, trying to minimize some cost. The `COST` parameter can be used to give certain electrodes larger cost than others, meaning they can be used but are less favorable. In general, I would avoid specifying this, letting each location have the same cost.

The parameter `CNT` refers to the number of electrodes that should be given to this position. This becomes relevant because you can specify multiple lines as part of the same unit, the same "neuron". These must have the same `NAME`, and each line in the neuron must start with an asterisk ($*$).

`stim` refers to whether the electrode is used for extracellular stimulation. We never use this capability in the lab, so I would ignore it.

A few example lines from a valid file might look like this:

```
Neuron click3: 482/298, 20/20
Neuron click4: 325/249, 50/50
Neuron click5: 475/510, 50/50, c17
Neuron click6: 605/200, 50/50
```

Creating and sending configurations

Once you have these files, you can pass them to `neuro-dish-router` on the command line with:

```
$ neuro-dish-router -n -s output-dir -l my-file.neuropos.nrk
```

The `-n` flag tells the program not to start the GUI, but to run in headless mode. The output will be placed into `output-dir`.

The output files will be in a "hex" format, ending with `.hex.nrk2`. These must be converted to a format that can be sent directly to the HiDens CMOS chip. You can do that with:

```
$ bit-streamer -n -f my-file.hex.nrk2
```

This will generate the file `my-file.cmdraw.nrk2`, which *can* be directly sent to the chip with:

```
$ nc 11.0.0.7 32126 < my-file.cmdraw.nrk2
```

Outline of a Python module for reading/writing configurations

There is currently no tool for programmatically reading or writing configuration files. One will be required for implementing this closed loop system. I would recommend creating one in Python, which makes manipulation of text files a cinch, and also allows easily running external programs (such as `neuro-dish-router`).

The package will need to do three things:

- Read, write, and generate desired neuropos files.
- Run `neuro-dish-router` and `bit-streamer` to generate actual configurations.
- Send the configuration to the chip.

To generate the text files, I'd recommend creating an abstraction of a *location*, which includes a name, location, size, and possibly a cost. Then you could make a *neuron*, which is one or more locations. Then reading and writing the neuropos files should be just spitting each location onto a new line, making sure that multi-location neurons have the same name and start with an asterisk.

Generating the configurations and sending them to the chip are easy. Use the `subprocess` module from Python's standard library to run each of the required executables in another process. Something like this.

```
import subprocess as sp

# Somehow generate the desired configuration. Could be from user input,
# from a computation, whatever.
config = get_desired_configuration()

# Generate a true configuration from the desired
neuropos_file = make_neuropos_file(config)
hex_file = neurpos_file.replace('neuropos.nrk', 'hex.nrk2')
proc = sp.run(['neuro-dish-router', '-n', '-s', 'output-dir', '-f', neuropos_file])

# Convert the hex file to a ``raw'' file
proc = sp.run(['bit-streamer', '-n', '-f', hex_file])

# Send to the chip
raw_file = hex_file.replace('hex', 'cmdraw')
proc = sp.run(['nc', '11.0.0.7', '32126', '<', raw_file])
```

Outline of a closed-loop experiment

Here is the outline of a fully closed-loop experiment in the HiDens system.

```
import bldsclient

# This method would generate a baseline starting configuration and
# send it to the chip. For example, this might be a random configuration,
# or the first of a sequence of high-density blocks.
config = generate_start_config()
raw_file = generate_raw_from_neuropos(config)
send_configuration(raw_file)

# Connect to the BLDS to collect data.
client = bldsclient.BldsClient()
client.connect()

# We might collect 5 minutes of data at each configuration
duration = float(60 * 5)
client.set('recording-length', duration)

# The main loop
while more_configs():

  # Collect all 5 minutes of data from the server.
  client.start_stream()
  data = get_all_data(client)
  client.stop_stream()

  # Compute a new configuration based on this
  config = compute_new_configuration(config, duration)

  # Generate the nearest possible configuration and send to the chip
  raw_file = generate_raw_from_neuropos(config)
  send_configuration(raw_file)
```

This is very rough, but it should give you an idea on how to get started.