LSCE Project Documentation

Brought to you by:

Pedro Rittner

Detian Shi

Nicholas Beaumont

Leina Sha

Danielle Lee

Anthony Chen

Welcome to the LSCE Project, We’ve Got Fun and Games!

**Data Importer**

The Data Importer module in this project is designed to import “.raw” data files created by Multichannel Systems’ “MC\_DataTool” as a series of numpy arrays representing time-continuous data from each electrode in the device. It can also import from legacy Matlab files.

For example, to load from 12 raw files located in the directory “/home/user/data” with sampling frequency rate of 20e4 and with prefix “slice3\_”:

Importer.loadFromRaw(“/home/user/data”, 12, “slice3\_”, 20e4)

Doing so will generate a folder called “slice2\_” with the processed “.npy” files inside. You can then call the data formatter on this output directory. There is also a option of saving to the legacy Matlab format by passing in “saveMat=True” in the parameter list.

Loading from legacy Matlab code can be done by calling the loadFromMat function. It takes as arguments the file directory, sampling frequency rate and the resampling frequency rate.

Note on writing further importers: due to memory requirements, it is probably better to write intermediate files to disk. An example can be seen in loadFromRaw. Always remember to clean up after yourself or you will soon out of hard drive space!

**Data Formatter**

The Data Formatter module is designed to be the most general tool in this project, and is designed to gather multiple extremely large numpy array dumps together in one HDF file such that they can be manipulated and viewed in a meaningful way. The Data Formatter.formatData method is the primary way to invoke this module, which when given a source folder, destination file, and optionally the name of a configuration file, will copy every numpy array dump (.npy file) in the source folder into a ‘raw\_data’ group inside of the destination hdf5-format file. The datasets created in the hdf file will have the same name as their source file, and come with two attributes by default, namely their data type and shape. Additional data can be specified both for individual datasets and for the raw\_data group as a whole by including a configuration file in the source directory. The configuration file should consist of one file whose name is specified in the function call (“config.ini” is checked for by default). The file should have the following format:

[raw\_data]

sampling\_rate = 60

time\_taken = 5pm, 12/20/2012

interesting\_times = 2:50, 10:32, 43:56

researcher\_comment = started with carbachol up top but with aCSF in tubes

[Electrode\_44\_master]

interesting\_times = 485, 1020, 1420

In the above file, [raw\_data] refers to data that will be included in the attributes of the raw\_data group within the created hdf5 file. [Electrode\_44\_master] in this case refers to a single file within the directory by the same name, and data assignments following that heading will be added to that dataset’s individual attributes.

**Data Visualization**

The Data Visualization module in this project is implemented specifically for an 8x8 arrangement of electrodes (8 rows of electrodes with 8 columns) with the corner electrodes missing. For future use, this implementation may be changed / extended by changing the constant variables in the corresponding datavisualization.py file. (See comments in file)

Implementation

The Data Visualization module displays the electrode data in two main views.

*View 1 - Viewing all electrode data concurrently*

View 1 is generated for a general viewing purpose. It will help the users identify patterns in the data for the electrodes as a whole and enable specific electrodes of interest to be identified.

View 1 graphs data for all electrodes concurrently. This view generates 60 graphs (8x8 with the corners missing) arranged in the way the electrodes were placed (ie electrode in column 1 row 1 would be graphed in column 1 row 1). Since the data is large and cannot fit on one graph, this view is scrollable. The users can stipulate what time window they want each graph to span (i.e. one view of the graph only views 3 seconds worth of data) and then scroll the graph to see data for later times. When the view is scrolled each graph for every individual electrode is scrolled concurrently. This is so that users can identify electrodes of interest by comparing their data to the overall electrode data. There are no axis labels or ticks on the graphs because this slows performance. In addition, this level of detail is not required for the purpose of assessing general trends and identifying electrodes of interest.

When you click on an individual graph in View 1, you will be brought to View 2, with the time window at precisely the time specified by the scroll bar in View 1

*View 2 - Viewing specific electrode data*

View 2 is generated so that users can view data for electrodes of interest. This view provides an in depth look at electrode data.

This view has axis labeled and ticks on the graph. It also has zoom functionality, movement functionality (enabling the focus of the graph to be moved) and saving functionality.

How To Use

The data visualization requires data in the form of 2D arrays. To achieve this, the data visualization should be used after the data importer and data formatter. Here is an example of how the different components should be used in conjunction:

Importer**.**loadFromRaw("E:\\LSCE\\110112") //this imports the data

DataFormatter**.**formatData("E:\\LSCE\\110112\\slice2\_", "fulldata") //this formats the data into hdf5 and saves it

  tmp **=** h5py**.**File("fulldata.hdf5", "r+") //opens the data file that was just created

  data **=** []

**for** dataset **in** tmp["raw\_data"]**.**keys(): //adds all data sets in hdf5 group to a list

       data**.**append(tmp["raw\_data"][dataset])

    datavisualization**.**analyze8x8data(data**=**data, samprate**=**1000, time**=**5) //data visualization function

    tmp**.**close()

Functions

Here are a list of relevant functions that can be executed (for detailed implementation see comments in file):

**analyze8x8data(data, time=1, samprate=2)**

Function which produces a visualization of 8x8 electrode data with a main view (graph of each electrode's data, arranged together according to the electrode positions) and zoom in view (graph of single electrode data).

Data = 2D Array of y values to be plotted

Time (in seconds) = the amount of time the graph should span in each window should be passed in as an integer.

Samprate = sampling rate, i.e how many data samples per second should be passed in as an integer

**analyze8x8Group(data, time=1, samprate=2)**

Function which produces a visualization of 8x8 electrode data with a main view (graph of each electrode's data, arranged together according to the electrode positions) and zoom in view (graph of single electrode data).

Data = 2D Array of y values to be plotted

Time (in seconds) = the amount of time the graph should span in each window should be passed in as an integer.

Samprate = sampling rate, i.e how many data samples per second should be passed in as an integer

**def analyzesingle(data, time, samprate)**

Function which produces visualization of single electrode data.

Data = Array of y values to be plotted

Time (in seconds) = the amount of time the graph should span in each window should be passed in as an integer.

Samprate = sampling rate, ie how many data samples per second should be passed in as an integer.