Electrophysiology Analysis Optimization using Intan Technologies

Documentation and Protocol presented
by

Baihan Lin

to
Olavarria Lab
Department of Psychology

in partial fulfillment of the data analysis task
in the subject of
Computational Neuroscience

University of Washington
Seattle, Washington, USA

Apr 2016

Mentors:
Prof. Jaime Olavarria
Dr. Adrian Andelin

Author:
Baihan Lin

Abstract:

The thesis is given birth due to the need to analyze a set of electrophysiology recording data and the curiosity of finding the best way to generate and present the result, and to simplify the entire electrophysiology analysis in a simple automated protocol. Comparing to the traditional method, which spends 17640 minutes (~300 hours) to analyze the 441 cases of data files in our current stage, my method only takes less than 10 minutes, which is 1764 times faster and saves 300 hours. My method also integrates various new features such as excel-friendly log and recursive file search. I hope this method described in this documentation can offer a more systematic and convenient way in the anlaysis of electrophysiology recording data. I look forward to further improvements if any. After tens of hours of my working each week to develop this automated protocol, I hope lab members in the future can simply run the program in 10 minutes to save their time and lives profoundly.

Comment:

This documentation is mainly for the purpose of recording my thoughts and attempts in optimizing the data analysis task, therefore not in a standard of submission in any kind.

Keyword:

MATLAB, protocol, electrophysiology experiments, Mac terminal, Intan Technologies

Contents:

Work Summary:

(What, when and how I approached the project)

I used MATLAB and GitHub to perform my project, because it seems previously I programmed a lot but never kept record for it. Working at night and other irregular hours, I found it sad that these attempts and efforts are sometimes untracked. Thus, I introduced this to ensure my attempts are tracked for my own record.

During the project, I reported weekly and communicated with Dr. Adrian Andelin in 7 emails. From GitHub, I made at least 63 commits (major adjustments of codes as shown in Figure 4), created 14 versions of analysis code for different purposes (Figure 5).

From the GitHub daily coding graph (Figure 1 and 2), I worked continuously for this new analysis project every week for tens of hours from April 13th 2016 when I was introduced to the traditional method.

From the GitHub coding punch card (Figure 3), it seems I coded for this project spread out the days, mostly at night and early morning. This might be odd but to me, coding at night is the most productive since I found it serene and focusing.

In summary, I put in considerable amount of effort and time into this project, and I really value the trust and responsibilities Prof. Olavarria and Dr. Adrian gave me. I sincerely hope my endeavor and effort do help facilitate the analysis in our lab!

Using my method, running the whole MATLAB analysis for all 441 cases can be accomplished within 10 minutes (normally 5 minutes is sufficient), with only one click and a wait for 10 minutes before viewing our products in specific folders. Comparing to the traditional method costing 17640 minutes, my method is 1764 times faster and saves almost 300 hours of work!

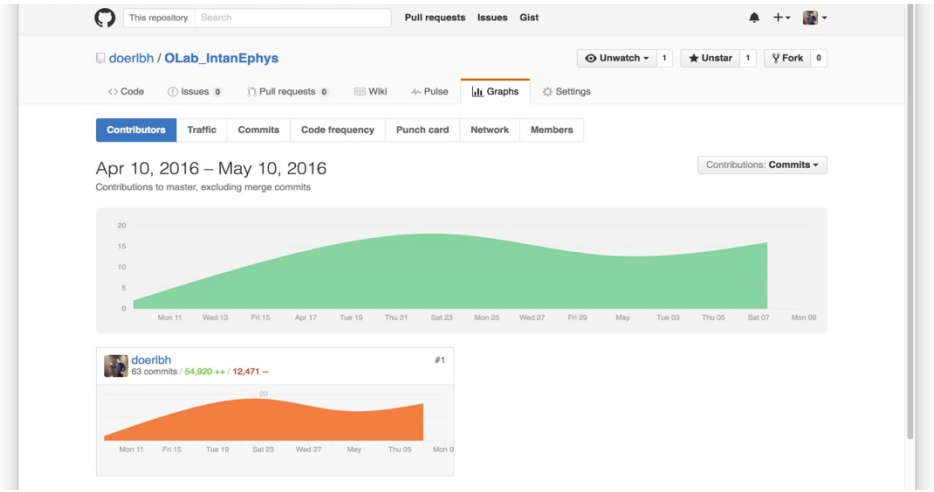


Figure 1. Contribution by days till May 10th 2016

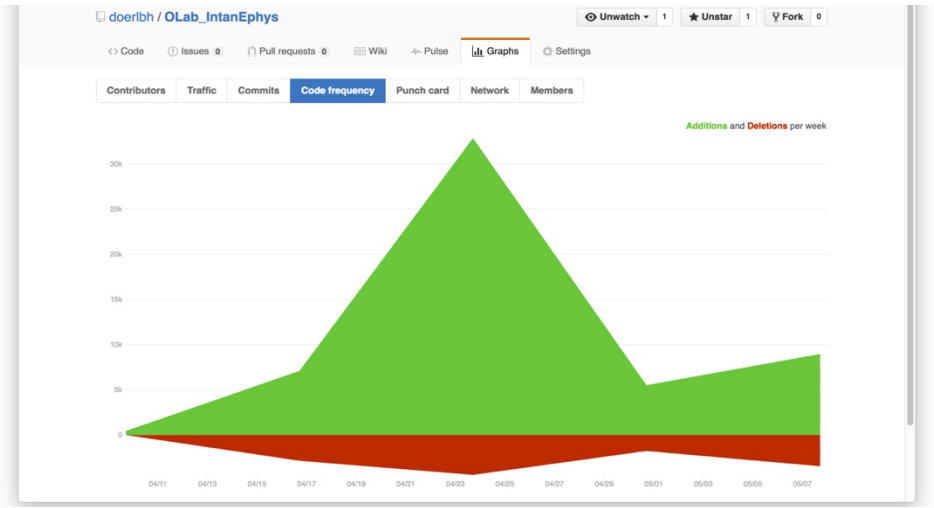


Figure 2. Code frequency with respect to addition and deletion till May 10th 2016

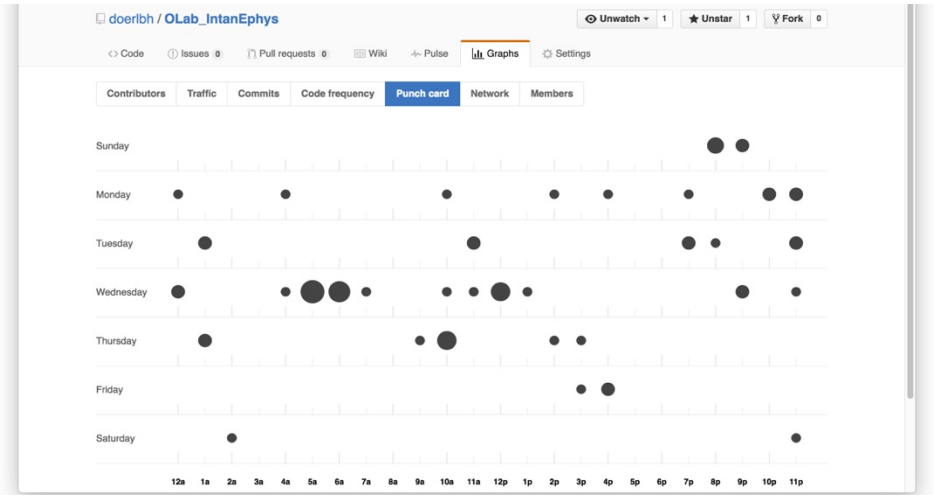


Figure 3. Punch card of coding time for this project till May 10th 2016

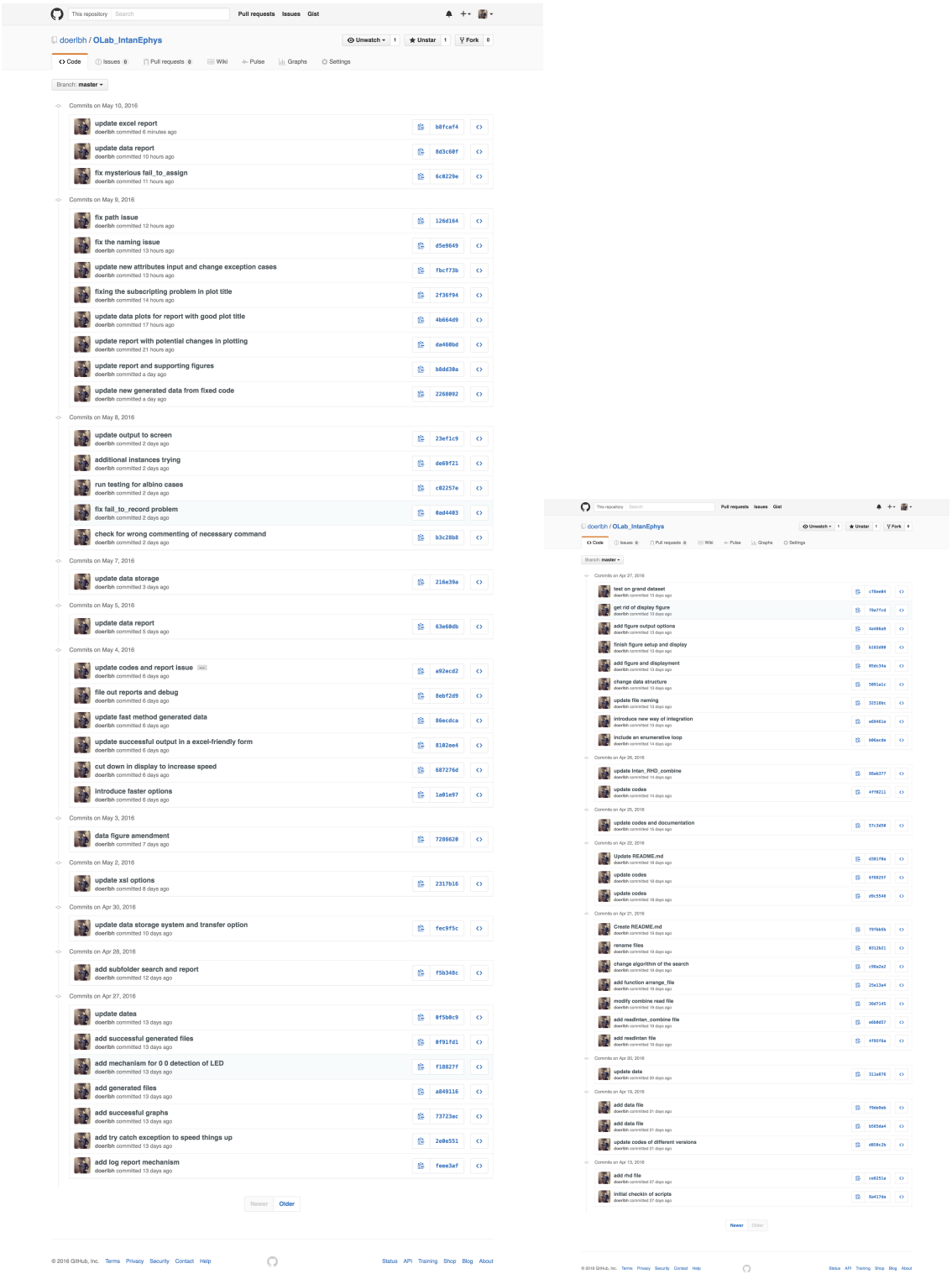


Figure 4. commits (major changes of codes) made by days till on May 10th, 2016

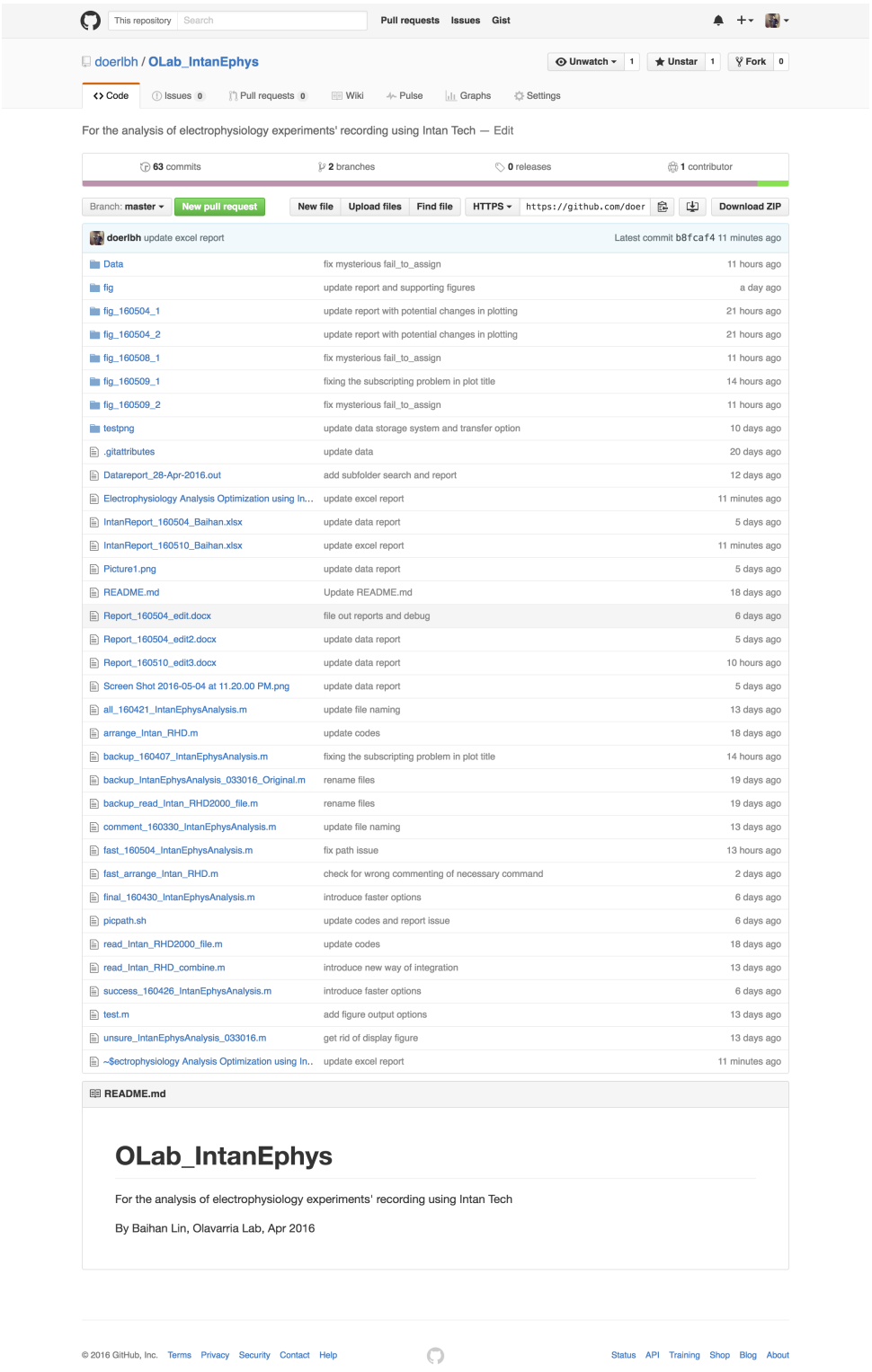


Figure 5. Screenshot of my GitHub repository on May 10th, 2016

Above is the screenshot of my GitHub repository of all the codes and my progress (Figure 5) and the link: https://github.com/doerlbh/OLab_IntanEphys/

Problem Identification:

(What to do)

To study the effect of sensory deprivation on brain development, Using a combination of transneuronal tracing, in situ hybridization for the immediate early gene Zif268 and electrophysiological recordings, our lab recently showed that the primary visual cortex (V1) in pigmented rats has ODCs, and these ODCs correlate with callosal inputs from the opposite hemispheres. Using similar methods, my project aims to understand the effect of monocular deprivation (MD) on the newly discovered system of ODCs in rat visual cortex. However, introducing a new system of electrophysiology, RHD2000 Amplifier Evaluation System by Intan Technologies, we need a new way of analyzing its corresponding data format.

The RHD2000 Amplifier Evaluation System is a modular family of open-source hardware and software that allows users to record biopotential signals from up to 256 low-noise amplifier channels using RHD2000 digital electrophysiology chips from Intan Technologies. As shown in Figure 6, A USB interface board connects to a host computer via a standard USB cable. Small amplifier boards connect to the interface board via thin, flexible all-digital cables that may be daisy-chained to form robust connections up to 10 meters in length.

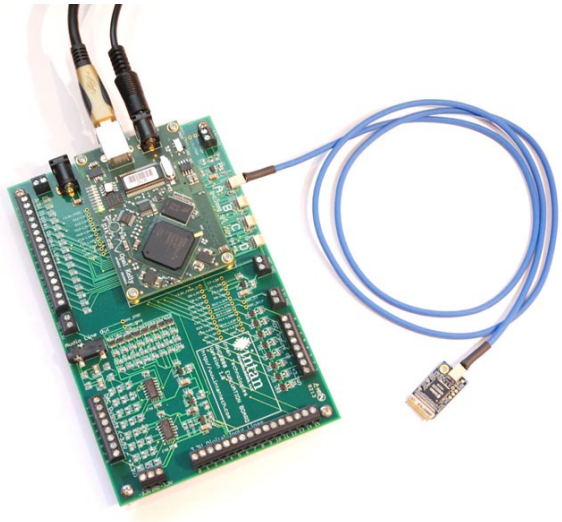


Figure 6, RHD2000 Amplifier Evaluation System

As shown in Figure 7, Open-source, multi-platform GUI software controls the operation of the amplifiers and streams data to the screen and to disk in real time at user-selected sampling rates from 1 kS/s to 30 kS/s.

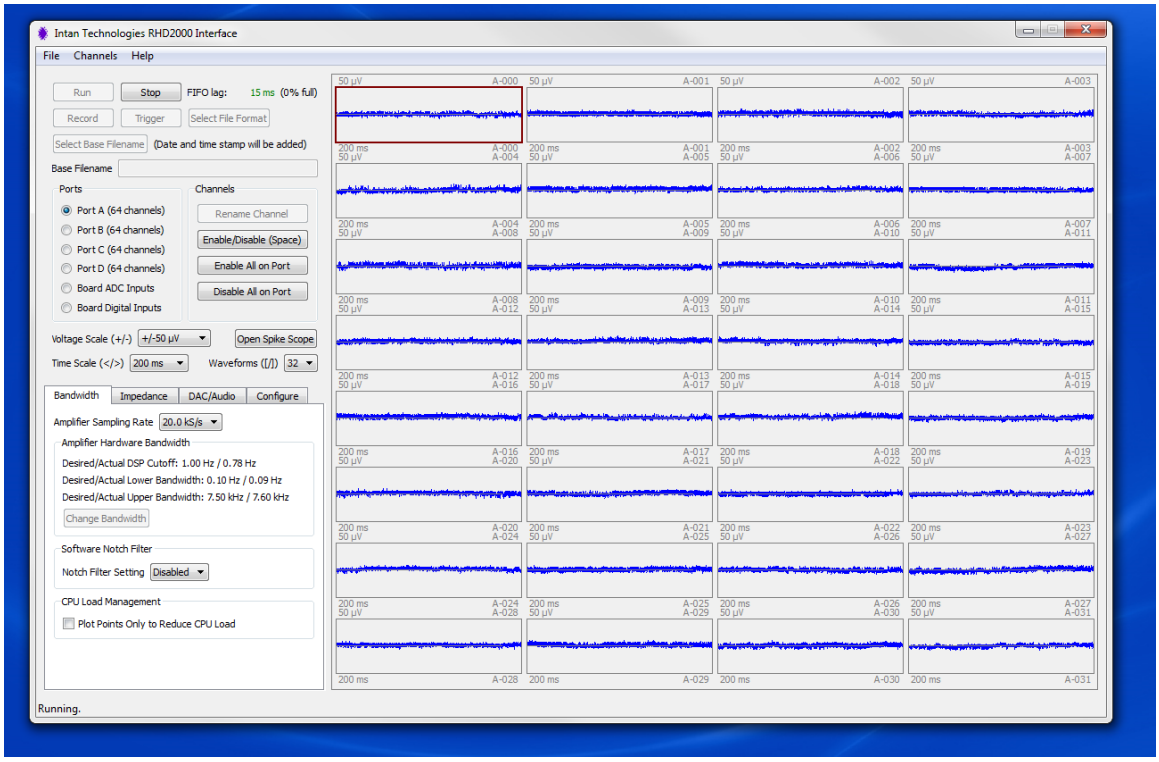


Figure 7, RHD2000 GUI interface

The RHD2000 Evaluation System allows users to perform the following functions:

- Monitor and record live signals from 16 to 256 low-noise amplifier channels using RHD2000 biopotential amplifier chips.
- Reconfigure amplifier bandwidths and sampling rates from software.
- Measure in situ electrode impedances (both magnitude and phase) at arbitrary frequencies with the click of a button.
- Use eight on-board digital-to-analog converters (DACs) to reconstruct analog waveforms from selected amplifier channels with <1 ms latency.
- Monitor audio of any two amplifier signals using a stereo "line out" jack.
- Record up to eight auxiliary analog inputs and 16 digital inputs synchronized with amplifier data.

These generated binary datasets which cannot be interpreted easily. And we need to use MATLAB in order to decipher the information and perform our customized analysis.

Traditional Solution:

(Work but still room)

The traditional solution developed by Intan Technologies and Dr. Adrian Andelin consists of two parts: reading the binary datasets and plotting with analysis.

Step 1: generate Intan ".rhd" files

From the RHD2000 Evaluation System, our electrophysiology data was recorded in segments of 60s, each includes input of signals from different channels.

Step 2: read Intan ".rhd" files

Intan files consists of binary or hex information like following (Figure 8):

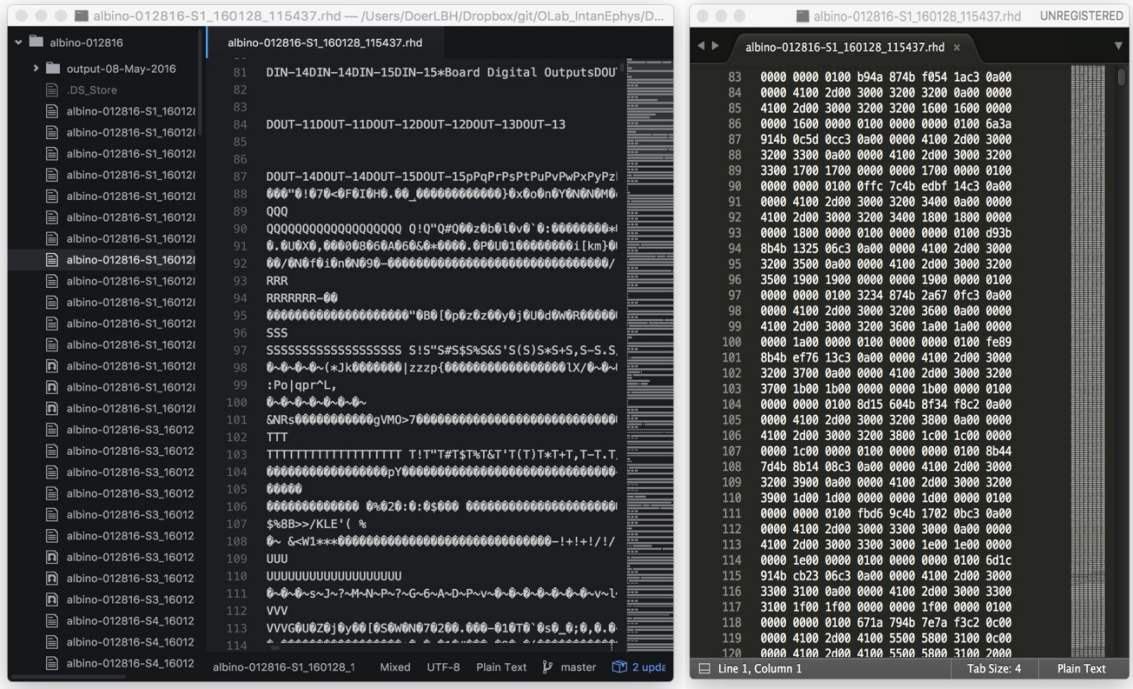


Figure 8. ".rhd" files are shown in binary (left) or hex (right), either unreadable

As shown, these ".rhd" files are not readable and has to be interpreted by official read_Intan_RHD2000_file.m and it generated a series of variables (Figure 9):

```
>> whos
```

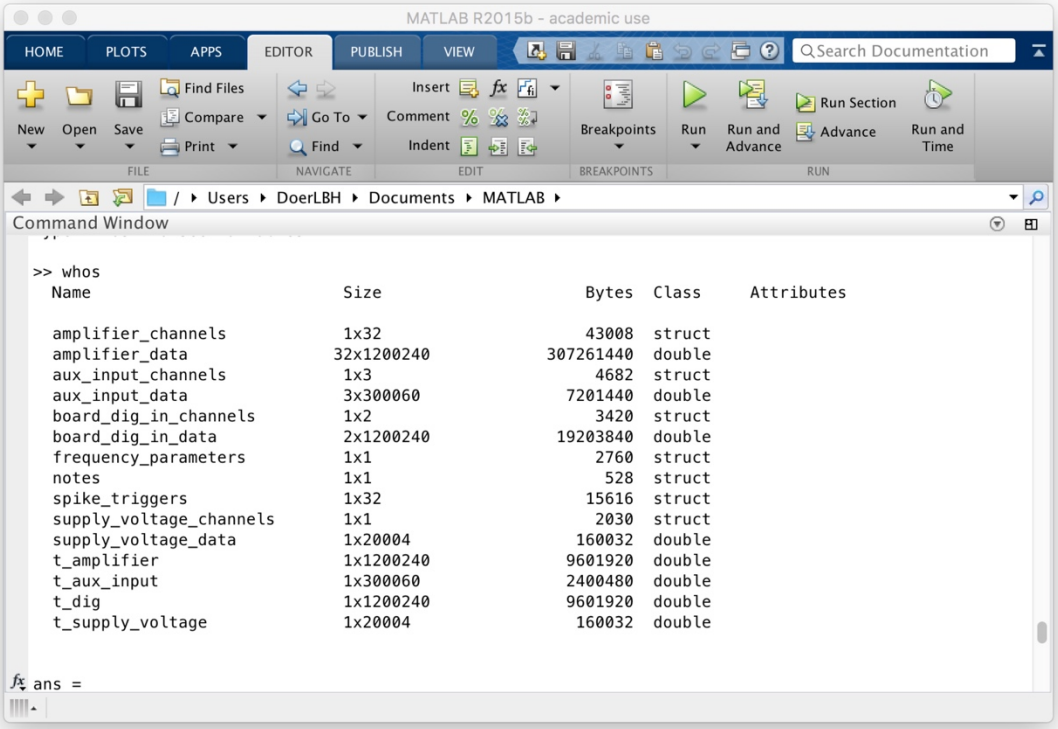


Figure 9. variables generated by read_Intan_RHD2000_file.m

Step 3: analyze variables extracted from ".rhd" files

Running backup_IntanEphysAnalysis_033016_Original.m, I encountered multiple issues, mostly due to MATLAB version discrepancy. And after debugging them, finally I made it run in backup_160407_IntanEphysAnalysis.m, whose console window is shown below in Figure 10.

This version of fixed analysis method, as implied in Figure 10 of this instance, still encounters various problems analyzing different cases of ".rhd" files, including but not limited by "Index exceeds matrix dimensions", "Reference to non-existent field 'Lrastercell'" etc..

This is due to the implicit nature of different cases of electrophysiology recording reflected by ".rhd". And to ensure success, one needs to run separate steps and modify various locations of parameters in order to make it run smoothly by getting rid of the errors.

Figure 10. Console window of the traditional method on a test "*.rhd" file

Step 4: export raster plots and recording spikes plot

As shown below in Figure 11, there is the spikes plot and raster plots for left eye and right eye after multiple modifications in parameters in each cases. However, to be noted, this is only for one segment of 60 seconds in a recording case, which means it doesn't necessary covers scientific meaning due to its local limitation.

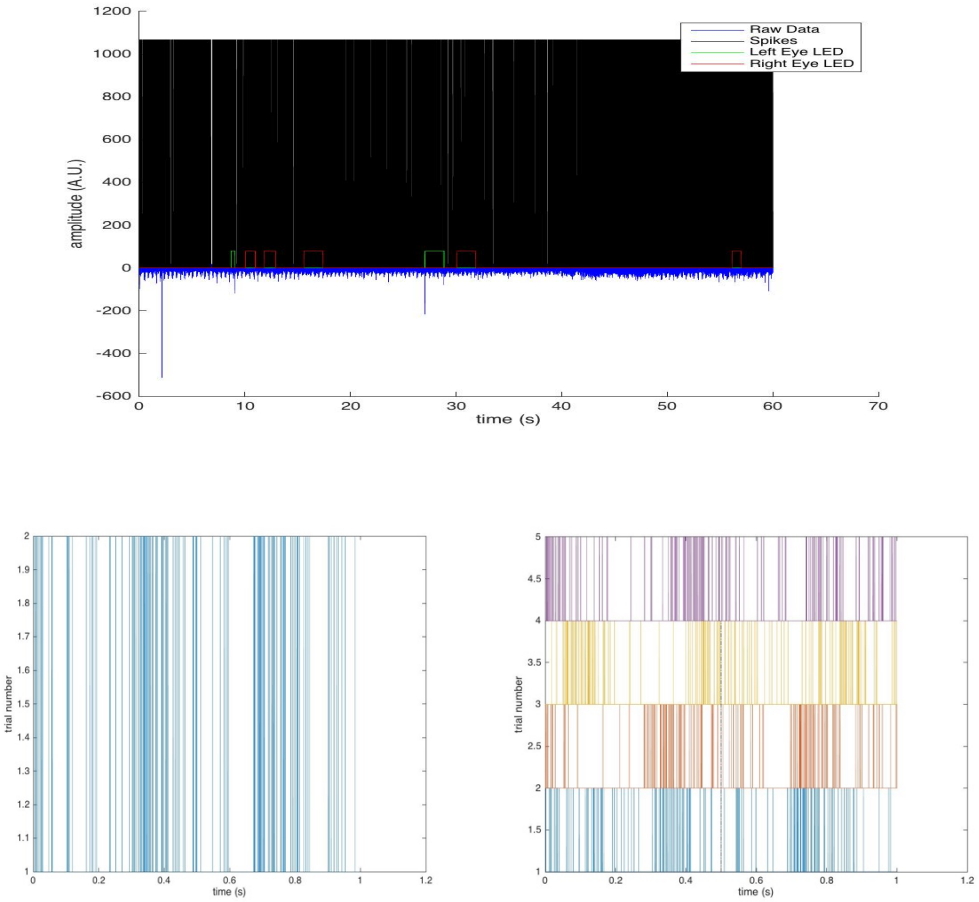


Figure 11. spikes plot and raster plots for left eye (left) and right eye (right)

Issues to tackle:

(Efficiency, Validity, and Human error)

1. Efficiency

The traditional solution developed by Intan Technologies and Dr. Adrian Andelin consists of two parts: reading the binary datasets and plotting with analysis.

Take current data collected from lab in the past half year as an example:

011216 (ignored due to failed testing of Intan equipment)	
albino-012816	129 cases
albino-020116	139 cases
MD-012516	62 cases
Normal-012616	111 cases
In total	**129 + 139 + 62 + 111 = 441 cases**

If one use this method, taking an average analyzing time of one instances for 40 minutes (for running the program, changing variables based on different properties of each case to make it work, exporting the images), he or she would be spending 441 * 40 minutes = **17,640 minutes** = 294 hours = 12.25 days (for 24 hours) = **14.7 weeks** (for 20 hours per week) = **147 days** (2 hours per day). This is terrifying for a lab to analysis only four rats.

2. Validity

As shown in previous graphs and analysis, the graphs and analysis are only done for one single ".rhd" file which only covers 60 seconds of segments of one case of recording. That makes the single analysis moot, so we need to add them together to see the difference in the entire recording dataset.

3. Human error

Because we have to change parameters in each sections of the code, not only does it cost a long time, but also grants chances to make multiple mistakes among these steps. Another thing to be noted is that, as shown in previous graphs and analysis, the graphs and analysis are only done for one segment, but we need the parameters to be the same for different segments of one single recording, which is commonly overlooked.

Concatenation Attempt 1: Mac Terminal (failed)

Perhaps we can concatenate the ".rhd" files of the same recording instance into one single file.

I tried below (Figure 12), making the assumption that ".rhd" files are simply headless binary datasets that can be easily combined.

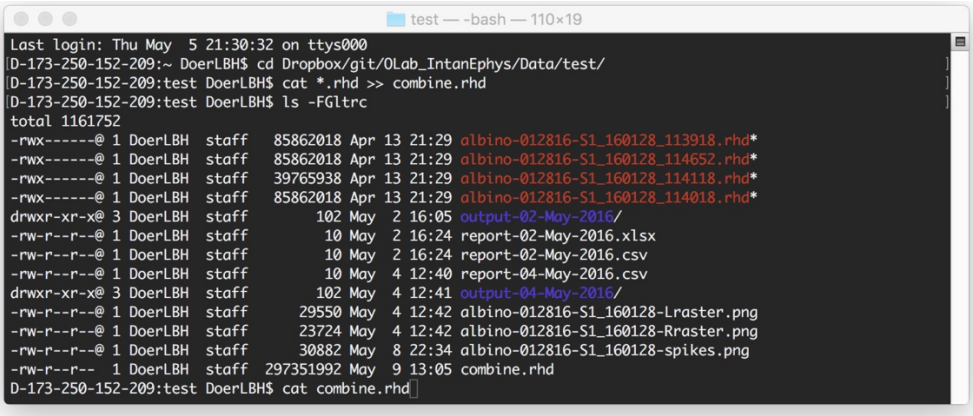


Figure 12. Mac terminal command to concatenate ".rhd" files

My assumption was wrong (Figure 13), so we need to combine them later via MATLAB.

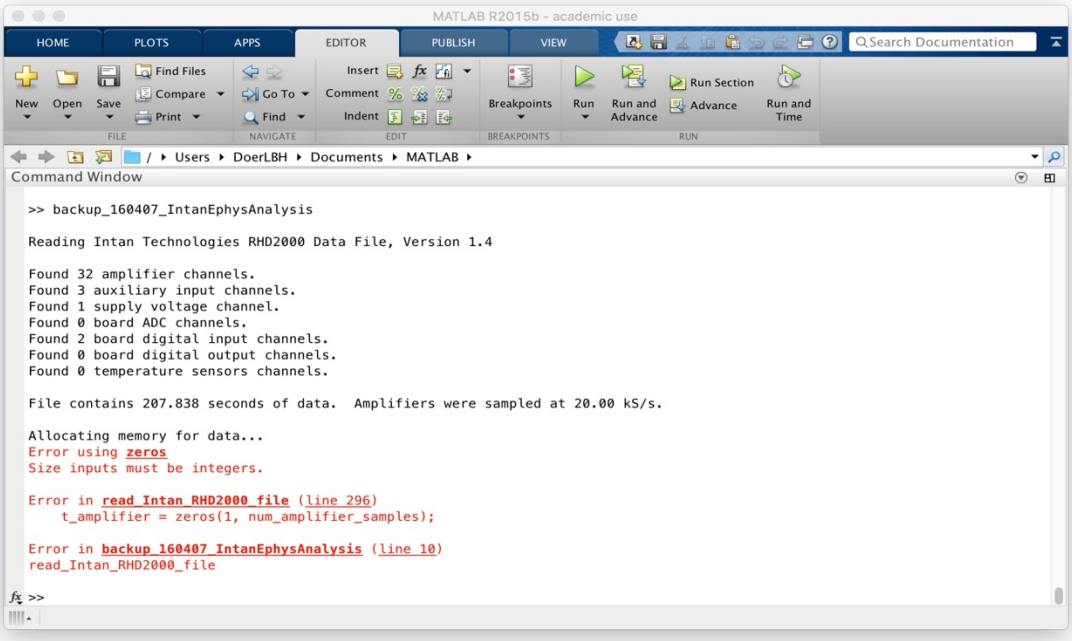


Figure 13. Error message by MATLAB for command concatenated ".rhd" file

Concatenation Attempt 2: MATLAB

(success)

I decided to use a different approach by combining different segments of recording data inside MATLAB. To do so, I have two ways:

1) **Combing the signals when reading each ".rhd" file.**

I modified the reading code into _combine.m, but unfortunately, it is not taking the variables as parallels as I expected. Thus, I decided to use the other approach.

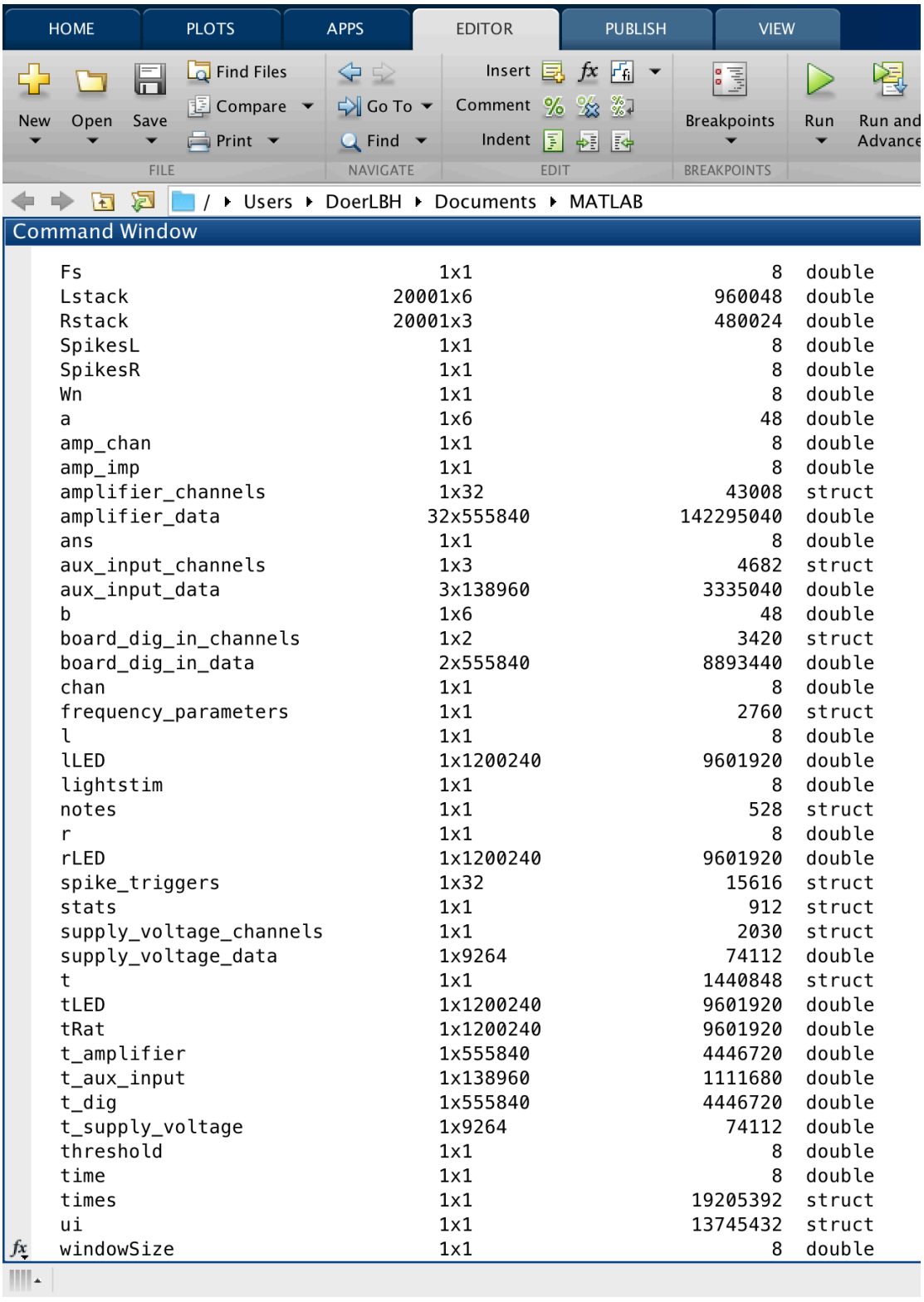
2) **Combing the signals after reading each ".rhd" file but before analysis**

I need an enumerative loop in order to do that. But when and where should I combine the data, and what data are shared by different cases?

After the reading of inputs, there are following variables (Figure 14). However, only the following marked red need to be combined:

amplifier_channels	1x32	43008	struct
amplifier_data	32x555840	142295040	double
aux_input_channels	1x3	4682	struct
aux_input_data	3x138960	3335040	double
board_dig_in_channels	1x2	3420	struct
board_dig_in_data	2x555840	8893440	double
frequency_parameters	1x1	2760	struct
notes	1x1	528	struct
spike_triggers	1x32	15616	struct
supply_voltage_channels	1x1	2030	struct
supply_voltage_data	1x9264	74112	double
t_amplifier	1x555840	4446720	double
t_aux_input	1x138960	1111680	double
t_dig	1x555840	4446720	double
t_supply_voltage	1x9264	74112	double

(Reference: /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test)



Fs	1x1	8	double
Lstack	20001x6	960048	double
Rstack	20001x3	480024	double
SpikesL	1x1	8	double
SpikesR	1x1	8	double
Wn	1x1	8	double
a	1x6	48	double
amp_chan	1x1	8	double
amp_imp	1x1	8	double
amplifier_channels	1x32	43008	struct
amplifier_data	32x555840	142295040	double
ans	1x1	8	double
aux_input_channels	1x3	4682	struct
aux_input_data	3x138960	3335040	double
b	1x6	48	double
board_dig_in_channels	1x2	3420	struct
board_dig_in_data	2x555840	8893440	double
chan	1x1	8	double
frequency_parameters	1x1	2760	struct
l	1x1	8	double
lLED	1x1200240	9601920	double
lightstim	1x1	8	double
notes	1x1	528	struct
r	1x1	8	double
rLED	1x1200240	9601920	double
spike_triggers	1x32	15616	struct
stats	1x1	912	struct
supply_voltage_channels	1x1	2030	struct
supply_voltage_data	1x9264	74112	double
t	1x1	1440848	struct
tLED	1x1200240	9601920	double
tRat	1x1200240	9601920	double
t_amplifier	1x555840	4446720	double
t_aux_input	1x138960	1111680	double
t_dig	1x555840	4446720	double
t_supply_voltage	1x9264	74112	double
threshold	1x1	8	double
time	1x1	8	double
times	1x1	19205392	struct
ui	1x1	13745432	struct
windowSize	1x1	8	double

Figure 14. screenshot of generated variables

Protocol Optimization

(success)

I optimized the codes with following new features I added (including but not limited by):

1. A user-friendly prompt to input folder path (Figure 15)

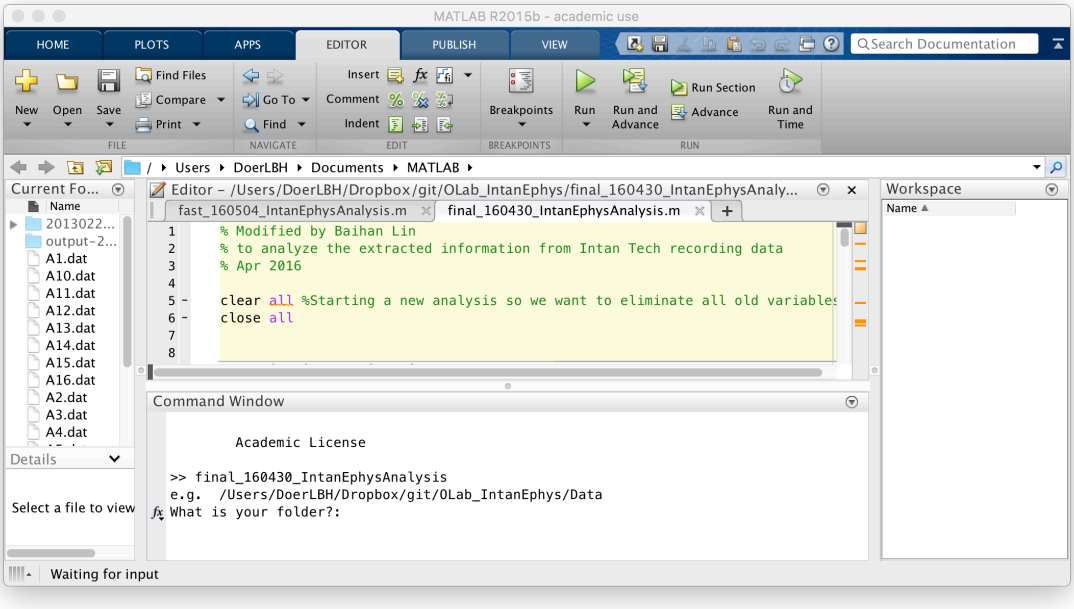


Figure 15. screenshot of the user-friendly prompt

2. read recursively all the ".rhd" files in the folder and subfolders
3. identify, group and sort ".rhd" files based on cases
4. concatenate consecutive input data in the different segments of the same case
5. automatically compare the amplifier channel to determine the one used
6. catch every possible exception to avoid crashes
7. better formatting of figures and save them as ".png" files in the folder
8. display variable outputs in a clearer usable way
9. add break points and debugging warnings

10. automatically extract the LED shinning times and assign to raster plots

11. choose clearer view of raster plots and fix the middle line issue

12. automatically add title and filename to plots generated

13. clear certain but not all variables in each loop to make sure smooth run

14. use diary to keep log in each analysis and save into a specific folder (Figure 16)

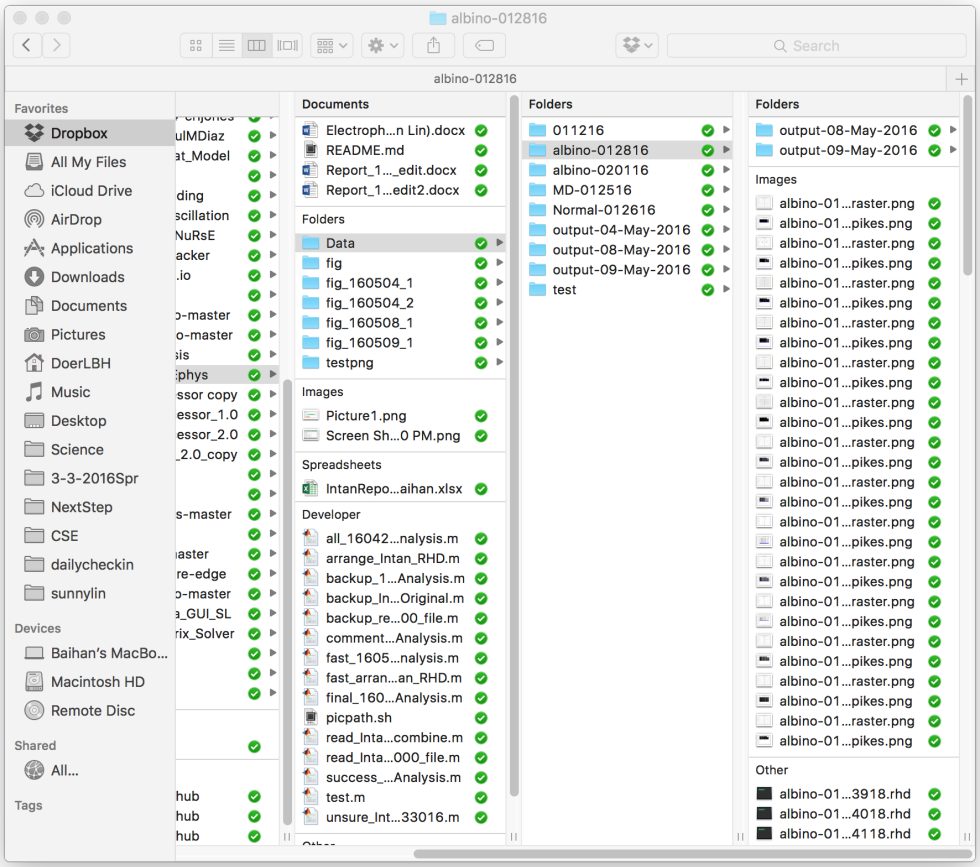


Figure 16. Output screenshot in the analyzed folder.

15. In summary, using my method, running the whole MATLAB analysis for all 441 cases can be accomplished within 10 minutes (normally 5 minutes is sufficient), with only one click and a wait for 10 minutes before viewing our products in specific folders. Comparing to the traditional method costing 17640 minutes, my method is 1764 times faster and saves almost 300 hours of work!

Excel Report Integration:

(success)

Now having accomplished the combination, analysis and plotting, I want it to be able to export analyzed parameters to Excel worksheet as our report.

I tried to use xlswrite() function in MATLAB but it seems MATLAB R2015b no longer supports xlsx format, thus I use a different approach.

I integrated terminal command and switch on diary functionality so as to store the console memory. With that in mind, I need to trim down the output of console window and speed up the analysis process by MATLAB.

To do that, in my fast_160504_IntanEphysAnalysis.m and fast_arrange_Intan_RHD.m, I commented out to suppress all unnecessary output. I also included "----------" for later word document arrangement before putting inside excel sheet.

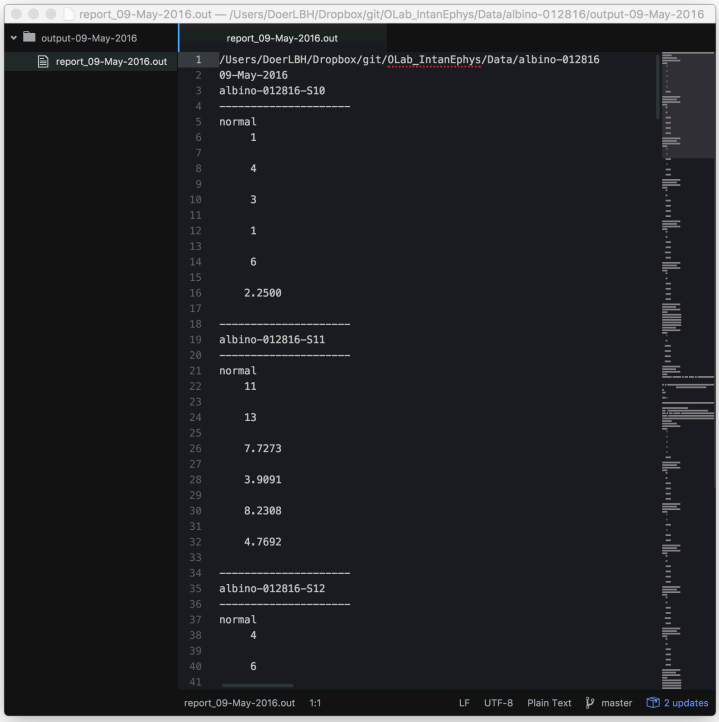


Figure 17. partial screenshot of output file formatting

The output file are shown like Figure 17, and it is ready to be put into Excel.

Frequently Asked Questions (FAQ)

(for more, email sunnylin@uw.edu or Google)

1. Why do I don't have enough plots generated?

This can be due to different reasons. A common mistake is to put the parent folder instead of the innermost subfolder as your input. If so, the program can mistakenly assign wrong attributes. Be sure to choose the innermost subfolder to start with. Another common reason can be a lack of LED input in the first place. This happens when both or one of the LED input is missing. Other possibilities will need more debugging.

2. Why are there mysterious blue horizontal lines across the spike plots?

This is due to the concatenation and it is normal. The program automatically add a datapoint of zero at zero second by default, thus creating the horizontal crossing.

3. Why do plots popping up and vanishing? Am I doing something wrong?

No, you are doing fine. The plots are designed to be popped when generated. But instead of letting hundred of plots flying out your screen, I closed them one by one in my codes.

4. How do I know whether the analysis is finished so I can close the program?

When you view the console and see ">>" in the front, you are all set to close.

5. Where can I find all the plots and output?

The log is located inside a folder name after "output" followed by date. The plots are located at the folder where the ".rhd" files are kept.

6. Why is there "fail to assign"? How do I fix it?

It is not clear why, but you can fix it quickly by restoring the unsuccessful ".rhd" files into a separate folder and rerun the program. Usually, will be successful. It is weird, but it works (as shown in /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test2).

Data Summary

(for 441 cases available now)

1. Excel report

After several steps of excel arrangements (Figure 18):

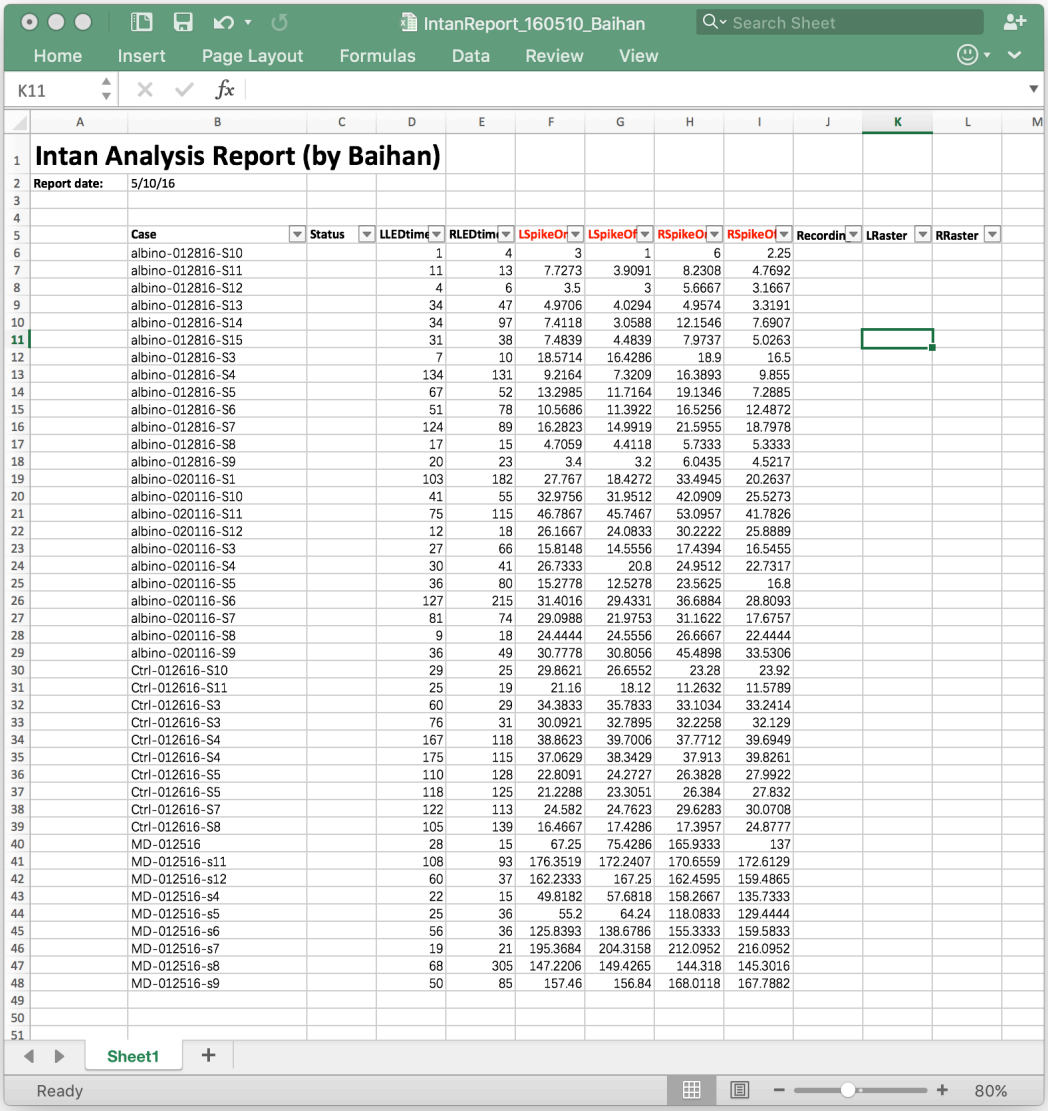
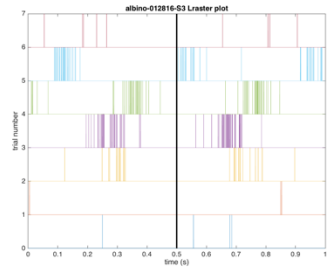
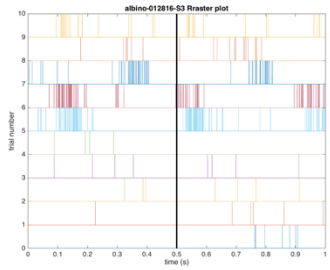


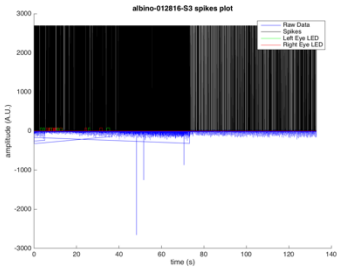
Figure 18. Intan Electrophysiology report 160510

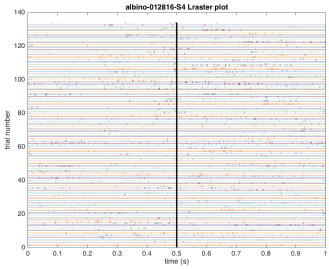
2. Figures

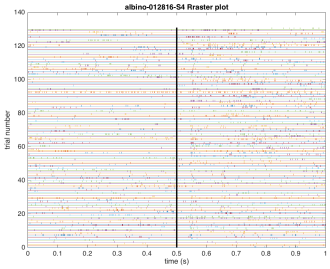
a) albino-012816

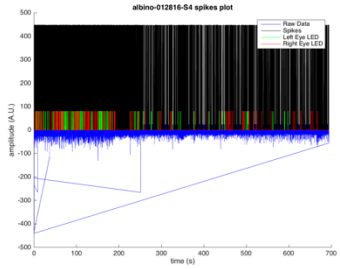


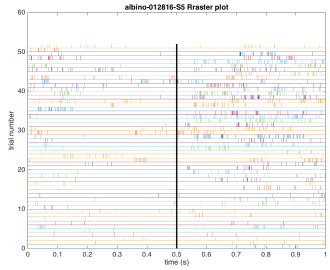


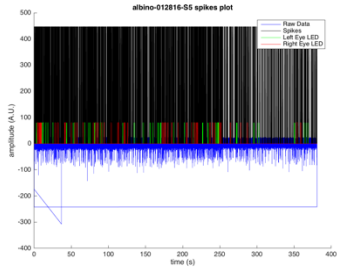


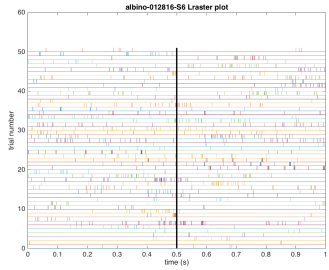


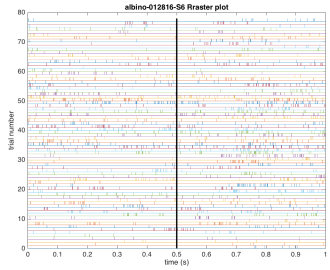


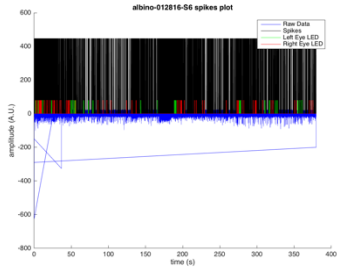


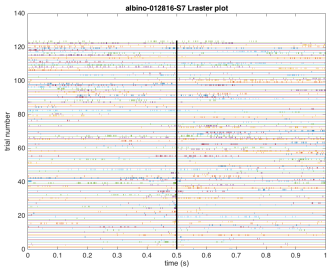


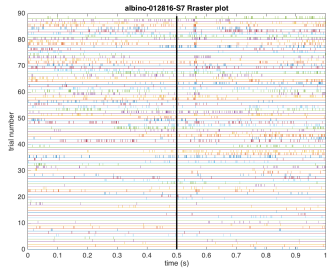


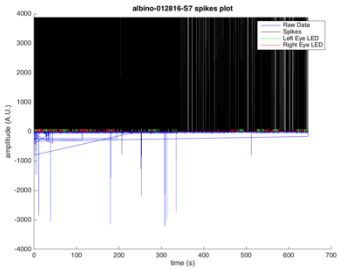


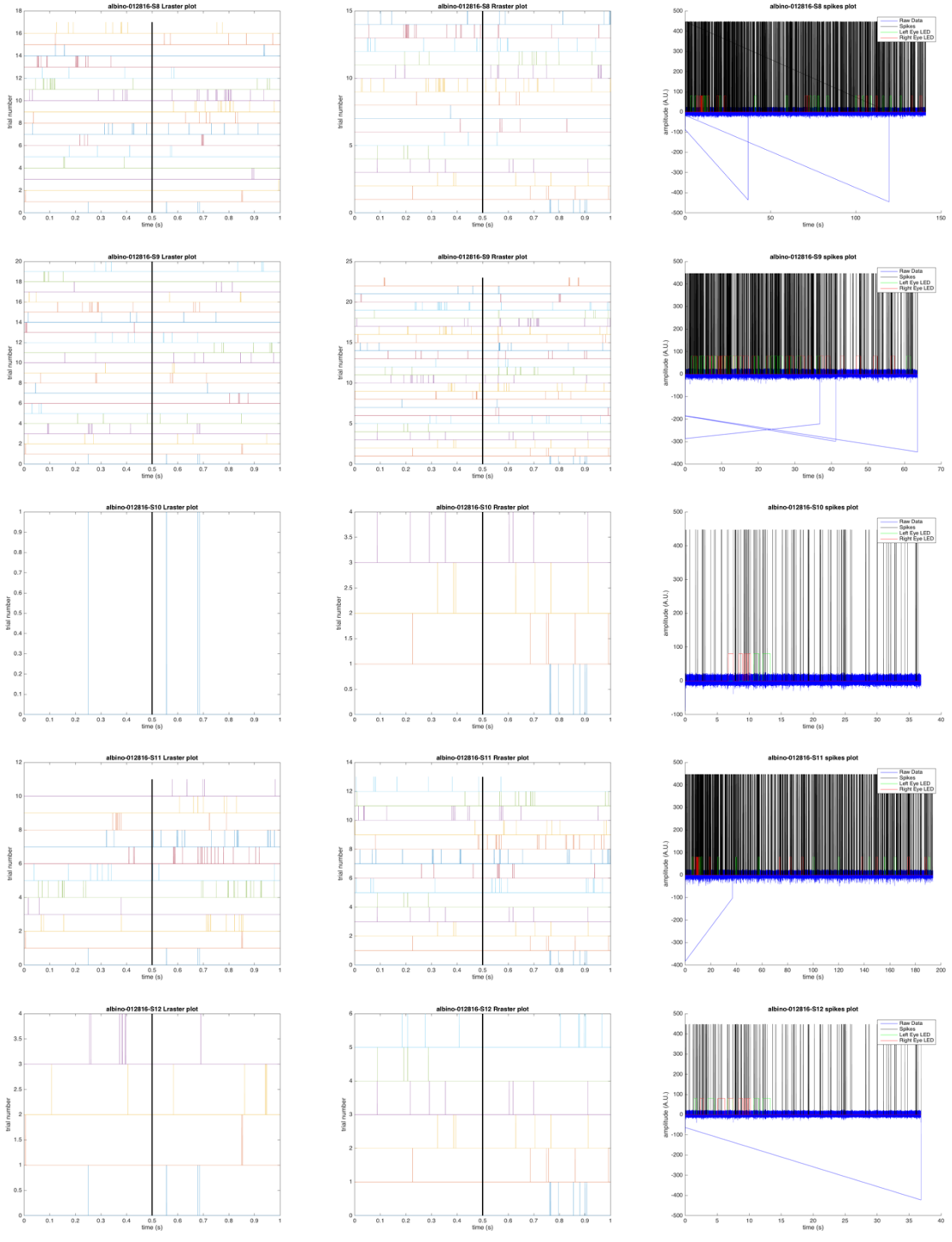


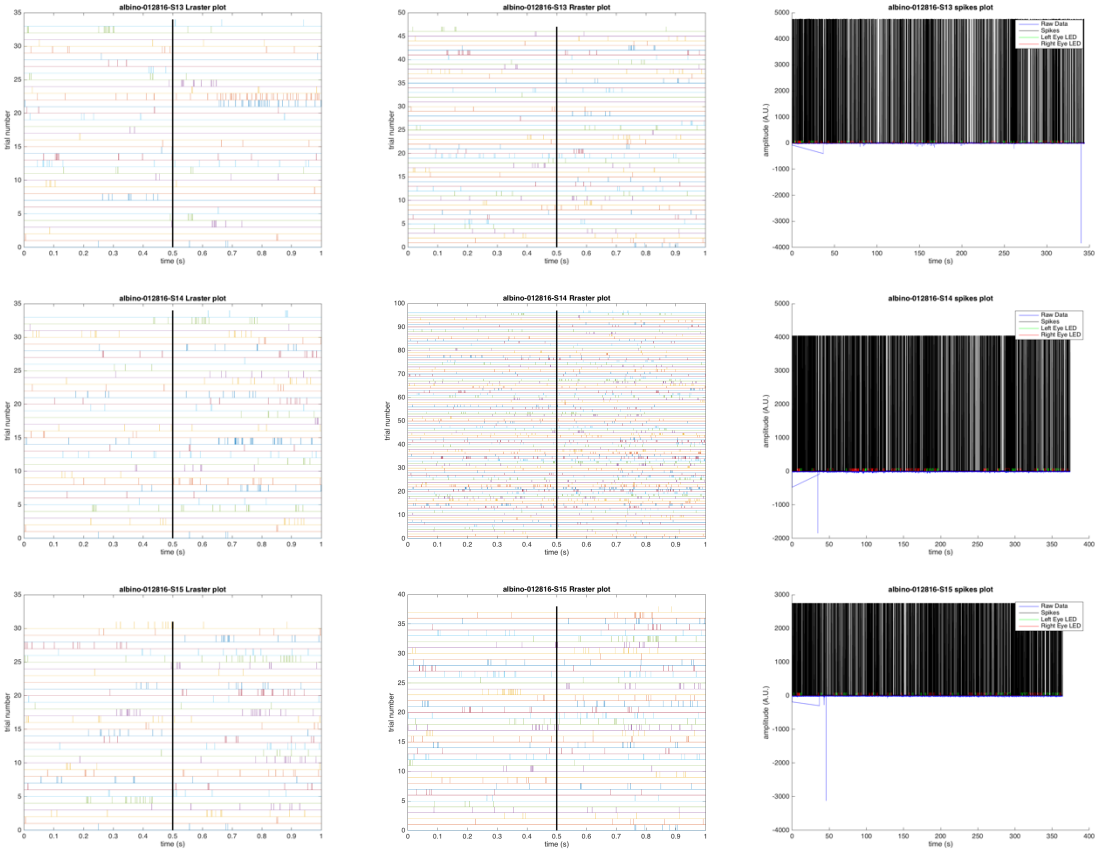




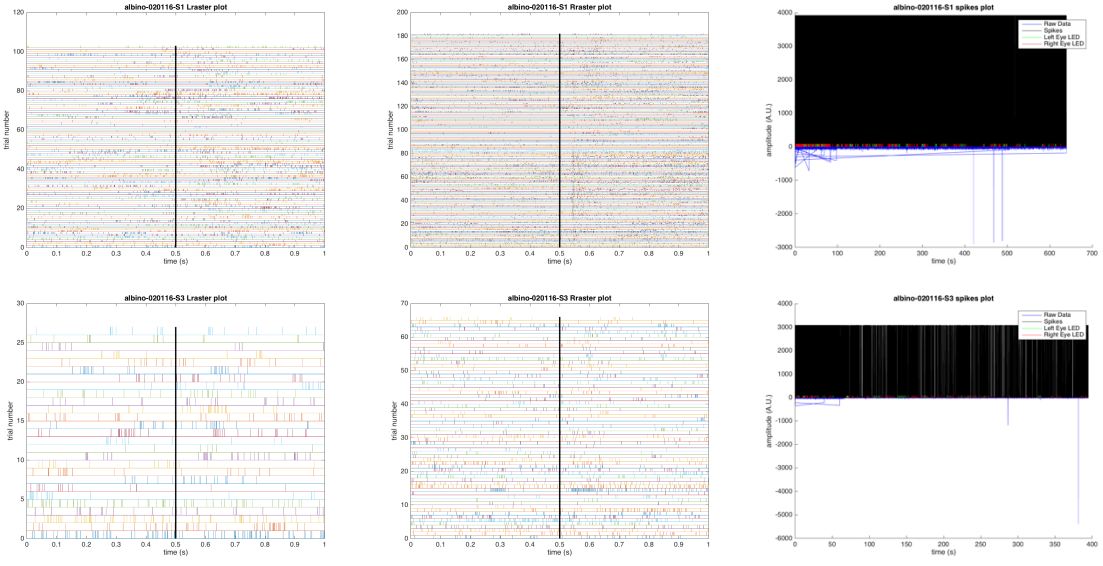


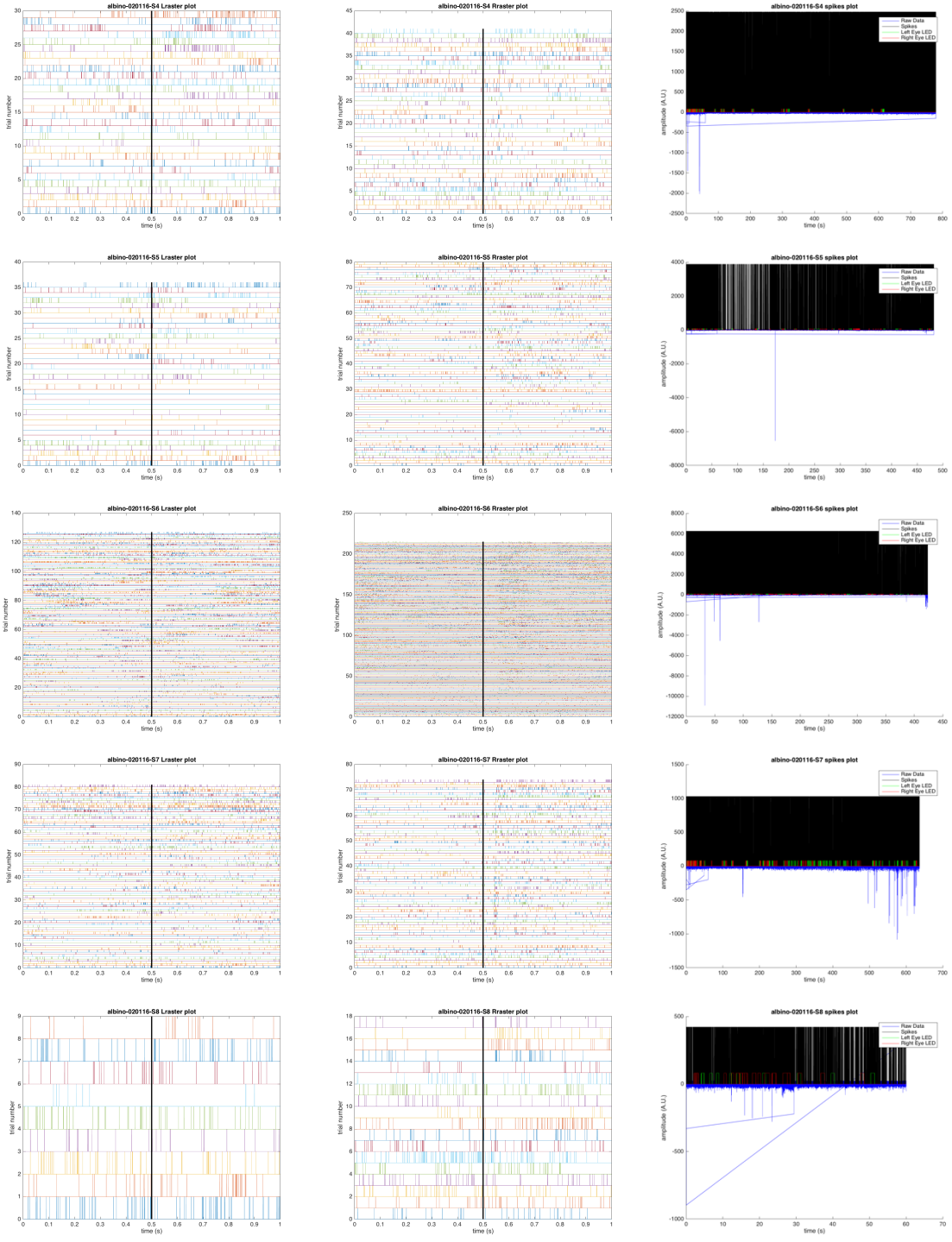


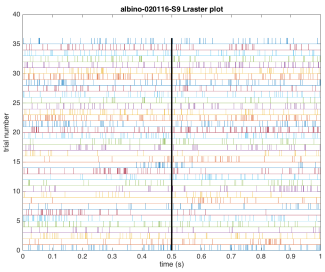

albino-012816-S8 Laster plot
albino-012816-S8 Rraster plot
albino-012816-S8 spikes plot
albino-012816-S9 Laster plot
albino-012816-S9 Rraster plot
albino-012816-S9 spikes plot
albino-012816-S10 Laster plot
albino-012816-S10 Rraster plot
albino-012816-S10 spikes plot
albino-012816-S11 Laster plot
albino-012816-S11 Rraster plot
albino-012816-S11 spikes plot
albino-012816-S12 Laster plot
albino-012816-S12 Rraster plot
albino-012816-S12 spikes plot
trial number
time (s)
amplitude (A.U.)
Raw Data
Spikes
Left Eye LED
Right Eye LED

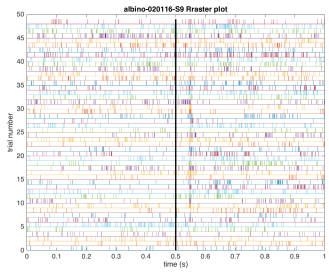


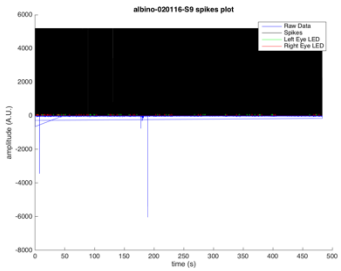
b) albino-020116

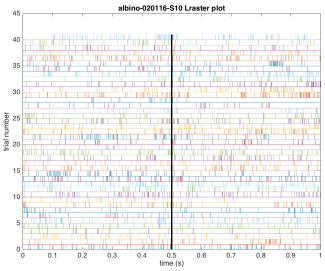


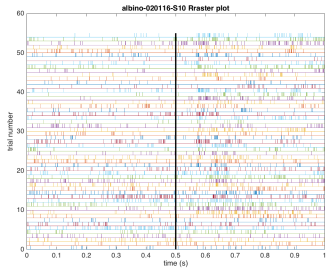

albino-020116-S4 Laster plot
albino-020116-S4 Rraster plot
albino-020116-S4 spikes plot
albino-020116-S5 Laster plot
albino-020116-S5 Rraster plot
albino-020116-S5 spikes plot
albino-020116-S6 Laster plot
albino-020116-S6 Rraster plot
albino-020116-S6 spikes plot
albino-020116-S7 Laster plot
albino-020116-S7 Rraster plot
albino-020116-S7 spikes plot
albino-020116-S8 Laster plot
albino-020116-S8 Rraster plot
albino-020116-S8 spikes plot
trial number
time (s)
amplitude (A.U.)
Raw Data
Spikes
Left Eye LED
Right Eye LED

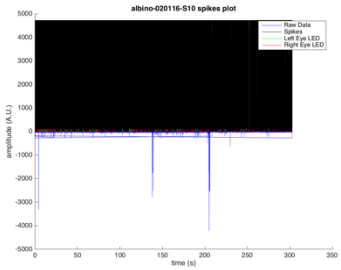

albino-020116-S9 Lraster plot
trial number
time (s)

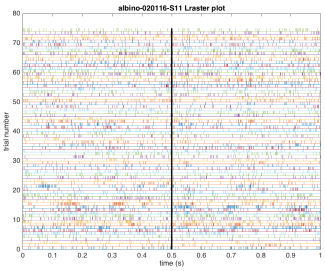

albino-020116-S9 Rraster plot
trial number
time (s)

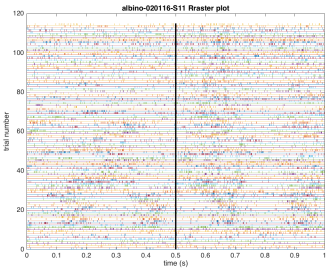

albino-020116-S9 spikes plot
Raw Data
Spikes
Left Eye LED
Right Eye LED
amplitude (A.U.)
time (s)

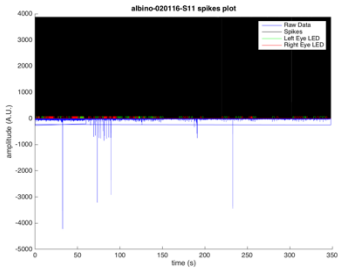

albino-020116-S10 Lraster plot
trial number
time (s)


albino-020116-S10 Rraster plot
trial number
time (s)

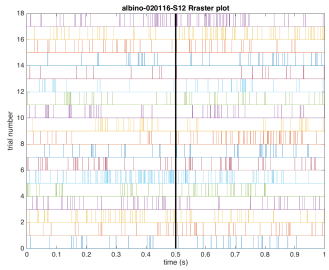

albino-020116-S10 spikes plot
Raw Data
Spikes
Left Eye LED
Right Eye LED
amplitude (A.U.)
time (s)

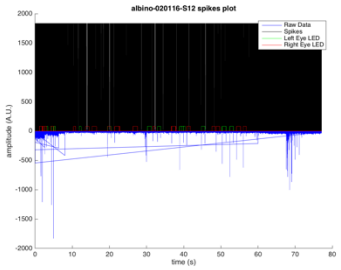

albino-020116-S11 Lraster plot
trial number
time (s)


albino-020116-S11 Rraster plot
trial number
time (s)

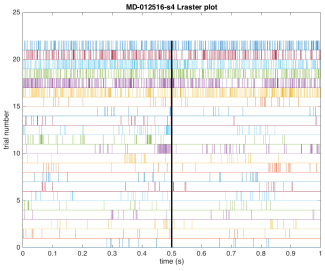

albino-020116-S11 spikes plot
Raw Data
Spikes
Left Eye LED
Right Eye LED
amplitude (A.U.)
time (s)

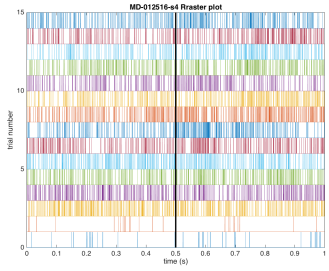

albino-020116-S12 Lraster plot
trial number
time (s)

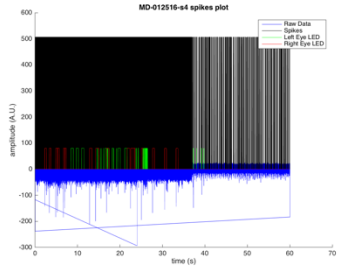

albino-020116-S12 Rraster plot
trial number
time (s)

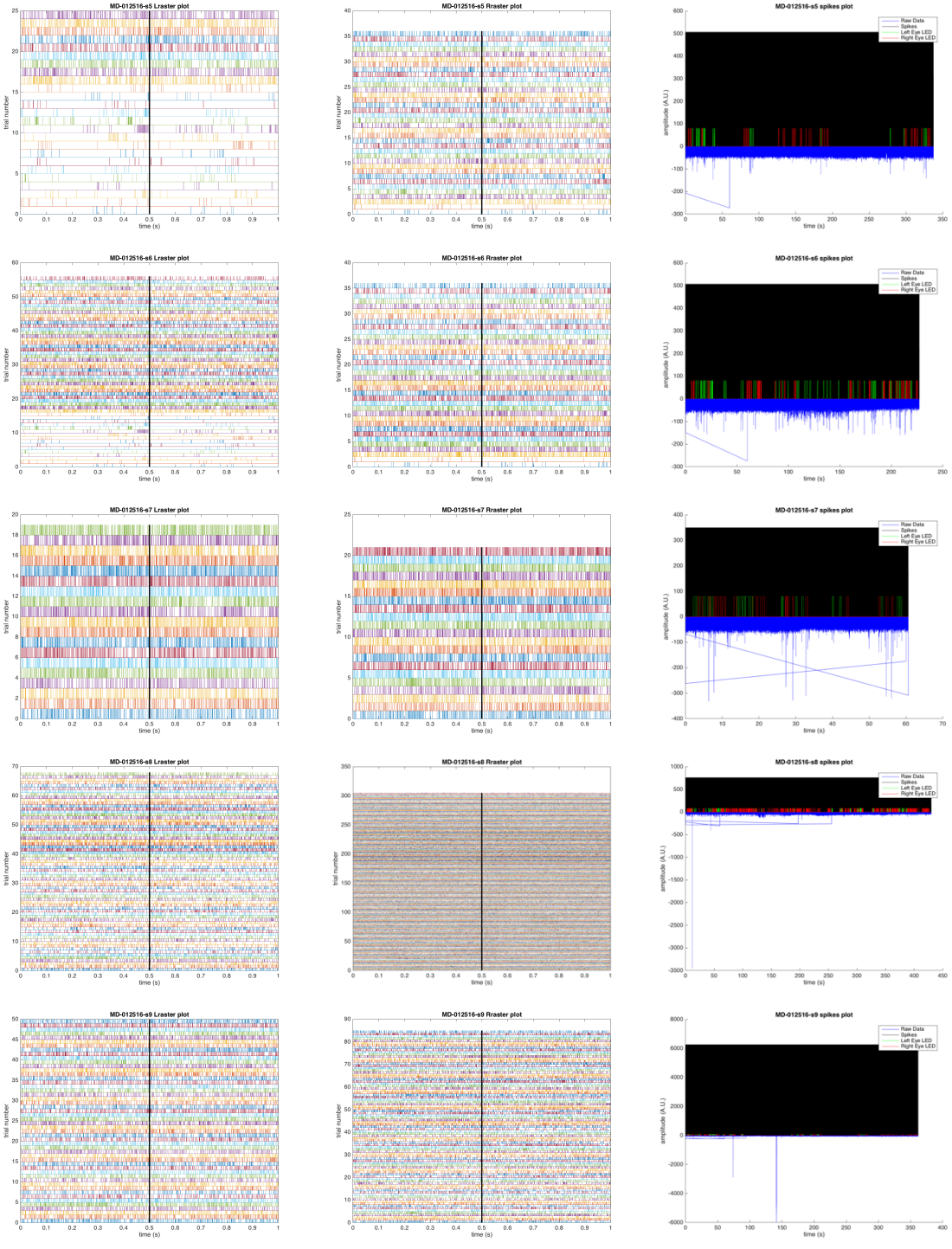

albino-020116-S12 spikes plot
Raw Data
Spikes
Left Eye LED
Right Eye LED
amplitude (A.U.)
time (s)

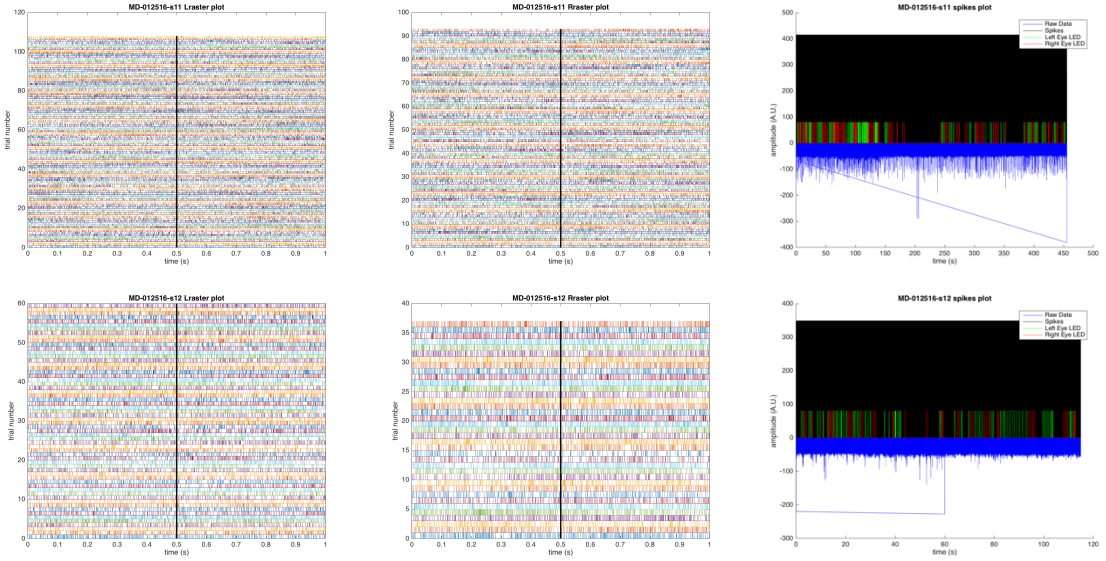
c) MD-012516


MD-012516-s4 Lraster plot
trial number
time (s)

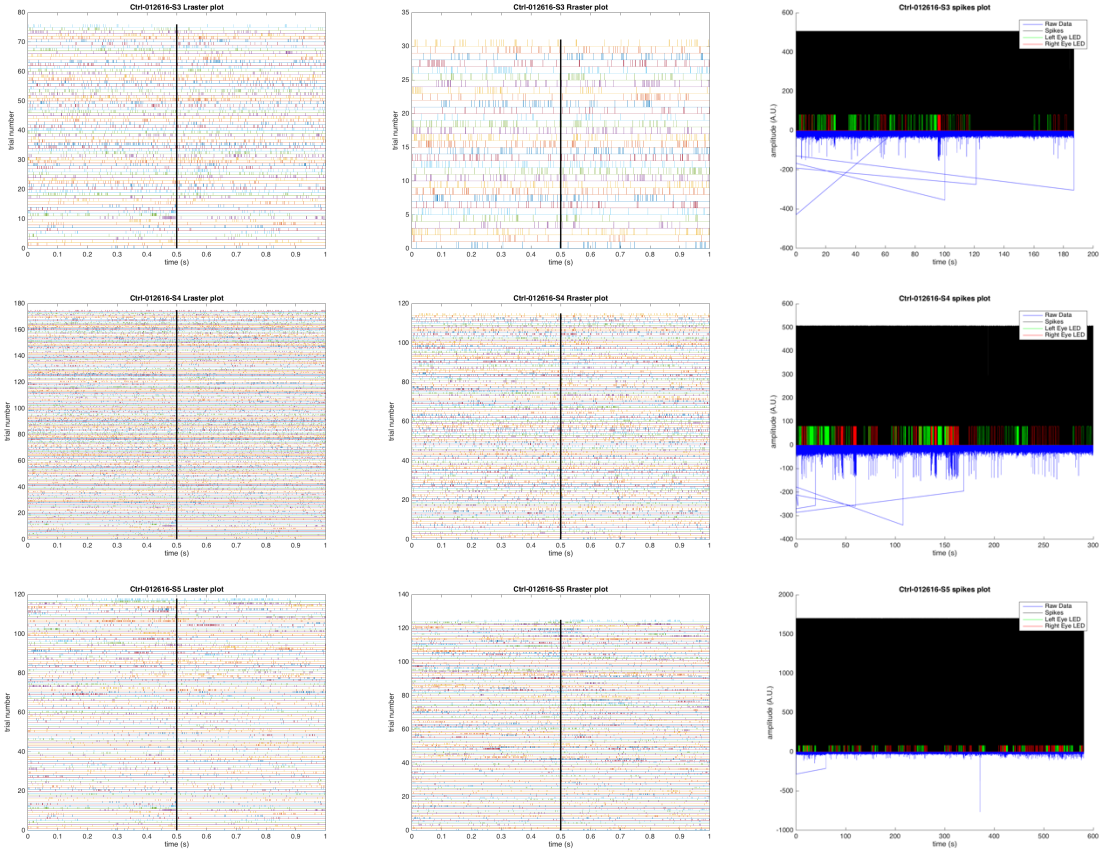

MD-012516-s4 Rraster plot
trial number
time (s)


MD-012516-s4 spikes plot
Raw Data
Spikes
Left Eye LED
Right Eye LED
amplitude (A.U.)
time (s)


MD-012516-s5 Laster plot
MD-012516-s5 Raster plot
MD-012516-s5 spikes plot
MD-012516-s6 Laster plot
MD-012516-s6 Raster plot
MD-012516-s6 spikes plot
MD-012516-s7 Laster plot
MD-012516-s7 Raster plot
MD-012516-s7 spikes plot
MD-012516-s8 Laster plot
MD-012516-s8 Raster plot
MD-012516-s8 spikes plot
MD-012516-s9 Laster plot
MD-012516-s9 Raster plot
MD-012516-s9 spikes plot
trial number
time (s)
amplitude (A.U.)
Raw Data
Spikes
Left Eye LED
Right Eye LED



d) Normal-012616



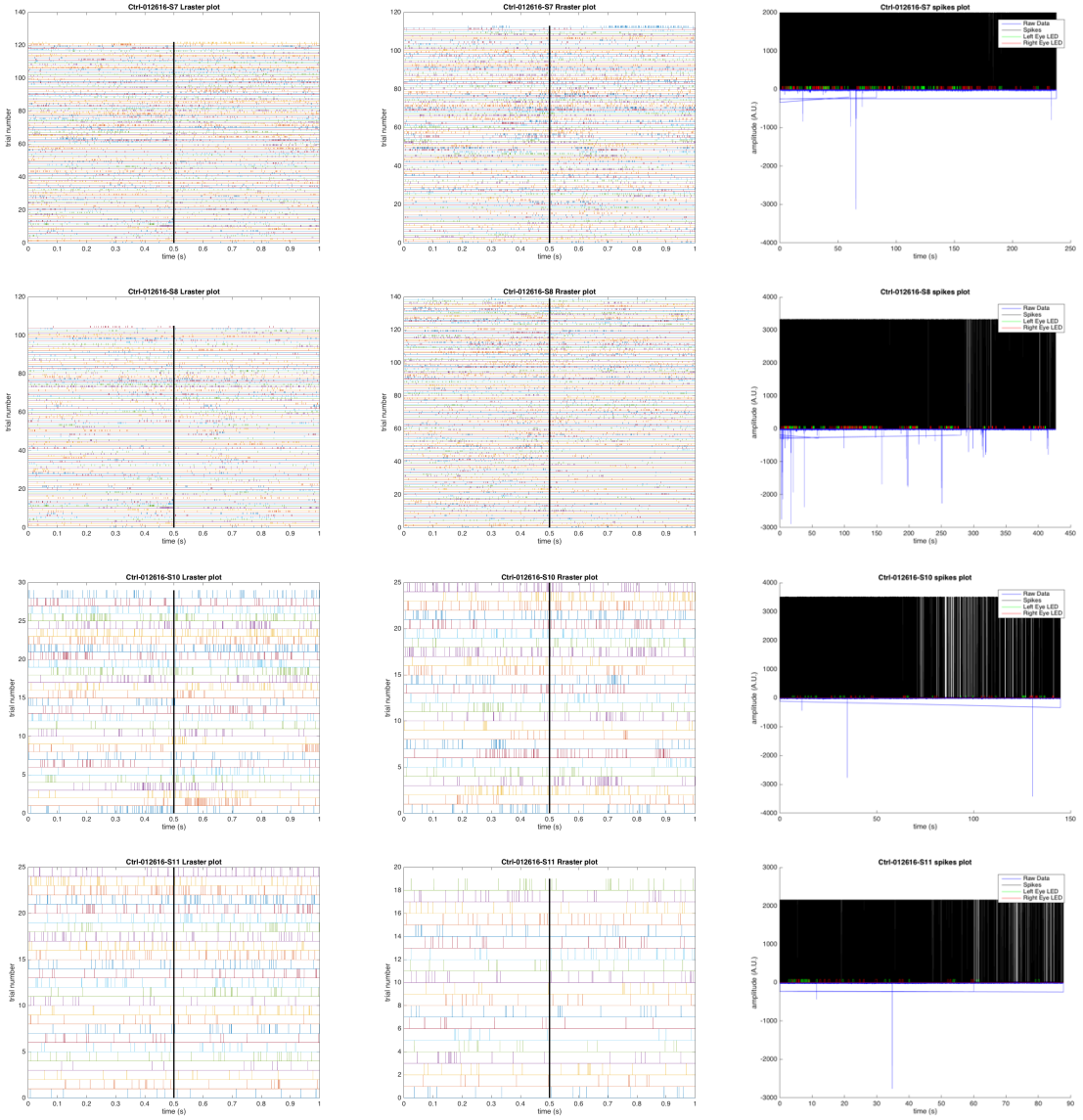


Figure 19. All the generated graphs from 441 ".rhd" files including spikes and raster plots

Code Summary:

(in order of importance from high to low)

1. fast_160504_IntanEphysAnalysis.m

```
% Modified by Baihan Lin
% to analyze the extracted information from Intan Tech recording data
% Apr 2016

clear all; %Starting a new analysis so we want to eliminate all old variables
close all;

%% Begin with opening file to be analyzed
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
% read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.

% read_Intan_RHD2000_file_combine
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%
% Example:
```

```
% >> clear
% >> read_Intan_RHD200_file
% >> whos
% >> amplifier_channels(1)
% >> plot(t_amplifier, amplifier_data(1,:))

% Here I change it from:
%[file, path, filterindex] = uigetfile('*.rhd', 'Select an RHD2000 Data File',
'MultiSelect', 'off');

% Read most recent file automatically.
%path = 'C:\Users\Reid\Documents\RHD2132\testing\';
%d = dir([path '*.rhd']);
%file = d(end).name;

% I decided to use automatic method for data collection:

prompt = 'What is your folder?: ';
disp('e.g.  /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data');
path = input(prompt,'s');
% /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test

% So that I can specify a folder to access all data files.

%[status, list] = system('cd path');

[~,list] = system(['find ' path ' -type f -name "*.rhd"']);

system(['mkdir ' path '/output-' date]);
system(['cd ' path '/output-' date]);

diary(strcat(path, '/output-', date, '/report_', date, '.out'));
diary on;
disp(path);
disp(date);

% warning('off','MATLAB:xlswrite:AddSheet');
% xlsfile = strcat(path, '/report-', date, '.xlsx');
% mycell = {'Excel'};
% xlswrite(xlsfile,mycell);
% % rpt = {strcat('report-', date, ' by Baihan Lin')};
```

```
% % xlswrite(xlsfile, rpt(1),'Report','A1');

files = strsplit(list);
length(files);

for countfile = 1:length(files)
    trials{countfile} = files{countfile}(1:end-11);
end

trials = unique(trials);
trials = trials(~cellfun('isempty',trials));

index = strfind(files, trials{1});

for trial = 1 : length(trials)
    filename = trials{trial};
    indexsep = strfind(filename,'/');
    last = indexsep(end);

    index = strfind(files, filename);
    indexmat = cell2mat(index);
    [~,first,~] = unique(indexmat, 'first');
    fast_arrange_Intan_RHD(files{first(1)});

    casename = strtok(filename(last+1:end), '_');

    amp_data = amplifier_data;
    disp(casename);
    try
        ai_data = aux_input_data;
    catch exception
        disp('bad_ai_intialization');
    end
    try
        bdi_data = board_dig_in_data;
    catch exception
        disp('bad_bdi_initialization');
    end
    try
        sv_data = supply_voltage_data;
    catch exception
```

```
    disp('bad_sv_initialization');
end
try
    t_ai = t_aux_input;
catch exception
    disp('bad_t_ai_initialzation');
end
try
    t_d = t_dig;
catch exception
    disp('bad_t_dig_initialization');
end
try
    t_sv = t_supply_voltage;
catch exception
    disp('bad_t_sv_initialization');
end
disp('----------------------');
disp('normal');
%     catch exception
%         disp('----------------------');
%          disp(filename(last+1:end));
%          disp('Fail_to_initialize')
%     end
t_amp = t_amplifier;

if length(index) > 1
    for ind = 2 : length(index)
        if index{ind} == 1
            fast_arrange_Intan_RHD(files{ind});
            try
                amp_data = [amp_data, amplifier_data];
            catch exception
                disp('fail_to_reassign_amp_1')
            end
            try
                ai_data = [ai_data, aux_input_data];
            catch exception
                disp('fail_to_reassign_ai_1')
            end
            try
```

```
            bdi_data = [bdi_data, board_dig_in_data];
        catch exception
            disp('fail_to_reassign_bdi_1')
        end
        try
            sv_data = [sv_data, supply_voltage_data];
        catch exception
            disp('fail_to_reassign_sv_1')
        end
        try
            t_amp = [t_amp, t_amplifier];
        catch exception
            disp('fail_to_reassign_t_amp_1')
        end
        try
            t_ai =  [t_ai,t_aux_input];
        catch exception
            disp('fail_to_reassign_t_ai_1')
        end
        try
            t_d = [t_d, t_dig];
        catch exception
            disp('fail_to_reassign_t_d_1')
        end
        try
            t_sv = [t_sv, t_supply_voltage];
        catch exception
            disp('fail_to_reassign_t_sv_1')
        end
    end
  end
end

amplifier_data = amp_data;
t_amplifier = t_amp;
try
    aux_input_data = ai_data;
    board_dig_in_data = bdi_data;
    supply_voltage_data = sv_data;
    t_aux_input = t_ai;
    t_dig = t_d;
```

```
        t_supply_voltage = t_sv;
    catch
        disp('fail_to_reassign_2');
    end

 %check what variables have been imported, especially if youre unsure
 %whether accessory amplifier channels were disabled or not during recording

    %% Establishing some basic variables from values pulled in by above
function

    amp_chan = 1;
    amp_imp = amplifier_channels(1).electrode_impedance_magnitude;

    for chan = 1 : length(amplifier_channels)
        if amplifier_channels(chan).electrode_impedance_magnitude < amp_imp
            amp_chan = chan;
            amp_imp =
amplifier_channels(chan).electrode_impedance_magnitude;
        end
    end

    %    disp('amplifer information');
    amplifier_channels(amp_chan);

    %Channel data is being collected on (on
    %preamplifier this would be 'A-004'
    %Above command gives data output about the channel on which data is being
    %collected

    %% Variable name changes below to simplify:

    try
        tRat = t_amplifier; %time variable for ephys data
        tLED = t_dig; %time variable for LED data
        try
            ui.ratData = amplifier_data(amp_chan,:);
        catch exception
            ui.ratData = amplifier_data(1,:);
        end
        lLED = board_dig_in_data(1,:);
```

```
        rLED = board_dig_in_data(2,:);
    catch
    end

    % %% Check variables look right-- plot should be identical to last one
    % figure % for spike detection
    %     hold on
    %     plot(tRat, ui.ratData,'blue')
    %     plot(tLED,lLED,'red')  %max makes red lines continue across top
half of vertical axis
    %     plot(tLED,rLED,'green')
    %     xlabel 'time (s)'
    %     ylabel 'amplitude (A.U.)'
    %     legend('Raw Data', 'Left Eye LED','Right Eye LED')

    %% filter data
    Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    try
        ui.ratData = filtfilt(b,a,ui.ratData);
        %% invert signal data for thresholding
        ui.ratData = ui.ratData.*(-1);
    catch
    end

    %% Setting threshold for spikes and finding light ON times
    threshold = 25;
    Fs =  20000; %amplifier_sample_rate
    windowSize = Fs * 0.05; %creates our time interval by taking the 20k
    % sampling rate at which the data was collected and converts it to
    % timestamps collected every millisecond, in other words the value of
    % windowSize is 1 ms.

    % Finding all spikes in recording:
    try
        ui.spikes = diff(ui.ratData > threshold) > 0.1;
    catch
    end

    %% Plot Again with spikes showing & correct time axis now:
```

```
    try
        time = length(tRat)-1;
        fig1 = figure; % for spike detection
        %     figure('units','normalized','position',[0 0 1 1]);
        %     figure('Visible','off');
        hold on;
        plot(tRat(1:time), ui.ratData(1:time),'blue');
        plot(tRat(1:time), ui.spikes*max(ui.ratData),'black');
        plot(tLED(1:time),lLED(1:time)*80,'green');  %max makes red lines
continue across top half of vertical axis
        plot(tLED(1:time),rLED(1:time)*80,'red');
        xlabel 'time (s)';
        ylabel 'amplitude (A.U.)';
        legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED');
        title(strcat(casename, ' spikes plot'));
        %     set(fig1, 'Position', [100, 100, 1920, 1080]);
        saveas(fig1, strcat(path, '/', casename, '-spikes.png'),'png');
        %     print(fig1, strcat(filename, '-spikes'),'-dpng');
        close(fig1);
    catch
    end

    %% Count spikes during each LED stimulation

    %     try
    %         SpikesL = sum(ui.spikes.*lLED(1:end-1));
    %         SpikesR = sum(ui.spikes.*rLED(1:end-1));
    %         disp(SpikesL);
    %         disp(SpikesR);
    %         %         disp(strcat('SpikeL = ', SpikesL));
    %         %         disp(strcat('SpikeR = ', SpikesR));
    %     catch
    %         disp('SpikesL_N/A');
    %         disp('SpikesR_N/A');
    %     end

    %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

    lightstim = 199; %change?
```

```
    try
        ui.leftLEDon = diff(lLED < -lightstim)>0.1;
        ui.rightLEDon = diff(rLED < -lightstim)>0.1;

        %        times.leftLED = find(leftLED == 500);
        %        times.rightLED = find(rightLED == 500);
        %rewrite:
        times.lLEDon = find(lLED == 1);
        % Gives all time points that left LED is on
        %result is a vector 1 x 80299
        times.lLEDoff = find(lLED == 0);
        % Gives all time points that left LED is off
        %result is a vector 1 x 1119941
        times.rLEDon = find(rLED == 1);
        times.rLEDoff = find(rLED == 0);
    catch
    end
% the above code will break out the time points when the LED is on for each
% side and when it is off. Next step:
    %         Need to ask it to count how many times ui.spikes takes place
    %         during each LEDon segment

    try
        if length(times.lLEDon) == 0 || length(times.rLEDon) == 0
            disp('WARNING!_Failed_to_detect_LED!')
            disp(filename);
            disp('----------------------');
        else
            %% gets light "on" times into one array
            times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
            %results in three specific time points for this LED
            times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
% this turns the times.xLEDon into a list of the points when the LED turned
            % on ***Use this for making a raster plot****
            %results in two specific time points for this LED

%% create raster plots-Left Stim
            % Error: Subscript indices must either be real ve integers or
            % logicals.
            for l = 1:length(times.lLEDstart)
                % collects window of data each time the light stimulus initiated
```

```
            windowSize = round(Fs*0.5); % window size in samples
            ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
        end

  %In original script, variable that's equivalent to 'times.lLEDon' is e.g.
        % a 23x1 double that includes only the start times for light turning
        % on. Maybe be better to use times.lLEDstart?
  %times.lLedon in this script gives the chunks when led was on, aka a
        %1x80299 double

        t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
        t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
        % creates time vector for raster

        times.Lrasterlight = 1:(length(times.lLEDstart));
        t.Lrasterlight =
ones(1,(length(times.lLEDstart)))*windowSize/Fs;
        % creates dashed line for indicating stim onset on raster plot

        ui.Lraster = horzcat(ui.Lrastercell{:});
        % concatenates cell array into a double

        Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
        % creates transform for stacking windowed data for the raster plot

%% create raster plots-Right Stim
        % Error: Subscript indices must either be real ve integers or
        % logicals.
        for r = 1:length(times.rLEDstart)
            % collects window of data each time the light stimulus initiated
            windowSize = round(Fs*0.5); % window size in samples
            ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
        end

        t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
        t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
        % creates time vector for raster

        times.Rrasterlight = 1:(length(times.rLEDstart));
```

```
          t.Rrasterlight =
ones(1,(length(times.rLEDstart)))*windowSize/Fs;
          % creates dashed line for indicating stim onset on raster plot

          ui.Rraster = horzcat(ui.Rrastercell{:});
          % concatenates cell array into a double

          Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
          % creates transform for stacking windowed data for the raster plot

%% Before running next part, need to find how many times light goes on,
% do this by checking the variable "times.lLEDstart" and "times.rLEDstart".
% It will either show the exact values for the start times or will indicate
% how many different light on times there are, if trials exceeds ~5.

          %           disp('LED light information: ');
          %           times
          LLEDtime = times.Lrasterlight(end);
          RLEDtime = times.Rrasterlight(end);
          disp(LLEDtime);
          disp(RLEDtime);

%% this number needs to be input as last value in reshape function below:

          ui.LrasterStack =
reshape(ui.Lraster,20001,length(times.Lrasterlight));
          ui.RrasterStack =
reshape(ui.Rraster,20001,length(times.Rrasterlight));

%% Check plot to verify reshape has been applied appropriately to LEFT data:

          fig2 = figure; % creates raster plot
          %    figure('units','normalized','position',[0 0 1 1]);
          %    figure('Visible','off');
          plot(t.Lraster,ui.LrasterStack+Lstack-1);
          hold on;
          line([0.5 0.5], [0 length(times.Lrasterlight)], 'Color', 'k',
'LineWidth',2)
          %    plot(t.Lrasterlight,times.Lrasterlight,'-black',
'LineWidth',8);
          hold off;
```

```
            ylabel 'trial number';
            xlabel 'time (s)';
            xlim([0 1]);
            %           ylim([0:1:LLEDtime]);
            title(strcat(casename, ' Lraster plot'));
            saveas(fig2, strcat(path, '/', casename, '-Lraster.png'),'png');
            %    print(fig2, strcat(filename, '-Lraster'),'-dpng');
            close(fig2);

 %% Check plot to verify reshape has been applied appropriately to RIGHT data:

            fig3 = figure; % creates raster plot
            %    figure('units','normalized','position',[0 0 1 1]);
            %    figure('Visible','off');
            plot(t.Rraster,ui.RrasterStack+Rstack-1);
            hold on;
            line([0.5 0.5], [0 length(times.Rrasterlight)], 'Color', 'k',
'LineWidth',2)
            hold off;
            ylabel 'trial number';
            xlabel 'time (s)';
            xlim([0 1]);
            %           ylim([0:1:LLEDtime]);
            title(strcat(casename, ' Rraster plot'));
            saveas(fig3, strcat(path, '/', casename, '-Rraster.png'),'png');
            %    print(fig3, strcat(filename, '-Rraster'),'-dpng');
            close(fig3);

            %% Lastly, get spike averages for each eye
            stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
            %    % calculates average number of spikes after light turned on
            stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
            %    % calculates average number of spikes preceding light onset

            %           disp('For left eye (L):');
            %           disp(strcat('spike average (on):',
stats.spikes.Laveon));
            %           disp(strcat('spike average (off):',
stats.spikes.Laveoff));
```

```
            LSpikeOn = stats.spikes.Laveon;
            LSpikeOff = stats.spikes.Laveoff;
            disp(LSpikeOn);
            disp(LSpikeOff);

            %% Lastly, get spike avwerages for each eye
            stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
            %     % calculates average number of spikes after light turned on
            stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
            %     % calculates average number of spikes preceding light onset
            %
            %           disp('For right eye (R):');
            %           disp(strcat('spike average (on):',
stats.spikes.Raveon));
            %           disp(strcat('spike average (off):',
stats.spikes.Raveoff));
            RSpikeOn = stats.spikes.Raveon;
            RSpikeOff = stats.spikes.Raveoff;
            disp(RSpikeOn);
            disp(RSpikeOff);

            %           disp(strcat('Finished!!!', filename));
            %           disp(strcat(SpikesL, SpikesR, LLEDtime, RLEDtime,
LSpikeOn, LSpikeOff, RSpikeOn, RSpikeOff));
            disp('----------------------');

        end
    catch
    end

%% clear all %Starting a new analysis so we want to eliminate all old variables
    close all;
    clearvars -except path list files trials trial index;

end

system(['cd ' path '/output-' date]);

diary off;
```

2. fast_arrange_Intan_RHD.m

```
% Modified by Baihan Lin
% to extract information from Intan Tech recording data
% Apr 2016

function fast_arrange_Intan_RHD(file)

% Built upon read_Intan_RHD2000_file
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%

tic;
fid = fopen(file, 'r');

s = dir(file);
filesize = s.bytes;

% Check 'magic number' at beginning of file to make sure this is an Intan
% Technologies RHD2000 data file.
magic_number = fread(fid, 1, 'uint32');
if magic_number ~= hex2dec('c6912702')
    error('Unrecognized file type.');
end

% Read version number.
data_file_main_version_number = fread(fid, 1, 'int16');
data_file_secondary_version_number = fread(fid, 1, 'int16');

% fprintf(1, '\n');
% fprintf(1, 'Reading Intan Technologies RHD2000 Data File,
Version %d.%d\n', ...
```

```
%    data_file_main_version_number, data_file_secondary_version_number);
% fprintf(1, '\n');

% Read information of sampling rate and amplifier frequency settings.
sample_rate = fread(fid, 1, 'single');
dsp_enabled = fread(fid, 1, 'int16');
actual_dsp_cutoff_frequency = fread(fid, 1, 'single');
actual_lower_bandwidth = fread(fid, 1, 'single');
actual_upper_bandwidth = fread(fid, 1, 'single');

desired_dsp_cutoff_frequency = fread(fid, 1, 'single');
desired_lower_bandwidth = fread(fid, 1, 'single');
desired_upper_bandwidth = fread(fid, 1, 'single');

% This tells us if a software 50/60 Hz notch filter was enabled during
% the data acquisition.
notch_filter_mode = fread(fid, 1, 'int16');
notch_filter_frequency = 0;
if (notch_filter_mode == 1)
    notch_filter_frequency = 50;
elseif (notch_filter_mode == 2)
    notch_filter_frequency = 60;
end

desired_impedance_test_frequency = fread(fid, 1, 'single');
actual_impedance_test_frequency = fread(fid, 1, 'single');

% Place notes in data strucure
notes = struct( ...
    'note1', fread_QString(fid), ...
    'note2', fread_QString(fid), ...
    'note3', fread_QString(fid) );

% If data file is from GUI v1.1 or later, see if temperature sensor data
% was saved.
num_temp_sensor_channels = 0;
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 1) ...
    || (data_file_main_version_number > 1))
    num_temp_sensor_channels = fread(fid, 1, 'int16');
end
```

```
% If data file is from GUI v1.3 or later, load eval board mode.
eval_board_mode = 0;
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 3) ...
    || (data_file_main_version_number > 1))
    eval_board_mode = fread(fid, 1, 'int16');
end

% Place frequency-related information in data structure.
frequency_parameters = struct( ...
    'amplifier_sample_rate', sample_rate, ...
    'aux_input_sample_rate', sample_rate / 4, ...
    'supply_voltage_sample_rate', sample_rate / 60, ...
    'board_adc_sample_rate', sample_rate, ...
    'board_dig_in_sample_rate', sample_rate, ...
    'desired_dsp_cutoff_frequency', desired_dsp_cutoff_frequency, ...
    'actual_dsp_cutoff_frequency', actual_dsp_cutoff_frequency, ...
    'dsp_enabled', dsp_enabled, ...
    'desired_lower_bandwidth', desired_lower_bandwidth, ...
    'actual_lower_bandwidth', actual_lower_bandwidth, ...
    'desired_upper_bandwidth', desired_upper_bandwidth, ...
    'actual_upper_bandwidth', actual_upper_bandwidth, ...
    'notch_filter_frequency', notch_filter_frequency, ...
    'desired_impedance_test_frequency',
desired_impedance_test_frequency, ...
    'actual_impedance_test_frequency', actual_impedance_test_frequency );

% Define data structure for spike trigger settings.
spike_trigger_struct = struct( ...
    'voltage_trigger_mode', {}, ...
    'voltage_threshold', {}, ...
    'digital_trigger_channel', {}, ...
    'digital_edge_polarity', {} );

new_trigger_channel = struct(spike_trigger_struct);
spike_triggers = struct(spike_trigger_struct);

% Define data structure for data channels.
channel_struct = struct( ...
    'native_channel_name', {}, ...
```

```
    'custom_channel_name', {}, ...
    'native_order', {}, ...
    'custom_order', {}, ...
    'board_stream', {}, ...
    'chip_channel', {}, ...
    'port_name', {}, ...
    'port_prefix', {}, ...
    'port_number', {}, ...
    'electrode_impedance_magnitude', {}, ...
    'electrode_impedance_phase', {} );

new_channel = struct(channel_struct);

% Create structure arrays for each type of data channel.
amplifier_channels = struct(channel_struct);
aux_input_channels = struct(channel_struct);
supply_voltage_channels = struct(channel_struct);
board_adc_channels = struct(channel_struct);
board_dig_in_channels = struct(channel_struct);
board_dig_out_channels = struct(channel_struct);

amplifier_index = 1;
aux_input_index = 1;
supply_voltage_index = 1;
board_adc_index = 1;
board_dig_in_index = 1;
board_dig_out_index = 1;

% Read signal summary from data file header.

number_of_signal_groups = fread(fid, 1, 'int16');

for signal_group = 1:number_of_signal_groups
    signal_group_name = fread_QString(fid);
    signal_group_prefix = fread_QString(fid);
    signal_group_enabled = fread(fid, 1, 'int16');
    signal_group_num_channels = fread(fid, 1, 'int16');
    signal_group_num_amp_channels = fread(fid, 1, 'int16');

    if (signal_group_num_channels > 0 && signal_group_enabled > 0)
        new_channel(1).port_name = signal_group_name;
```

```
        new_channel(1).port_prefix = signal_group_prefix;
        new_channel(1).port_number = signal_group;
        for signal_channel = 1:signal_group_num_channels
            new_channel(1).native_channel_name = fread_QString(fid);
            new_channel(1).custom_channel_name = fread_QString(fid);
            new_channel(1).native_order = fread(fid, 1, 'int16');
            new_channel(1).custom_order = fread(fid, 1, 'int16');
            signal_type = fread(fid, 1, 'int16');
            channel_enabled = fread(fid, 1, 'int16');
            new_channel(1).chip_channel = fread(fid, 1, 'int16');
            new_channel(1).board_stream = fread(fid, 1, 'int16');
            new_trigger_channel(1).voltage_trigger_mode = fread(fid, 1,
'int16');
            new_trigger_channel(1).voltage_threshold = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_trigger_channel = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_edge_polarity = fread(fid, 1,
'int16');
            new_channel(1).electrode_impedance_magnitude = fread(fid, 1,
'single');
            new_channel(1).electrode_impedance_phase = fread(fid, 1,
'single');

            if (channel_enabled)
                switch (signal_type)
                    case 0
                        amplifier_channels(amplifier_index) = new_channel;
                        spike_triggers(amplifier_index) =
new_trigger_channel;
                        amplifier_index = amplifier_index + 1;
                    case 1
                        aux_input_channels(aux_input_index) = new_channel;
                        aux_input_index = aux_input_index + 1;
                    case 2
                        supply_voltage_channels(supply_voltage_index) =
new_channel;
                        supply_voltage_index = supply_voltage_index + 1;
                    case 3
                        board_adc_channels(board_adc_index) = new_channel;
                        board_adc_index = board_adc_index + 1;
```

```
                case 4
                    board_dig_in_channels(board_dig_in_index) =
new_channel;
                    board_dig_in_index = board_dig_in_index + 1;
                case 5
                    board_dig_out_channels(board_dig_out_index) =
new_channel;
                    board_dig_out_index = board_dig_out_index + 1;
                otherwise
                    error('Unknown channel type');
            end
        end

    end
  end
end

% Summarize contents of data file.
num_amplifier_channels = amplifier_index - 1;
num_aux_input_channels = aux_input_index - 1;
num_supply_voltage_channels = supply_voltage_index - 1;
num_board_adc_channels = board_adc_index - 1;
num_board_dig_in_channels = board_dig_in_index - 1;
num_board_dig_out_channels = board_dig_out_index - 1;

% fprintf(1, 'Found %d amplifier channel%s.\n', ...
%     num_amplifier_channels, plural(num_amplifier_channels));
% fprintf(1, 'Found %d auxiliary input channel%s.\n', ...
%     num_aux_input_channels, plural(num_aux_input_channels));
% fprintf(1, 'Found %d supply voltage channel%s.\n', ...
%     num_supply_voltage_channels, plural(num_supply_voltage_channels));
% fprintf(1, 'Found %d board ADC channel%s.\n', ...
%     num_board_adc_channels, plural(num_board_adc_channels));
% fprintf(1, 'Found %d board digital input channel%s.\n', ...
%     num_board_dig_in_channels, plural(num_board_dig_in_channels));
% fprintf(1, 'Found %d board digital output channel%s.\n', ...
%     num_board_dig_out_channels, plural(num_board_dig_out_channels));
% fprintf(1, 'Found %d temperature sensors channel%s.\n', ...
%     num_temp_sensor_channels, plural(num_temp_sensor_channels));
% fprintf(1, '\n');
```

```
% Determine how many samples the data file contains.

% Each data block contains 60 amplifier samples.
bytes_per_block = 60 * 4;  % timestamp data
bytes_per_block = bytes_per_block + 60 * 2 * num_amplifier_channels;
% Auxiliary inputs are sampled 4x slower than amplifiers
bytes_per_block = bytes_per_block + 15 * 2 * num_aux_input_channels;
% Supply voltage is sampled 60x slower than amplifiers
bytes_per_block = bytes_per_block + 1 * 2 * num_supply_voltage_channels;
% Board analog inputs are sampled at same rate as amplifiers
bytes_per_block = bytes_per_block + 60 * 2 * num_board_adc_channels;
% Board digital inputs are sampled at same rate as amplifiers
if (num_board_dig_in_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Board digital outputs are sampled at same rate as amplifiers
if (num_board_dig_out_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Temp sensor is sampled 60x slower than amplifiers
if (num_temp_sensor_channels > 0)
   bytes_per_block = bytes_per_block + 1 * 2 * num_temp_sensor_channels;
end

% How many data blocks remain in this file?
data_present = 0;
bytes_remaining = filesize - ftell(fid);
if (bytes_remaining > 0)
    data_present = 1;
end

num_data_blocks = bytes_remaining / bytes_per_block;

num_amplifier_samples = 60 * num_data_blocks;
num_aux_input_samples = 15 * num_data_blocks;
num_supply_voltage_samples = 1 * num_data_blocks;
num_board_adc_samples = 60 * num_data_blocks;
num_board_dig_in_samples = 60 * num_data_blocks;
num_board_dig_out_samples = 60 * num_data_blocks;

record_time = num_amplifier_samples / sample_rate;
```

```
% if (data_present)
%     fprintf(1, 'File contains %0.3f seconds of data.  Amplifiers were
sampled at %0.2f kS/s.\n', ...
%         record_time, sample_rate / 1000);
%     fprintf(1, '\n');
% else
%     fprintf(1, 'Header file contains no data.  Amplifiers were sampled
at %0.2f kS/s.\n', ...
%         sample_rate / 1000);
%     fprintf(1, '\n');
% end

if (data_present)

    % Pre-allocate memory for data.
%     fprintf(1, 'Allocating memory for data...\n');

    t_amplifier = zeros(1, num_amplifier_samples);

    amplifier_data = zeros(num_amplifier_channels, num_amplifier_samples);
    aux_input_data = zeros(num_aux_input_channels, num_aux_input_samples);
    supply_voltage_data = zeros(num_supply_voltage_channels,
num_supply_voltage_samples);
    temp_sensor_data = zeros(num_temp_sensor_channels,
num_supply_voltage_samples);
    board_adc_data = zeros(num_board_adc_channels, num_board_adc_samples);
    board_dig_in_data = zeros(num_board_dig_in_channels,
num_board_dig_in_samples);
    board_dig_in_raw = zeros(1, num_board_dig_in_samples);
    board_dig_out_data = zeros(num_board_dig_out_channels,
num_board_dig_out_samples);
    board_dig_out_raw = zeros(1, num_board_dig_out_samples);

    % Read sampled data from file.
%     fprintf(1, 'Reading data from file...\n');

    amplifier_index = 1;
    aux_input_index = 1;
    supply_voltage_index = 1;
    board_adc_index = 1;
```

```
    board_dig_in_index = 1;
    board_dig_out_index = 1;

    print_increment = 10;
    percent_done = print_increment;
    for i=1:num_data_blocks
        % In version 1.2, we moved from saving timestamps as unsigned
        % integeters to signed integers to accomidate negative (adjusted)
        % timestamps for pretrigger data.
        if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 2) ...
        || (data_file_main_version_number > 1))
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'int32');
        else
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'uint32');
        end
        if (num_amplifier_channels > 0)
            amplifier_data(:, amplifier_index:(amplifier_index+59)) =
fread(fid, [60, num_amplifier_channels], 'uint16')';
        end
        if (num_aux_input_channels > 0)
            aux_input_data(:, aux_input_index:(aux_input_index+14)) =
fread(fid, [15, num_aux_input_channels], 'uint16')';
        end
        if (num_supply_voltage_channels > 0)
            supply_voltage_data(:, supply_voltage_index) = fread(fid, [1,
num_supply_voltage_channels], 'uint16')';
        end
        if (num_temp_sensor_channels > 0)
            temp_sensor_data(:, supply_voltage_index) = fread(fid, [1,
num_temp_sensor_channels], 'int16')';
        end
        if (num_board_adc_channels > 0)
            board_adc_data(:, board_adc_index:(board_adc_index+59)) =
fread(fid, [60, num_board_adc_channels], 'uint16')';
        end
        if (num_board_dig_in_channels > 0)
            board_dig_in_raw(board_dig_in_index:(board_dig_in_index+59)) =
fread(fid, 60, 'uint16');
```

```
        end
        if (num_board_dig_out_channels > 0)

board_dig_out_raw(board_dig_out_index:(board_dig_out_index+59)) =
fread(fid, 60, 'uint16');
        end

        amplifier_index = amplifier_index + 60;
        aux_input_index = aux_input_index + 15;
        supply_voltage_index = supply_voltage_index + 1;
        board_adc_index = board_adc_index + 60;
        board_dig_in_index = board_dig_in_index + 60;
        board_dig_out_index = board_dig_out_index + 60;

        fraction_done = 100 * (i / num_data_blocks);
        if (fraction_done >= percent_done)
%           fprintf(1, '%d%% done...\n', percent_done);
           percent_done = percent_done + print_increment;
        end
    end

    % Make sure we have read exactly the right amount of data.
    bytes_remaining = filesize - ftell(fid);
    if (bytes_remaining ~= 0)
        %error('Error: End of file not reached.');
    end

end

% Close data file.
fclose(fid);

if (data_present)

%    fprintf(1, 'Parsing data...\n');

    % Extract digital input channels to separate variables.
    for i=1:num_board_dig_in_channels
      mask = 2^(board_dig_in_channels(i).native_order) *
ones(size(board_dig_in_raw));
      board_dig_in_data(i, :) = (bitand(board_dig_in_raw, mask) > 0);
```

```
    end
    for i=1:num_board_dig_out_channels
      mask = 2^(board_dig_out_channels(i).native_order) *
ones(size(board_dig_out_raw));
      board_dig_out_data(i, :) = (bitand(board_dig_out_raw, mask) > 0);
    end

    % Scale voltage levels appropriately.
    amplifier_data = 0.195 * (amplifier_data - 32768); % units = microvolts
    aux_input_data = 37.4e-6 * aux_input_data; % units = volts
    supply_voltage_data = 74.8e-6 * supply_voltage_data; % units = volts
    if (eval_board_mode == 1)
        board_adc_data = 152.59e-6 * (board_adc_data - 32768); % units = volts
    else
        board_adc_data = 50.354e-6 * board_adc_data; % units = volts
    end
    temp_sensor_data = temp_sensor_data / 100; % units = deg C

    % Check for gaps in timestamps.
    num_gaps = sum(diff(t_amplifier) ~= 1);
%    if (num_gaps == 0)
%        fprintf(1, 'No missing timestamps in data.\n');
%    else
%        fprintf(1, 'Warning: %d gaps in timestamp data found.  Time scale
will not be uniform!\n', ...
%            num_gaps);
%    end

    % Scale time steps (units = seconds).
    t_amplifier = t_amplifier / sample_rate;
    t_aux_input = t_amplifier(1:4:end);
    t_supply_voltage = t_amplifier(1:60:end);
    t_board_adc = t_amplifier;
    t_dig = t_amplifier;
    t_temp_sensor = t_supply_voltage;

    % If the software notch filter was selected during the recording, apply
the
    % same notch filter to amplifier data here.
    if (notch_filter_frequency > 0)
%        fprintf(1, 'Applying notch filter...\n');
```

```
        print_increment = 10;
        percent_done = print_increment;
        for i=1:num_amplifier_channels
            amplifier_data(i,:) = ...
                notch_filter(amplifier_data(i,:), sample_rate,
notch_filter_frequency, 10);

            fraction_done = 100 * (i / num_amplifier_channels);
            if (fraction_done >= percent_done)
%                 fprintf(1, '%d%% done...\n', percent_done);
                percent_done = percent_done + print_increment;
            end

        end
    end

end

% Move variables to base workspace.

move_to_base_workspace(notes);
move_to_base_workspace(frequency_parameters);

if (num_amplifier_channels > 0)
    move_to_base_workspace(amplifier_channels);
    if (data_present)
        move_to_base_workspace(amplifier_data);
        move_to_base_workspace(t_amplifier);
    end
    move_to_base_workspace(spike_triggers);
end
if (num_aux_input_channels > 0)
    move_to_base_workspace(aux_input_channels);
    if (data_present)
        move_to_base_workspace(aux_input_data);
        move_to_base_workspace(t_aux_input);
    end
end
if (num_supply_voltage_channels > 0)
    move_to_base_workspace(supply_voltage_channels);
```

```
    if (data_present)
        move_to_base_workspace(supply_voltage_data);
        move_to_base_workspace(t_supply_voltage);
    end
end
if (num_board_adc_channels > 0)
    move_to_base_workspace(board_adc_channels);
    if (data_present)
        move_to_base_workspace(board_adc_data);
        move_to_base_workspace(t_board_adc);
    end
end
if (num_board_dig_in_channels > 0)
    move_to_base_workspace(board_dig_in_channels);
    if (data_present)
        move_to_base_workspace(board_dig_in_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_board_dig_out_channels > 0)
    move_to_base_workspace(board_dig_out_channels);
    if (data_present)
        move_to_base_workspace(board_dig_out_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_temp_sensor_channels > 0)
    if (data_present)
        move_to_base_workspace(temp_sensor_data);
        move_to_base_workspace(t_temp_sensor);
    end
end

% fprintf(1, 'Done!  Elapsed time: %0.1f seconds\n', toc);
% if (data_present)
%     fprintf(1, 'Extracted data are now available in the MATLAB
workspace.\n');
% else
%     fprintf(1, 'Extracted waveform information is now available in the
MATLAB workspace.\n');
% end
```

```
% fprintf(1, 'Type ''whos'' to see variables.\n');
% fprintf(1, '\n');

return


function a = fread_QString(fid)

% a = read_QString(fid)
%
% Read Qt style QString.  The first 32-bit unsigned number indicates
% the length of the string (in bytes).  If this number equals 0xFFFFFFFF,
% the string is null.

a = '';
length = fread(fid, 1, 'uint32');
if length == hex2num('ffffffff')
   return;
end
% convert length from bytes to 16-bit Unicode words
length = length / 2;

for i=1:length
   a(i) = fread(fid, 1, 'uint16');
end

return


function s = plural(n)

% s = plural(n)
%
% Utility function to optionally plurailze words based on the value
% of n.

if (n == 1)
   s = '';
else
   s = 's';
end
```

```
return


function out = notch_filter(in, fSample, fNotch, Bandwidth)

% out = notch_filter(in, fSample, fNotch, Bandwidth)
%
% Implements a notch filter (e.g., for 50 or 60 Hz) on vector 'in'.
% fSample = sample rate of data (in Hz or Samples/sec)
% fNotch = filter notch frequency (in Hz)
% Bandwidth = notch 3-dB bandwidth (in Hz).  A bandwidth of 10 Hz is
%   recommended for 50 or 60 Hz notch filters; narrower bandwidths lead to
%   poor time-domain properties with an extended ringing response to
%   transient disturbances.
%
% Example:  If neural data was sampled at 30 kSamples/sec
% and you wish to implement a 60 Hz notch filter:
%
% out = notch_filter(in, 30000, 60, 10);

tstep = 1/fSample;
Fc = fNotch*tstep;

L = length(in);

% Calculate IIR filter parameters
d = exp(-2*pi*(Bandwidth/2)*tstep);
b = (1 + d*d)*cos(2*pi*Fc);
a0 = 1;
a1 = -b;
a2 = d*d;
a = (1 + d*d)/2;
b0 = 1;
b1 = -2*cos(2*pi*Fc);
b2 = 1;

out = zeros(size(in));
out(1) = in(1);
out(2) = in(2);
% (If filtering a continuous data stream, change out(1) and out(2) to the
```

```
%  previous final two values of out.)

% Run filter
for i=3:L
    out(i) = (a*b2*in(i-2) + a*b1*in(i-1) + a*b0*in(i) - a2*out(i-2) -
a1*out(i-1))/a0;
end

return


function move_to_base_workspace(variable)

% move_to_base_workspace(variable)
%
% Move variable from function workspace to base MATLAB workspace so
% user will have access to it after the program ends.

variable_name = inputname(1);
assignin('base', variable_name, variable);

return;
```

3. final_160430_IntanEphysAnalysis.m

```
% Modified by Baihan Lin
% to analyze the extracted information from Intan Tech recording data
% Apr 2016

clear all %Starting a new analysis so we want to eliminate all old variables
close all


%% Begin with opening file to be analyzed
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
```

```
%
% read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.


% read_Intan_RHD2000_file_combine
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%
% Example:
% >> clear
% >> read_Intan_RHD200_file
% >> whos
% >> amplifier_channels(1)
% >> plot(t_amplifier, amplifier_data(1,:))

% Here I change it from:
%[file, path, filterindex] = uigetfile('*.rhd', 'Select an RHD2000 Data File',
'MultiSelect', 'off');

% Read most recent file automatically.
%path = 'C:\Users\Reid\Documents\RHD2132\testing\';
%d = dir([path '*.rhd']);
%file = d(end).name;

% I decided to use automatic method for data collection:

prompt = 'What is your folder?: ';
disp('e.g.  /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data');
```

```
path = input(prompt,'s');
% /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test

% So that I can specify a folder to access all data files.

%[status, list] = system('cd path');

[~,list] = system(['find ' path ' -type f -name "*.rhd"']);

system(['mkdir ' path '/output-' date]);
system(['cd ' path '/output-' date]);

diary(strcat(path, '/output-', date, '/report_', date, '.out'));
diary on;
disp(path);
disp(date);

warning('off','MATLAB:xlswrite:AddSheet');
xlsfile = strcat(path, '/report-', date, '.xlsx');
mycell = {'Excel'};
xlswrite(xlsfile,mycell);
% rpt = {strcat('report-', date, ' by Baihan Lin')};
% xlswrite(xlsfile, rpt(1),'Report','A1');



files = strsplit(list);
length(files);

for countfile = 1:length(files)
    trials{countfile} = files{countfile}(1:end-11);
end

trials = unique(trials);
trials = trials(~cellfun('isempty',trials));

index = strfind(files, trials{1});

for trial = 1 : length(trials)
    filename = trials{trial};
    index = strfind(files, filename);
```

```
indexmat = cell2mat(index);
[~,first,~] = unique(indexmat, 'first');
arrange_Intan_RHD(files{first(1)});

amp_data = amplifier_data;
try
    ai_data = aux_input_data;
    bdi_data = board_dig_in_data;
    sv_data = supply_voltage_data;
    t_ai = t_aux_input;
    t_d = t_dig;
    t_sv = t_supply_voltage;
catch exception
end
t_amp = t_amplifier;

if length(index) > 1
    for ind = 2 : length(index)
        if index{ind} == 1
            arrange_Intan_RHD(files{ind});
            try

                amp_data = [amp_data, amplifier_data];
                ai_data = [ai_data, aux_input_data];
                bdi_data = [bdi_data, board_dig_in_data];
                sv_data = [sv_data, supply_voltage_data];
                t_amp = [t_amp, t_amplifier];
                t_ai =  [t_ai,t_aux_input];
                t_d = [t_d, t_dig];
                t_sv = [t_sv, t_supply_voltage];


            catch exception

            end
        end
    end
end

amplifier_data = amp_data;
t_amplifier = t_amp;
```

```
    try
        aux_input_data = ai_data;
        board_dig_in_data = bdi_data;
        supply_voltage_data = sv_data;
        t_aux_input = t_ai;
        t_dig = t_d;
        t_supply_voltage = t_sv;
    catch
    end

    %check what variables have been imported, especially if youre unsure
    %whether accessory amplifier channels were disabled or not during
recording

    %% Establishing some basic variables from values pulled in by above
function

    amp_chan = 1;
    amp_imp = amplifier_channels(1).electrode_impedance_magnitude;

    for chan = 1 : length(amplifier_channels)
        if amplifier_channels(chan).electrode_impedance_magnitude < amp_imp
            amp_chan = chan;
            amp_imp =
amplifier_channels(chan).electrode_impedance_magnitude;
        end
    end

    disp('amplifer information');
    amplifier_channels(amp_chan)

    %Channel data is being collected on (on
    %preamplifier this would be 'A-004'
    %Above command gives data output about the channel on which data is being
    %collected

    %% Variable name changes below to simplify:

    try
        tRat = t_amplifier; %time variable for ephys data
        tLED = t_dig; %time variable for LED data
```

```
        try
            ui.ratData = amplifier_data(amp_chan,:);
        catch exception
            ui.ratData = amplifier_data(1,:);
        end
        lLED = board_dig_in_data(1,:);
        rLED = board_dig_in_data(2,:);
    catch
    end

    % %% Check variables look right-- plot should be identical to last one
    % figure % for spike detection
    %    hold on
    %    plot(tRat, ui.ratData,'blue')
    %    plot(tLED,lLED,'red')  %max makes red lines continue across top
half of vertical axis
    %    plot(tLED,rLED,'green')
    %    xlabel 'time (s)'
    %    ylabel 'amplitude (A.U.)'
    %    legend('Raw Data', 'Left Eye LED','Right Eye LED')

    %% filter data
    Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    try
        ui.ratData = filtfilt(b,a,ui.ratData);
        %% invert signal data for thresholding
        ui.ratData = ui.ratData.*(-1);
    catch
    end

    %% Setting threshold for spikes and finding light ON times
    threshold = 25;
    Fs =  20000; %amplifier_sample_rate
    windowSize = Fs * 0.05; %creates our time interval by taking the 20k
    % sampling rate at which the data was collected and converts it to
    % timestamps collected every millisecond, in other words the value of
    % windowSize is 1 ms.
```

```
    % Finding all spikes in recording:
    try
        ui.spikes = diff(ui.ratData > threshold) > 0.1;
    catch
    end


    %% Plot Again with spikes showing & correct time axis now:

    try
        time = length(tRat)-1;
        fig1 = figure; % for spike detection
        %     figure('units','normalized','position',[0 0 1 1]);
        %     figure('Visible','off');
        hold on
        plot(tRat(1:time), ui.ratData(1:time),'blue');
        plot(tRat(1:time), ui.spikes*max(ui.ratData),'black');
        plot(tLED(1:time),lLED(1:time)*80,'green');  %max makes red lines
continue across top half of vertical axis
        plot(tLED(1:time),rLED(1:time)*80,'red');
        xlabel 'time (s)';
        ylabel 'amplitude (A.U.)';
        legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED');
        %     set(fig1, 'Position', [100, 100, 1920, 1080]);
        saveas(fig1, strcat(filename, '-spikes.png'),'png');
        %     print(fig1, strcat(filename, '-spikes'),'-dpng');
        close(fig1);
    catch
    end

    %% Count spikes during each LED stimulation

    try
        SpikesL = sum(ui.spikes.*lLED(1:end-1));
        SpikesR = sum(ui.spikes.*rLED(1:end-1));

        disp(strcat('spikeL = ', SpikesL));
        disp(strcat('spikeR = ', SpikesR));
    catch
    end
```

```
    %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

    lightstim = 199; %change?
    try
        ui.leftLEDon = diff(lLED < -lightstim)>0.1;
        ui.rightLEDon = diff(rLED < -lightstim)>0.1;

        %        times.leftLED = find(leftLED == 500);
        %        times.rightLED = find(rightLED == 500);
        %rewrite:
        times.lLEDon = find(lLED == 1);
        % Gives all time points that left LED is on
        %result is a vector 1 x 80299
        times.lLEDoff = find(lLED == 0);
        % Gives all time points that left LED is off
        %result is a vector 1 x 1119941
        times.rLEDon = find(rLED == 1);
        times.rLEDoff = find(rLED == 0);
    catch
    end
    % the above code will break out the time points when the LED is on for
each
    % side and when it is off. Next step:
    %         Need to ask it to count how many times ui.spikes takes place
    %         during each LEDon segment

    try
        if length(times.lLEDon) == 0 || length(times.rLEDon) == 0
            disp('WARNING! Failed to detect LED!')
            disp(filename);
            disp('----------------------');
        else



            %% gets light "on" times into one array
            times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
            %results in three specific time points for this LED
            times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
```

```
        % this turns the times.xLEDon into a list of the points when the
LED turned
        % on ***Use this for making a raster plot****
        %results in two specific time points for this LED


        %% create raster plots-Left Stim
        % Error: Subscript indices must either be real ve integers or
        % logicals.
        for l = 1:length(times.lLEDstart)
            % collects window of data each time the light stimulus initiated
            windowSize = round(Fs*0.5); % window size in samples
            ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
        end

        %In original script, variable that's equivalent to 'times.lLEDon'
is e.g.
        % a 23x1 double that includes only the start times for light turning
        % on. Maybe be better to use times.lLEDstart?
        %times.lLedon in this script gives the chunks when led was on,
aka a
        %1x80299 double

        t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
        t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
        % creates time vector for raster

        times.Lrasterlight = 1:(length(times.lLEDstart));
        t.Lrasterlight =
ones(1,(length(times.lLEDstart)))*windowSize/Fs;
        % creates dashed line for indicating stim onset on raster plot

        ui.Lraster = horzcat(ui.Lrastercell{:});
        % concatenates cell array into a double

        Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
        % creates transform for stacking windowed data for the raster plot


        %% create raster plots-Right Stim
```

```
        % Error: Subscript indices must either be real ve integers or
        % logicals.
        for r = 1:length(times.rLEDstart)
            % collects window of data each time the light stimulus initiated
            windowSize = round(Fs*0.5); % window size in samples
            ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
        end


        t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
        t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
        % creates time vector for raster

        times.Rrasterlight = 1:(length(times.rLEDstart));
        t.Rrasterlight =
ones(1,(length(times.rLEDstart)))*windowSize/Fs;
        % creates dashed line for indicating stim onset on raster plot

        ui.Rraster = horzcat(ui.Rrastercell{:});
        % concatenates cell array into a double

        Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
        % creates transform for stacking windowed data for the raster plot


        %% Before running next part, need to find how many times light
goes on,
        % do this by checking the variable "times.lLEDstart" and
"times.rLEDstart".
        % It will either show the exact values for the start times or will
indicate
        % how many different light on times there are, if trials exceeds
~5.

        disp('LED light information: ');
        times


        %% this number needs to be input as last value in reshape function
below:
```

```
        ui.LrasterStack =
reshape(ui.Lraster,20001,length(times.Lrasterlight));
        ui.RrasterStack =
reshape(ui.Rraster,20001,length(times.Rrasterlight));
        %% Check plot to verify reshape has been applied appropriately
to LEFT data:

        fig2 = figure % creates raster plot
        %    figure('units','normalized','position',[0 0 1 1]);
        %    figure('Visible','off');
        plot(t.Lraster,ui.LrasterStack+Lstack-1);
        hold on
        line([0.5 0.5], [0 length(times.Lrasterlight)], 'Color', 'k',
'LineWidth',2)
        %    plot(t.Lrasterlight,times.Lrasterlight,'-black',
'LineWidth',8);
        hold off
        ylabel 'trial number';
        xlabel 'time (s)';
        xlim([0 1]);
        saveas(fig2, strcat(filename, '-Lraster.png'),'png');
        %    print(fig2, strcat(filename, '-Lraster'),'-dpng');
        close(fig2);

        %% Check plot to verify reshape has been applied appropriately
to RIGHT data:

        fig3 = figure % creates raster plot
        %    figure('units','normalized','position',[0 0 1 1]);
        %    figure('Visible','off');
        plot(t.Rraster,ui.RrasterStack+Rstack-1);
        hold on
        line([0.5 0.5], [0 length(times.Rrasterlight)], 'Color', 'k',
'LineWidth',2)
        hold off
        ylabel 'trial number';
        xlabel 'time (s)';
        xlim([0 1]);
        saveas(fig3, strcat(filename, '-Rraster.png'),'png');
```

```
            %     print(fig3, strcat(filename, '-Rraster'),'-dpng');
            close(fig3);

            %% Lastly, get spike averages for each eye
            stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
            %     % calculates average number of spikes after light turned on
            stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
            %     % calculates average number of spikes preceding light onset

            disp('For left eye (L):');
            disp(strcat('spike average (on):', stats.spikes.Laveon));
            disp(strcat('spike average (off):', stats.spikes.Laveoff));


            %% Lastly, get spike avwerages for each eye
            stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
            %     % calculates average number of spikes after light turned on
            stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
            %     % calculates average number of spikes preceding light onset

            disp('For right eye (R):');
            disp(strcat('spike average (on):', stats.spikes.Raveon));
            disp(strcat('spike average (off):', stats.spikes.Raveoff));

            disp(strcat('Finished!!!', filename));
            disp('--------------------------------');

        end
    catch
    end
    %%

    %     clear all %Starting a new analysis so we want to eliminate all old
variables
    close all
    clearvars -except path list files trials trial index;
```

```
end

system(['cd ' path '/output-' date]);

diary off;
```

4. arrange_Intan_RHD.m

```
function arrange_Intan_RHD(file)

% Modified by Baihan Lin
% Apr 2016

% Built upon read_Intan_RHD2000_file
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%

tic;
fid = fopen(file, 'r');

s = dir(file);
filesize = s.bytes;

% Check 'magic number' at beginning of file to make sure this is an Intan
% Technologies RHD2000 data file.
magic_number = fread(fid, 1, 'uint32');
if magic_number ~= hex2dec('c6912702')
    error('Unrecognized file type.');
end

% Read version number.
data_file_main_version_number = fread(fid, 1, 'int16');
```

```
data_file_secondary_version_number = fread(fid, 1, 'int16');

fprintf(1, '\n');
fprintf(1, 'Reading Intan Technologies RHD2000 Data File,
Version %d.%d\n', ...
    data_file_main_version_number, data_file_secondary_version_number);
fprintf(1, '\n');

% Read information of sampling rate and amplifier frequency settings.
sample_rate = fread(fid, 1, 'single');
dsp_enabled = fread(fid, 1, 'int16');
actual_dsp_cutoff_frequency = fread(fid, 1, 'single');
actual_lower_bandwidth = fread(fid, 1, 'single');
actual_upper_bandwidth = fread(fid, 1, 'single');

desired_dsp_cutoff_frequency = fread(fid, 1, 'single');
desired_lower_bandwidth = fread(fid, 1, 'single');
desired_upper_bandwidth = fread(fid, 1, 'single');

% This tells us if a software 50/60 Hz notch filter was enabled during
% the data acquisition.
notch_filter_mode = fread(fid, 1, 'int16');
notch_filter_frequency = 0;
if (notch_filter_mode == 1)
    notch_filter_frequency = 50;
elseif (notch_filter_mode == 2)
    notch_filter_frequency = 60;
end

desired_impedance_test_frequency = fread(fid, 1, 'single');
actual_impedance_test_frequency = fread(fid, 1, 'single');

% Place notes in data strucure
notes = struct( ...
    'note1', fread_QString(fid), ...
    'note2', fread_QString(fid), ...
    'note3', fread_QString(fid) );

% If data file is from GUI v1.1 or later, see if temperature sensor data
% was saved.
num_temp_sensor_channels = 0;
```

```
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 1) ...
    || (data_file_main_version_number > 1))
    num_temp_sensor_channels = fread(fid, 1, 'int16');
end

% If data file is from GUI v1.3 or later, load eval board mode.
eval_board_mode = 0;
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 3) ...
    || (data_file_main_version_number > 1))
    eval_board_mode = fread(fid, 1, 'int16');
end

% Place frequency-related information in data structure.
frequency_parameters = struct( ...
    'amplifier_sample_rate', sample_rate, ...
    'aux_input_sample_rate', sample_rate / 4, ...
    'supply_voltage_sample_rate', sample_rate / 60, ...
    'board_adc_sample_rate', sample_rate, ...
    'board_dig_in_sample_rate', sample_rate, ...
    'desired_dsp_cutoff_frequency', desired_dsp_cutoff_frequency, ...
    'actual_dsp_cutoff_frequency', actual_dsp_cutoff_frequency, ...
    'dsp_enabled', dsp_enabled, ...
    'desired_lower_bandwidth', desired_lower_bandwidth, ...
    'actual_lower_bandwidth', actual_lower_bandwidth, ...
    'desired_upper_bandwidth', desired_upper_bandwidth, ...
    'actual_upper_bandwidth', actual_upper_bandwidth, ...
    'notch_filter_frequency', notch_filter_frequency, ...
    'desired_impedance_test_frequency',
desired_impedance_test_frequency, ...
    'actual_impedance_test_frequency', actual_impedance_test_frequency );

% Define data structure for spike trigger settings.
spike_trigger_struct = struct( ...
    'voltage_trigger_mode', {}, ...
    'voltage_threshold', {}, ...
    'digital_trigger_channel', {}, ...
    'digital_edge_polarity', {} );

new_trigger_channel = struct(spike_trigger_struct);
```

```
spike_triggers = struct(spike_trigger_struct);

% Define data structure for data channels.
channel_struct = struct( ...
    'native_channel_name', {}, ...
    'custom_channel_name', {}, ...
    'native_order', {}, ...
    'custom_order', {}, ...
    'board_stream', {}, ...
    'chip_channel', {}, ...
    'port_name', {}, ...
    'port_prefix', {}, ...
    'port_number', {}, ...
    'electrode_impedance_magnitude', {}, ...
    'electrode_impedance_phase', {} );

new_channel = struct(channel_struct);

% Create structure arrays for each type of data channel.
amplifier_channels = struct(channel_struct);
aux_input_channels = struct(channel_struct);
supply_voltage_channels = struct(channel_struct);
board_adc_channels = struct(channel_struct);
board_dig_in_channels = struct(channel_struct);
board_dig_out_channels = struct(channel_struct);

amplifier_index = 1;
aux_input_index = 1;
supply_voltage_index = 1;
board_adc_index = 1;
board_dig_in_index = 1;
board_dig_out_index = 1;

% Read signal summary from data file header.

number_of_signal_groups = fread(fid, 1, 'int16');

for signal_group = 1:number_of_signal_groups
    signal_group_name = fread_QString(fid);
    signal_group_prefix = fread_QString(fid);
    signal_group_enabled = fread(fid, 1, 'int16');
```

```
    signal_group_num_channels = fread(fid, 1, 'int16');
    signal_group_num_amp_channels = fread(fid, 1, 'int16');

    if (signal_group_num_channels > 0 && signal_group_enabled > 0)
        new_channel(1).port_name = signal_group_name;
        new_channel(1).port_prefix = signal_group_prefix;
        new_channel(1).port_number = signal_group;
        for signal_channel = 1:signal_group_num_channels
            new_channel(1).native_channel_name = fread_QString(fid);
            new_channel(1).custom_channel_name = fread_QString(fid);
            new_channel(1).native_order = fread(fid, 1, 'int16');
            new_channel(1).custom_order = fread(fid, 1, 'int16');
            signal_type = fread(fid, 1, 'int16');
            channel_enabled = fread(fid, 1, 'int16');
            new_channel(1).chip_channel = fread(fid, 1, 'int16');
            new_channel(1).board_stream = fread(fid, 1, 'int16');
            new_trigger_channel(1).voltage_trigger_mode = fread(fid, 1,
'int16');
            new_trigger_channel(1).voltage_threshold = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_trigger_channel = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_edge_polarity = fread(fid, 1,
'int16');
            new_channel(1).electrode_impedance_magnitude = fread(fid, 1,
'single');
            new_channel(1).electrode_impedance_phase = fread(fid, 1,
'single');

            if (channel_enabled)
                switch (signal_type)
                    case 0
                        amplifier_channels(amplifier_index) = new_channel;
                        spike_triggers(amplifier_index) =
new_trigger_channel;
                        amplifier_index = amplifier_index + 1;
                    case 1
                        aux_input_channels(aux_input_index) = new_channel;
                        aux_input_index = aux_input_index + 1;
                    case 2
```

```
                    supply_voltage_channels(supply_voltage_index) =
new_channel;
                    supply_voltage_index = supply_voltage_index + 1;
                case 3
                    board_adc_channels(board_adc_index) = new_channel;
                    board_adc_index = board_adc_index + 1;
                case 4
                    board_dig_in_channels(board_dig_in_index) =
new_channel;
                    board_dig_in_index = board_dig_in_index + 1;
                case 5
                    board_dig_out_channels(board_dig_out_index) =
new_channel;
                    board_dig_out_index = board_dig_out_index + 1;
                otherwise
                    error('Unknown channel type');
            end
        end

    end
end

% Summarize contents of data file.
num_amplifier_channels = amplifier_index - 1;
num_aux_input_channels = aux_input_index - 1;
num_supply_voltage_channels = supply_voltage_index - 1;
num_board_adc_channels = board_adc_index - 1;
num_board_dig_in_channels = board_dig_in_index - 1;
num_board_dig_out_channels = board_dig_out_index - 1;

fprintf(1, 'Found %d amplifier channel%s.\n', ...
    num_amplifier_channels, plural(num_amplifier_channels));
fprintf(1, 'Found %d auxiliary input channel%s.\n', ...
    num_aux_input_channels, plural(num_aux_input_channels));
fprintf(1, 'Found %d supply voltage channel%s.\n', ...
    num_supply_voltage_channels, plural(num_supply_voltage_channels));
fprintf(1, 'Found %d board ADC channel%s.\n', ...
    num_board_adc_channels, plural(num_board_adc_channels));
fprintf(1, 'Found %d board digital input channel%s.\n', ...
    num_board_dig_in_channels, plural(num_board_dig_in_channels));
```

```
fprintf(1, 'Found %d board digital output channel%s.\n', ...
    num_board_dig_out_channels, plural(num_board_dig_out_channels));
fprintf(1, 'Found %d temperature sensors channel%s.\n', ...
    num_temp_sensor_channels, plural(num_temp_sensor_channels));
fprintf(1, '\n');

% Determine how many samples the data file contains.

% Each data block contains 60 amplifier samples.
bytes_per_block = 60 * 4;  % timestamp data
bytes_per_block = bytes_per_block + 60 * 2 * num_amplifier_channels;
% Auxiliary inputs are sampled 4x slower than amplifiers
bytes_per_block = bytes_per_block + 15 * 2 * num_aux_input_channels;
% Supply voltage is sampled 60x slower than amplifiers
bytes_per_block = bytes_per_block + 1 * 2 * num_supply_voltage_channels;
% Board analog inputs are sampled at same rate as amplifiers
bytes_per_block = bytes_per_block + 60 * 2 * num_board_adc_channels;
% Board digital inputs are sampled at same rate as amplifiers
if (num_board_dig_in_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Board digital outputs are sampled at same rate as amplifiers
if (num_board_dig_out_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Temp sensor is sampled 60x slower than amplifiers
if (num_temp_sensor_channels > 0)
   bytes_per_block = bytes_per_block + 1 * 2 * num_temp_sensor_channels;
end

% How many data blocks remain in this file?
data_present = 0;
bytes_remaining = filesize - ftell(fid);
if (bytes_remaining > 0)
    data_present = 1;
end

num_data_blocks = bytes_remaining / bytes_per_block;

num_amplifier_samples = 60 * num_data_blocks;
num_aux_input_samples = 15 * num_data_blocks;
```

```
num_supply_voltage_samples = 1 * num_data_blocks;
num_board_adc_samples = 60 * num_data_blocks;
num_board_dig_in_samples = 60 * num_data_blocks;
num_board_dig_out_samples = 60 * num_data_blocks;

record_time = num_amplifier_samples / sample_rate;

if (data_present)
    fprintf(1, 'File contains %0.3f seconds of data.  Amplifiers were sampled
at %0.2f kS/s.\n', ...
        record_time, sample_rate / 1000);
    fprintf(1, '\n');
else
    fprintf(1, 'Header file contains no data.  Amplifiers were sampled
at %0.2f kS/s.\n', ...
        sample_rate / 1000);
    fprintf(1, '\n');
end

if (data_present)

    % Pre-allocate memory for data.
    fprintf(1, 'Allocating memory for data...\n');

    t_amplifier = zeros(1, num_amplifier_samples);

    amplifier_data = zeros(num_amplifier_channels, num_amplifier_samples);
    aux_input_data = zeros(num_aux_input_channels, num_aux_input_samples);
    supply_voltage_data = zeros(num_supply_voltage_channels,
num_supply_voltage_samples);
    temp_sensor_data = zeros(num_temp_sensor_channels,
num_supply_voltage_samples);
    board_adc_data = zeros(num_board_adc_channels, num_board_adc_samples);
    board_dig_in_data = zeros(num_board_dig_in_channels,
num_board_dig_in_samples);
    board_dig_in_raw = zeros(1, num_board_dig_in_samples);
    board_dig_out_data = zeros(num_board_dig_out_channels,
num_board_dig_out_samples);
    board_dig_out_raw = zeros(1, num_board_dig_out_samples);

    % Read sampled data from file.
```

```
    fprintf(1, 'Reading data from file...\n');

    amplifier_index = 1;
    aux_input_index = 1;
    supply_voltage_index = 1;
    board_adc_index = 1;
    board_dig_in_index = 1;
    board_dig_out_index = 1;

    print_increment = 10;
    percent_done = print_increment;
    for i=1:num_data_blocks
        % In version 1.2, we moved from saving timestamps as unsigned
        % integeters to signed integers to accomidate negative (adjusted)
        % timestamps for pretrigger data.
        if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 2) ...
        || (data_file_main_version_number > 1))
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'int32');
        else
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'uint32');
        end
        if (num_amplifier_channels > 0)
            amplifier_data(:, amplifier_index:(amplifier_index+59)) =
fread(fid, [60, num_amplifier_channels], 'uint16')';
        end
        if (num_aux_input_channels > 0)
            aux_input_data(:, aux_input_index:(aux_input_index+14)) =
fread(fid, [15, num_aux_input_channels], 'uint16')';
        end
        if (num_supply_voltage_channels > 0)
            supply_voltage_data(:, supply_voltage_index) = fread(fid, [1,
num_supply_voltage_channels], 'uint16')';
        end
        if (num_temp_sensor_channels > 0)
            temp_sensor_data(:, supply_voltage_index) = fread(fid, [1,
num_temp_sensor_channels], 'int16')';
        end
        if (num_board_adc_channels > 0)
```

```
            board_adc_data(:, board_adc_index:(board_adc_index+59)) =
fread(fid, [60, num_board_adc_channels], 'uint16')';
        end
        if (num_board_dig_in_channels > 0)
            board_dig_in_raw(board_dig_in_index:(board_dig_in_index+59)) =
fread(fid, 60, 'uint16');
        end
        if (num_board_dig_out_channels > 0)

board_dig_out_raw(board_dig_out_index:(board_dig_out_index+59)) =
fread(fid, 60, 'uint16');
        end

        amplifier_index = amplifier_index + 60;
        aux_input_index = aux_input_index + 15;
        supply_voltage_index = supply_voltage_index + 1;
        board_adc_index = board_adc_index + 60;
        board_dig_in_index = board_dig_in_index + 60;
        board_dig_out_index = board_dig_out_index + 60;

        fraction_done = 100 * (i / num_data_blocks);
        if (fraction_done >= percent_done)
            fprintf(1, '%d%% done...\n', percent_done);
            percent_done = percent_done + print_increment;
        end
    end

    % Make sure we have read exactly the right amount of data.
    bytes_remaining = filesize - ftell(fid);
    if (bytes_remaining ~= 0)
        %error('Error: End of file not reached.');
    end

end

% Close data file.
fclose(fid);

if (data_present)

    fprintf(1, 'Parsing data...\n');
```

```
    % Extract digital input channels to separate variables.
    for i=1:num_board_dig_in_channels
      mask = 2^(board_dig_in_channels(i).native_order) *
ones(size(board_dig_in_raw));
      board_dig_in_data(i, :) = (bitand(board_dig_in_raw, mask) > 0);
    end
    for i=1:num_board_dig_out_channels
      mask = 2^(board_dig_out_channels(i).native_order) *
ones(size(board_dig_out_raw));
      board_dig_out_data(i, :) = (bitand(board_dig_out_raw, mask) > 0);
    end

    % Scale voltage levels appropriately.
    amplifier_data = 0.195 * (amplifier_data - 32768); % units = microvolts
    aux_input_data = 37.4e-6 * aux_input_data; % units = volts
    supply_voltage_data = 74.8e-6 * supply_voltage_data; % units = volts
    if (eval_board_mode == 1)
        board_adc_data = 152.59e-6 * (board_adc_data - 32768); % units = volts
    else
        board_adc_data = 50.354e-6 * board_adc_data; % units = volts
    end
    temp_sensor_data = temp_sensor_data / 100; % units = deg C

    % Check for gaps in timestamps.
    num_gaps = sum(diff(t_amplifier) ~= 1);
    if (num_gaps == 0)
        fprintf(1, 'No missing timestamps in data.\n');
    else
        fprintf(1, 'Warning: %d gaps in timestamp data found.  Time scale will
not be uniform!\n', ...
            num_gaps);
    end

    % Scale time steps (units = seconds).
    t_amplifier = t_amplifier / sample_rate;
    t_aux_input = t_amplifier(1:4:end);
    t_supply_voltage = t_amplifier(1:60:end);
    t_board_adc = t_amplifier;
    t_dig = t_amplifier;
    t_temp_sensor = t_supply_voltage;
```

```
    % If the software notch filter was selected during the recording, apply
the
    % same notch filter to amplifier data here.
    if (notch_filter_frequency > 0)
        fprintf(1, 'Applying notch filter...\n');

        print_increment = 10;
        percent_done = print_increment;
        for i=1:num_amplifier_channels
            amplifier_data(i,:) = ...
                notch_filter(amplifier_data(i,:), sample_rate,
notch_filter_frequency, 10);

            fraction_done = 100 * (i / num_amplifier_channels);
            if (fraction_done >= percent_done)
                fprintf(1, '%d%% done...\n', percent_done);
                percent_done = percent_done + print_increment;
            end

        end
    end

end

% Move variables to base workspace.

move_to_base_workspace(notes);
move_to_base_workspace(frequency_parameters);

if (num_amplifier_channels > 0)
    move_to_base_workspace(amplifier_channels);
    if (data_present)
        move_to_base_workspace(amplifier_data);
        move_to_base_workspace(t_amplifier);
    end
    move_to_base_workspace(spike_triggers);
end
if (num_aux_input_channels > 0)
    move_to_base_workspace(aux_input_channels);
    if (data_present)
```

```
        move_to_base_workspace(aux_input_data);
        move_to_base_workspace(t_aux_input);
    end
end
if (num_supply_voltage_channels > 0)
    move_to_base_workspace(supply_voltage_channels);
    if (data_present)
        move_to_base_workspace(supply_voltage_data);
        move_to_base_workspace(t_supply_voltage);
    end
end
if (num_board_adc_channels > 0)
    move_to_base_workspace(board_adc_channels);
    if (data_present)
        move_to_base_workspace(board_adc_data);
        move_to_base_workspace(t_board_adc);
    end
end
if (num_board_dig_in_channels > 0)
    move_to_base_workspace(board_dig_in_channels);
    if (data_present)
        move_to_base_workspace(board_dig_in_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_board_dig_out_channels > 0)
    move_to_base_workspace(board_dig_out_channels);
    if (data_present)
        move_to_base_workspace(board_dig_out_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_temp_sensor_channels > 0)
    if (data_present)
        move_to_base_workspace(temp_sensor_data);
        move_to_base_workspace(t_temp_sensor);
    end
end

fprintf(1, 'Done!  Elapsed time: %0.1f seconds\n', toc);
if (data_present)
```

```
    fprintf(1, 'Extracted data are now available in the MATLAB
workspace.\n');
else
    fprintf(1, 'Extracted waveform information is now available in the MATLAB
workspace.\n');
end
fprintf(1, 'Type ''whos'' to see variables.\n');
fprintf(1, '\n');

return


function a = fread_QString(fid)

% a = read_QString(fid)
%
% Read Qt style QString.  The first 32-bit unsigned number indicates
% the length of the string (in bytes).  If this number equals 0xFFFFFFFF,
% the string is null.

a = '';
length = fread(fid, 1, 'uint32');
if length == hex2num('ffffffff')
    return;
end
% convert length from bytes to 16-bit Unicode words
length = length / 2;

for i=1:length
    a(i) = fread(fid, 1, 'uint16');
end

return


function s = plural(n)

% s = plural(n)
%
% Utility function to optionally plurailze words based on the value
% of n.
```

```
if (n == 1)
   s = '';
else
   s = 's';
end

return


function out = notch_filter(in, fSample, fNotch, Bandwidth)

% out = notch_filter(in, fSample, fNotch, Bandwidth)
%
% Implements a notch filter (e.g., for 50 or 60 Hz) on vector 'in'.
% fSample = sample rate of data (in Hz or Samples/sec)
% fNotch = filter notch frequency (in Hz)
% Bandwidth = notch 3-dB bandwidth (in Hz).  A bandwidth of 10 Hz is
%   recommended for 50 or 60 Hz notch filters; narrower bandwidths lead to
%   poor time-domain properties with an extended ringing response to
%   transient disturbances.
%
% Example:  If neural data was sampled at 30 kSamples/sec
% and you wish to implement a 60 Hz notch filter:
%
% out = notch_filter(in, 30000, 60, 10);

tstep = 1/fSample;
Fc = fNotch*tstep;

L = length(in);

% Calculate IIR filter parameters
d = exp(-2*pi*(Bandwidth/2)*tstep);
b = (1 + d*d)*cos(2*pi*Fc);
a0 = 1;
a1 = -b;
a2 = d*d;
a = (1 + d*d)/2;
b0 = 1;
b1 = -2*cos(2*pi*Fc);
```

```
b2 = 1;

out = zeros(size(in));
out(1) = in(1);
out(2) = in(2);
% (If filtering a continuous data stream, change out(1) and out(2) to the
%  previous final two values of out.)

% Run filter
for i=3:L
    out(i) = (a*b2*in(i-2) + a*b1*in(i-1) + a*b0*in(i) - a2*out(i-2) -
a1*out(i-1))/a0;
end

return


function move_to_base_workspace(variable)

% move_to_base_workspace(variable)
%
% Move variable from function workspace to base MATLAB workspace so
% user will have access to it after the program ends.

variable_name = inputname(1);
assignin('base', variable_name, variable);

return;
```

5. success_160426_IntanEphysAnalysis.m

```
% Modified by Baihan Lin
% to extract information from Intan Tech recording data
% Apr 2016

clear all %Starting a new analysis so we want to eliminate all old variables
close all
```

```
%% Begin with opening file to be analyzed
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
% read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.

% Modified by Baihan Lin
% Apr 2016

% read_Intan_RHD2000_file_combine
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%
% Example:
% >> clear
% >> read_Intan_RHD200_file
% >> whos
% >> amplifier_channels(1)
% >> plot(t_amplifier, amplifier_data(1,:))

% Here I change it from:
%[file, path, filterindex] = uigetfile('*.rhd', 'Select an RHD2000 Data File',
'MultiSelect', 'off');

% Read most recent file automatically.
```

```
%path = 'C:\Users\Reid\Documents\RHD2132\testing\';
%d = dir([path '*.rhd']);
%file = d(end).name;

% I decided to use automatic method for data collection:

prompt = 'What is your folder?: ';
disp('e.g.  /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data');
path = input(prompt,'s');
% /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test

% So that I can specify a folder to access all data files.

%[status, list] = system('cd path');

[~,list] = system(['find ' path ' -type f -name "*.rhd"']);

system(['mkdir output-' date]);
system(['cd output-' date]);

diary(strcat(path, 'report_', date, '.out'));
diary on;
disp(path);
disp(date);

files = strsplit(list);
length(files);

for countfile = 1:length(files)
    trials{countfile} = files{countfile}(1:end-11);
end

trials = unique(trials);
trials = trials(~cellfun('isempty',trials));

index = strfind(files, trials{1});

for trial = 1 : length(trials)
    filename = trials{trial};
    index = strfind(files, filename);
    indexmat = cell2mat(index);
```

```
[~,first,~] = unique(indexmat, 'first');
arrange_Intan_RHD(files{first(1)});

amp_data = amplifier_data;
try
    ai_data = aux_input_data;
    bdi_data = board_dig_in_data;
    sv_data = supply_voltage_data;
    t_ai = t_aux_input;
    t_d = t_dig;
    t_sv = t_supply_voltage;
catch exception
end
t_amp = t_amplifier;

if length(index) > 1
    for ind = 2 : length(index)
        if index{ind} == 1
            arrange_Intan_RHD(files{ind});
            try

                amp_data = [amp_data, amplifier_data];
                ai_data = [ai_data, aux_input_data];
                bdi_data = [bdi_data, board_dig_in_data];
                sv_data = [sv_data, supply_voltage_data];
                t_amp = [t_amp, t_amplifier];
                t_ai =  [t_ai,t_aux_input];
                t_d = [t_d, t_dig];
                t_sv = [t_sv, t_supply_voltage];


            catch exception

            end
        end
    end
end

amplifier_data = amp_data;
t_amplifier = t_amp;
try
```

```
        aux_input_data = ai_data;
        board_dig_in_data = bdi_data;
        supply_voltage_data = sv_data;
        t_aux_input = t_ai;
        t_dig = t_d;
        t_supply_voltage = t_sv;
    catch
    end

    %check what variables have been imported, especially if youre unsure
    %whether accessory amplifier channels were disabled or not during
recording

    %% Establishing some basic variables from values pulled in by above
function

    amp_chan = 1;
    amp_imp = amplifier_channels(1).electrode_impedance_magnitude;

    for chan = 1 : length(amplifier_channels)
        if amplifier_channels(chan).electrode_impedance_magnitude < amp_imp
            amp_chan = chan;
            amp_imp =
amplifier_channels(chan).electrode_impedance_magnitude;
        end
    end

    disp('amplifer information');
    amplifier_channels(amp_chan)

    %Channel data is being collected on (on
    %preamplifier this would be 'A-004'
    %Above command gives data output about the channel on which data is being
    %collected

    %% Variable name changes below to simplify:

    try
        tRat = t_amplifier; %time variable for ephys data
        tLED = t_dig; %time variable for LED data
        try
```

```
            ui.ratData = amplifier_data(amp_chan,:);
        catch exception
            ui.ratData = amplifier_data(1,:);
        end
        lLED = board_dig_in_data(1,:);
        rLED = board_dig_in_data(2,:);
    catch
    end

    % %% Check variables look right-- plot should be identical to last one
    % figure % for spike detection
    %     hold on
    %     plot(tRat, ui.ratData,'blue')
    %     plot(tLED,lLED,'red')  %max makes red lines continue across top
half of vertical axis
    %     plot(tLED,rLED,'green')
    %     xlabel 'time (s)'
    %     ylabel 'amplitude (A.U.)'
    %     legend('Raw Data', 'Left Eye LED','Right Eye LED')

    %% filter data
    Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    try
        ui.ratData = filtfilt(b,a,ui.ratData);
        %% invert signal data for thresholding
        ui.ratData = ui.ratData.*(-1);
    catch
    end

    %% Setting threshold for spikes and finding light ON times
    threshold = 25;
    Fs =  20000; %amplifier_sample_rate
    windowSize = Fs * 0.05; %creates our time interval by taking the 20k
    % sampling rate at which the data was collected and converts it to
    % timestamps collected every millisecond, in other words the value of
    % windowSize is 1 ms.


    % Finding all spikes in recording:
```

```
    try
        ui.spikes = diff(ui.ratData > threshold) > 0.1;
    catch
    end


    %% Plot Again with spikes showing & correct time axis now:

    try
        time = length(tRat)-1;
        fig1 = figure; % for spike detection
        %     figure('units','normalized','position',[0 0 1 1]);
        %     figure('Visible','off');
        hold on
        plot(tRat(1:time), ui.ratData(1:time),'blue');
        plot(tRat(1:time), ui.spikes*max(ui.ratData),'black');
        plot(tLED(1:time),lLED(1:time)*80,'green');  %max makes red lines
continue across top half of vertical axis
        plot(tLED(1:time),rLED(1:time)*80,'red');
        xlabel 'time (s)';
        ylabel 'amplitude (A.U.)';
        legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED');
        %     set(fig1, 'Position', [100, 100, 1920, 1080]);
        saveas(fig1, strcat(filename, '-spikes.png'),'png');
        %     print(fig1, strcat(filename, '-spikes'),'-dpng');
        close(fig1);
    catch
    end

    %% Count spikes during each LED stimulation

    try
        SpikesL = sum(ui.spikes.*lLED(1:end-1));
        SpikesR = sum(ui.spikes.*rLED(1:end-1));

        disp(strcat('spikeL = ', SpikesL));
        disp(strcat('spikeR = ', SpikesR));
    catch
    end
```

```
    %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

    lightstim = 199; %change?
    try
        ui.leftLEDon = diff(lLED < -lightstim)>0.1;
        ui.rightLEDon = diff(rLED < -lightstim)>0.1;

        %        times.leftLED = find(leftLED == 500);
        %        times.rightLED = find(rightLED == 500);
        %rewrite:
        times.lLEDon = find(lLED == 1);
        % Gives all time points that left LED is on
        %result is a vector 1 x 80299
        times.lLEDoff = find(lLED == 0);
        % Gives all time points that left LED is off
        %result is a vector 1 x 1119941
        times.rLEDon = find(rLED == 1);
        times.rLEDoff = find(rLED == 0);
    catch
    end
    % the above code will break out the time points when the LED is on for
each
    % side and when it is off. Next step:
    %         Need to ask it to count how many times ui.spikes takes place
    %         during each LEDon segment

    try
        if length(times.lLEDon) == 0 || length(times.rLEDon) == 0
            disp('WARNING! Failed to detect LED!')
            disp(filename);
            disp('----------------------');
        else



            %% gets light "on" times into one array
            times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
            %results in three specific time points for this LED
            times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
```

```
            % this turns the times.xLEDon into a list of the points when the
LED turned
            % on ***Use this for making a raster plot****
            %results in two specific time points for this LED


            %% create raster plots-Left Stim
            % Error: Subscript indices must either be real ve integers or
            % logicals.
            for l = 1:length(times.lLEDstart)
                % collects window of data each time the light stimulus initiated
                windowSize = round(Fs*0.5); % window size in samples
                ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
            end

            %In original script, variable that's equivalent to 'times.lLEDon'
is e.g.
            % a 23x1 double that includes only the start times for light turning
            % on. Maybe be better to use times.lLEDstart?
            %times.lLedon in this script gives the chunks when led was on,
aka a
            %1x80299 double

            t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
            t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
            % creates time vector for raster

            times.Lrasterlight = 1:(length(times.lLEDstart));
            t.Lrasterlight =
ones(1,(length(times.lLEDstart)))*windowSize/Fs;
            % creates dashed line for indicating stim onset on raster plot

            ui.Lraster = horzcat(ui.Lrastercell{:});
            % concatenates cell array into a double

            Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
            % creates transform for stacking windowed data for the raster plot


            %% create raster plots-Right Stim
```

```
        % Error: Subscript indices must either be real ve integers or
        % logicals.
        for r = 1:length(times.rLEDstart)
            % collects window of data each time the light stimulus initiated
            windowSize = round(Fs*0.5); % window size in samples
            ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
        end


        t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
        t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
        % creates time vector for raster

        times.Rrasterlight = 1:(length(times.rLEDstart));
        t.Rrasterlight =
ones(1,(length(times.rLEDstart)))*windowSize/Fs;
        % creates dashed line for indicating stim onset on raster plot

        ui.Rraster = horzcat(ui.Rrastercell{:});
        % concatenates cell array into a double

        Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
        % creates transform for stacking windowed data for the raster plot


        %% Before running next part, need to find how many times light
goes on,
        % do this by checking the variable "times.lLEDstart" and
"times.rLEDstart".
        % It will either show the exact values for the start times or will
indicate
        % how many different light on times there are, if trials exceeds
~5.

        disp('LED light information: ');
        times


        %% this number needs to be input as last value in reshape function
below:
```

```
        ui.LrasterStack =
reshape(ui.Lraster,20001,length(times.Lrasterlight));
        ui.RrasterStack =
reshape(ui.Rraster,20001,length(times.Rrasterlight));
        %% Check plot to verify reshape has been applied appropriately
to LEFT data:

        fig2 = figure % creates raster plot
        %    figure('units','normalized','position',[0 0 1 1]);
        %    figure('Visible','off');
        plot(t.Lraster,ui.LrasterStack+Lstack-1);
        hold on
        line([0.5 0.5], [0 length(times.Lrasterlight)], 'Color', 'k',
'LineWidth',2)
        %    plot(t.Lrasterlight,times.Lrasterlight,'-black',
'LineWidth',8);
        hold off
        ylabel 'trial number';
        xlabel 'time (s)';
        xlim([0 1]);
        saveas(fig2, strcat(filename, '-Lraster.png'),'png');
        %    print(fig2, strcat(filename, '-Lraster'),'-dpng');
        close(fig2);

        %% Check plot to verify reshape has been applied appropriately
to RIGHT data:

        fig3 = figure % creates raster plot
        %    figure('units','normalized','position',[0 0 1 1]);
        %    figure('Visible','off');
        plot(t.Rraster,ui.RrasterStack+Rstack-1);
        hold on
        line([0.5 0.5], [0 length(times.Rrasterlight)], 'Color', 'k',
'LineWidth',2)
        hold off
        ylabel 'trial number';
        xlabel 'time (s)';
        xlim([0 1]);
        saveas(fig3, strcat(filename, '-Rraster.png'),'png');
```

```
            %    print(fig3, strcat(filename, '-Rraster'),'-dpng');
            close(fig3);

            %% Lastly, get spike averages for each eye
            stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
            %    % calculates average number of spikes after light turned on
            stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
            %    % calculates average number of spikes preceding light onset

            disp('For left eye (L):');
            disp(strcat('spike average (on):', stats.spikes.Laveon));
            disp(strcat('spike average (off):', stats.spikes.Laveoff));


            %% Lastly, get spike avwerages for each eye
            stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
            %    % calculates average number of spikes after light turned on
            stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
            %    % calculates average number of spikes preceding light onset

            disp('For right eye (R):');
            disp(strcat('spike average (on):', stats.spikes.Raveon));
            disp(strcat('spike average (off):', stats.spikes.Raveoff));

            disp(strcat('Finished!!!', filename));
            disp('--------------------------------');

        end
    catch
    end
    %%

    %    clear all %Starting a new analysis so we want to eliminate all old
variables
    close all
    clearvars -except path list files trials trial index;
```

```
end

diary off;
```

6. all_160421_IntanEphysAnalysis.m

```
clear all %Starting a new analysis so we want to eliminate all old variables
close all

 %% Begin with opening file to be analyzed
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
% read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.

read_Intan_RHD_combine
% /Users/DoerLBH/Dropbox/git/OLab_IntanEphys/Data/test

%check what variables have been imported, especially if youre unsure
%whether accessory amplifier channels were disabled or not during recording
%% Establishing some basic variables from values pulled in by above function

amp_chan = 1;
amp_imp = amplifier_channels(1).electrode_impedance_magnitude

for chan = 1 : length(amplifier_channels)
   if amplifier_channels(chan).electrode_impedance_magnitude < amp_imp
       amp_chan = chan;
```

```
   end
end

amplifier_channels(amp_chan) %Channel data is being collected on (on
%preamplifier this would be 'A-004'
%Above command gives data output about the channel on which data is being
%collected

%% Variable name changes below to simplify:

tRat = t_amplifier; %time variable for ephys data
tLED = t_dig; %time variable for LED data
ui.ratData = amplifier_data(amp_chan,:);
lLED = board_dig_in_data(1,:);
rLED = board_dig_in_data(2,:);

% %% Check variables look right-- plot should be identical to last one
% figure % for spike detection
%     hold on
%     plot(tRat, ui.ratData,'blue')
%     plot(tLED,lLED,'red')  %max makes red lines continue across top half
of vertical axis
%     plot(tLED,rLED,'green')
%     xlabel 'time (s)'
%     ylabel 'amplitude (A.U.)'
%     legend('Raw Data', 'Left Eye LED','Right Eye LED')

%% filter data
     Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    ui.ratData = filtfilt(b,a,ui.ratData);
    %% invert signal data for thresholding
    ui.ratData = ui.ratData.*(-1);



    %% Setting threshold for spikes and finding light ON times
threshold = 25;
Fs =  20000; %amplifier_sample_rate
windowSize = Fs * 0.05; %creates our time interval by taking the 20k
% sampling rate at which the data was collected and converts it to
```

```
% timestamps collected every millisecond, in other words the value of
% windowSize is 1 ms.


% Finding all spikes in recording:
    ui.spikes = diff(ui.ratData > threshold) > 0.1;



%% Plot Again with spikes showing & correct time axis now:

time = length(tRat)-1;
figure % for spike detection
    hold on
    plot(tRat(1:time), ui.ratData(1:time),'blue')
    plot(tRat(1:time), ui.spikes*max(ui.ratData),'black')
    plot(tLED(1:time),lLED(1:time)*80,'green')  %max makes red lines
continue across top half of vertical axis
    plot(tLED(1:time),rLED(1:time)*80,'red')
    xlabel 'time (s)'
    ylabel 'amplitude (A.U.)'
    legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED')



%% Count spikes during each LED stimulation

 SpikesL = sum(ui.spikes.*lLED(1:end-1))
 SpikesR = sum(ui.spikes.*rLED(1:end-1))


    %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

     lightstim = 199; %change?
        ui.leftLEDon = diff(lLED < -lightstim)>0.1;
        ui.rightLEDon = diff(rLED < -lightstim)>0.1;
%        times.leftLED = find(leftLED == 500);
%        times.rightLED = find(rightLED == 500);
%rewrite:
times.lLEDon = find(lLED == 1);
```

```
% Gives all time points that left LED is on
    %result is a vector 1 x 80299
times.lLEDoff = find(lLED == 0);
% Gives all time points that left LED is off
    %result is a vector 1 x 1119941
times.rLEDon = find(rLED == 1);
times.rLEDoff = find(rLED == 0);
% the above code will break out the time points when the LED is on for each
% side and when it is off. Next step:
%          Need to ask it to count how many times ui.spikes takes place
%          during each LEDon segment

%% gets light "on" times into one array
times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
    %results in three specific time points for this LED
times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
% this turns the times.xLEDon into a list of the points when the LED turned
% on ***Use this for making a raster plot****
    %results in two specific time points for this LED


    %% create raster plots-Left Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for l = 1:length(times.lLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
    end

    %In original script, variable that's equivalent to 'times.lLEDon' is e.g.
    % a 23x1 double that includes only the start times for light turning
    % on. Maybe be better to use times.lLEDstart?
    %times.lLedon in this script gives the chunks when led was on, aka a
    %1x80299 double

    t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
    t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
    % creates time vector for raster
```

```
    times.Lrasterlight = 1:(length(times.lLEDstart));
    t.Lrasterlight = ones(1,(length(times.lLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Lraster = horzcat(ui.Lrastercell{:});
    % concatenates cell array into a double

    Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
    % creates transform for stacking windowed data for the raster plot

    %% create raster plots-Right Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for r = 1:length(times.rLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
    end


    t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
    t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
    % creates time vector for raster

    times.Rrasterlight = 1:(length(times.rLEDstart));
    t.Rrasterlight = ones(1,(length(times.rLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Rraster = horzcat(ui.Rrastercell{:});
    % concatenates cell array into a double

    Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
    % creates transform for stacking windowed data for the raster plot


%% Before running next part, need to find how many times light goes on,
% do this by checking the variable "times.lLEDstart" and "times.rLEDstart".
% It will either show the exact values for the start times or will indicate
% how many different light on times there are, if trials exceeds ~5.
times
```

```
%% this number needs to be input as last value in reshape function below:


    ui.LrasterStack =
reshape(ui.Lraster,20001,length(times.Lrasterlight));
    ui.RrasterStack =
reshape(ui.Rraster,20001,length(times.Rrasterlight));
%% Check plot to verify reshape has been applied appropriately to LEFT data:

   figure % creates raster plot
   plot(t.Lraster,ui.LrasterStack+Lstack-1);
   hold on
   line([0.5 0.5], [0 length(times.Lrasterlight)], 'Color', 'k',
'LineWidth',2)
   %    plot(t.Lrasterlight,times.Lrasterlight,'-black', 'LineWidth',8);
   hold off
   ylabel 'trial number';
   xlabel 'time (s)';
   xlim([0 1]);


%% Check plot to verify reshape has been applied appropriately to RIGHT data:

   figure % creates raster plot
   plot(t.Rraster,ui.RrasterStack+Rstack-1);
   hold on
   line([0.5 0.5], [0 length(times.Rrasterlight)], 'Color', 'k',
'LineWidth',2)
   hold off
   ylabel 'trial number';
   xlabel 'time (s)';
   xlim([0 1]);

%% Lastly, get spike averages for each eye
    stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
%    % calculates average number of spikes after light turned on
    stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
```

```
%     % calculates average number of spikes preceding light onset


stats.spikes.Laveon
stats.spikes.Laveoff

%% Lastly, get spike avwerages for each eye
    stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
%     % calculates average number of spikes after light turned on
    stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Raveon
stats.spikes.Raveoff
```

7. read_Intan_RHD_combine.m

```
function read_Intan_RHD_combine

% Modified by Baihan Lin
% Apr 2016

% read_Intan_RHD2000_file_combine
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
%
% Example:
% >> clear
% >> read_Intan_RHD200_file
```

```
% >> whos
% >> amplifier_channels(1)
% >> plot(t_amplifier, amplifier_data(1,:))

% Here I change it from:
%[file, path, filterindex] = uigetfile('*.rhd', 'Select an RHD2000 Data File',
'MultiSelect', 'off');

% Read most recent file automatically.
%path = 'C:\Users\Reid\Documents\RHD2132\testing\';
%d = dir([path '*.rhd']);
%file = d(end).name;

% I decided to use automatic method for data collection:

prompt = 'What is your folder?: ';
path = input(prompt,'s');

% So that I can specify a folder to access all data files.

%[status, list] = system('cd path');
[~,list] = system(['find ' path ' -type f -name "*.rhd"']);

files = strsplit(list);
length(files);

for countfile = 1:length(files)
    trials{countfile} = files{countfile}(1:end-11);
end

trials = unique(trials);
trials = trials(~cellfun('isempty',trials));

index = strfind(files, trials{1});

for trial = 1 : length(trials)
    index = strfind(files, trials{trial});
    indexmat = cell2mat(index);
    [~,first,~] = unique(indexmat, 'first');
    arrange_Intan_RHD(files{first(trial)});
```

```
    amp_data = amplifier_data;
    ai_data = aux_input_data;
    bdi_data = board_dig_in_data;
    sv_data = supply_voltage_data;
    t_amp = t_amplifier;
    t_ai = t_aux_input;
    t_d = t_dig;
    t_sv = t_supply_voltage;

    if length(index) > 1
        for ind = 2 : length(index)
            if index{ind} == 1
                arrange_Intan_RHD(files{ind});
                amp_data = [amp_data, amplifier_data];
                ai_data = [ai_data, aux_input_data];
                bdi_data = [bdi_data, board_dig_in_data];
                sv_data = [sv_data, supply_voltage_data];
                t_amp = [t_amp, t_amplifier];
                t_ai =  [t_ai,t_aux_input];
                t_d = [t_d, t_dig];
                t_sv = [t_sv, t_supply_voltage];
            end
        end
    end

    amplifier_data = amp_data;
    aux_input_data = ai_data;
    board_dig_in_data = bdi_data;
    supply_voltage_data = sv_data;
    t_amplifier = t_amp;
    t_aux_input = t_ai;
    t_dig = t_d;
    t_supply_voltage = t_sv;

    disp('----------------')
end

end
```

8. backup_160407_IntanEphysAnalysis.m

```
% Debugged and Modifidied by Baihan Lin, Olavarria Lab
% April 2016

%% Begin with opening file to be analyzed
clear all %Starting a new analysis so we want to eliminate all old variables
close all
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.

%check what variables have been imported, especially if youre unsure
%whether accessory amplifier channels were disabled or not during recording
%% Establishing some basic variables from values pulled in by above function

amplifier_channels(1) %Channel data is being collected on (on
%preamplifier this would be 'A-004'
%Above command gives data output about the channel on which data is being
%collected

%% Variable name changes below to simplify:

tRat = t_amplifier; %time variable for ephys data
tLED = t_dig; %time variable for LED data
ui.ratData = amplifier_data(1,:);
lLED = board_dig_in_data(1,:);
rLED = board_dig_in_data(2,:);

% %% Check variables look right-- plot should be identical to last one
```

```
% figure % for spike detection
%    hold on
%    plot(tRat, ui.ratData,'blue')
%    plot(tLED,lLED,'red')  %max makes red lines continue across top half
of vertical axis
%    plot(tLED,rLED,'green')
%    xlabel 'time (s)'
%    ylabel 'amplitude (A.U.)'
%    legend('Raw Data', 'Left Eye LED','Right Eye LED')

%% filter data
     Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    ui.ratData = filtfilt(b,a,ui.ratData);
    %% invert signal data for thresholding
    ui.ratData = ui.ratData.*(-1);


    %% Setting threshold for spikes and finding light ON times
threshold = 20;
Fs =  20000; %amplifier_sample_rate
windowSize = Fs * 0.05; %creates our time interval by taking the 20k
% sampling rate at which the data was collected and converts it to
% timestamps collected every millisecond, in other words the value of
% windowSize is 1 ms.


% Finding all spikes in recording:
    ui.spikes = diff(ui.ratData > threshold) > 0.1;



%% Plot Again with spikes showing & correct time axis now:

time =length(tRat)-1
figure % for spike detection
    hold on
    plot(tRat(1:time), ui.ratData(1:time),'blue')
    plot(tRat(1:time), ui.spikes*max(ui.ratData),'black')
```

```
    plot(tLED(1:time),lLED(1:time)*80,'green')  %max makes red lines
continue across top half of vertical axis
    plot(tLED(1:time),rLED(1:time)*80,'red')
    xlabel 'time (s)'
    ylabel 'amplitude (A.U.)'
    legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED')



%% Count spikes during each LED stimulation

 SpikesL = sum(ui.spikes.*lLED(1:end-1))
 SpikesR = sum(ui.spikes.*rLED(1:end-1))


    %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

     lightstim = 199; %change?
        ui.leftLEDon = diff(lLED < -lightstim)>0.1;
        ui.rightLEDon = diff(rLED < -lightstim)>0.1;
%        times.leftLED = find(leftLED == 500);
%        times.rightLED = find(rightLED == 500);
%rewrite:
times.lLEDon = find(lLED == 1);
% Gives all time points that left LED is on
    %result is a vector 1 x 80299
times.lLEDoff = find(lLED == 0);
% Gives all time points that left LED is off
    %result is a vector 1 x 1119941
times.rLEDon = find(rLED == 1);
times.rLEDoff = find(rLED == 0);
% the above code will break out the time points when the LED is on for each
% side and when it is off. Next step:
%         Need to ask it to count how many times ui.spikes takes place
%         during each LEDon segment

%% gets light "on" times into one array
times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
    %results in three specific time points for this LED
times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
```

```
% this turns the times.xLEDon into a list of the points when the LED turned
% on ***Use this for making a raster plot****
    %results in two specific time points for this LED


    %% create raster plots-Left Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for l = 1:length(times.lLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
    end

    %In original script, variable that's equivalent to 'times.lLEDon' is e.g.
    % a 23x1 double that includes only the start times for light turning
    % on. Maybe be better to use times.lLEDstart?
    %times.lLedon in this script gives the chunks when led was on, aka a
    %1x80299 double

    t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
    t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
    % creates time vector for raster

    times.Lrasterlight = 1:(length(times.lLEDstart));
    t.Lrasterlight = ones(1,(length(times.lLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Lraster = horzcat(ui.Lrastercell{:});
    % concatenates cell array into a double

    Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
    % creates transform for stacking windowed data for the raster plot

    %% create raster plots-Right Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for r = 1:length(times.rLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
```

```
       ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
    end


    t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
    t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
    % creates time vector for raster

    times.Rrasterlight = 1:(length(times.rLEDstart));
    t.Rrasterlight = ones(1,(length(times.rLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Rraster = horzcat(ui.Rrastercell{:});
    % concatenates cell array into a double

    Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
    % creates transform for stacking windowed data for the raster plot


%% Before running next part, need to find how many times light goes on,
% do this by checking the variable "times.lLEDstart" and "times.rLEDstart".
% It will either show the exact values for the start times or will indicate
% how many different light on times there are, if trials exceeds ~5.
times


%% this number needs to be input as last value in reshape function below:


     ui.LrasterStack = reshape(ui.Lraster,20001,5);
     ui.RrasterStack = reshape(ui.Rraster,20001,9);
%% Check plot to verify reshape has been applied appropriately to LEFT data:

    figure % creates raster plot
    plot(t.Lraster,ui.LrasterStack+Lstack);
    hold on
    plot(t.Lrasterlight,times.Lrasterlight,'-.black');
    hold off
    ylabel 'trial number'
    xlabel 'time (s)'
```

```
%% Check plot to verify reshape has been applied appropriately to RIGHT data:

    figure % creates raster plot
    plot(t.Rraster,ui.RrasterStack+Rstack);
    hold on
    plot(t.Rrasterlight,times.Rrasterlight,'-.black');
    hold off
    ylabel 'trial number'
    xlabel 'time (s)'
%% Lastly, get spike averages for each eye
     stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Laveon
stats.spikes.Laveoff

%% Lastly, get spike avwerages for each eye
     stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Raveon
stats.spikes.Raveoff
```

9. backup_IntanEphysAnalysis_033016_Original.m

```
%% Begin with opening file to be analyzed
clear all %Starting a new analysis so we want to eliminate all old variables
```

```
close all
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.

%check what variables have been imported, especially if youre unsure
%whether accessory amplifier channels were disabled or not during recording
%% Establishing some basic variables from values pulled in by above function

amplifier_channels(1) %Channel data is being collected on (on
%preamplifier this would be 'A-004'
%Above command gives data output about the channel on which data is being
%collected

%% Variable name changes below to simplify:

tRat = t_amplifier; %time variable for ephys data
tLED = t_dig; %time variable for LED data
ui.ratData = amplifier_data(1,:);
lLED = board_dig_in_data(1,:);
rLED = board_dig_in_data(2,:);

% %% Check variables look right-- plot should be identical to last one
% figure % for spike detection
%     hold on
%     plot(tRat, ui.ratData,'blue')
%     plot(tLED,lLED,'red')  %max makes red lines continue across top half
of vertical axis
%     plot(tLED,rLED,'green')
%     xlabel 'time (s)'
```

```
%    ylabel 'amplitude (A.U.)'
%    legend('Raw Data', 'Left Eye LED','Right Eye LED')

%% filter data
     Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    ui.ratData = filtfilt(b,a,ui.ratData);
    %% invert signal data for thresholding
    ui.ratData = ui.ratData.*(-1);


    %% Setting threshold for spikes and finding light ON times
threshold = 20;
Fs =  20000; %amplifier_sample_rate
windowSize = Fs * 0.05; %creates our time interval by taking the 20k
% sampling rate at which the data was collected and converts it to
% timestamps collected every millisecond, in other words the value of
% windowSize is 1 ms.


% Finding all spikes in recording:
    ui.spikes = diff(ui.ratData > threshold) > 0.1;



%% Plot Again with spikes showing & correct time axis now:

t=length(tRat)-1
figure % for spike detection
    hold on
    plot(tRat(1:t), ui.ratData(1:t),'blue')
    plot(tRat(1:t), ui.spikes*max(ui.ratData),'black')
    plot(tLED(1:t),lLED(1:t)*80,'green')  %max makes red lines continue
across top half of vertical axis
    plot(tLED(1:t),rLED(1:t)*80,'red')
    xlabel 'time (s)'
    ylabel 'amplitude (A.U.)'
    legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED')
```

```
%% Count spikes during each LED stimulation

 SpikesL = sum(ui.spikes.*lLED(1:end-1))
 SpikesR = sum(ui.spikes.*rLED(1:end-1))


   %%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful

    lightstim = 199; %change?
       ui.leftLEDon = diff(lLED < -lightstim)>0.1;
       ui.rightLEDon = diff(rLED < -lightstim)>0.1;
%        times.leftLED = find(leftLED == 500);
%        times.rightLED = find(rightLED == 500);
%rewrite:
times.lLEDon = find(lLED == 1);
% Gives all time points that left LED is on
   %result is a vector 1 x 80299
times.lLEDoff = find(lLED == 0);
% Gives all time points that left LED is off
   %result is a vector 1 x 1119941
times.rLEDon = find(rLED == 1);
times.rLEDoff = find(rLED == 0);
% the above code will break out the time points when the LED is on for each
% side and when it is off. Next step:
%         Need to ask it to count how many times ui.spikes takes place
%         during each LEDon segment

%% gets light "on" times into one array
times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
   %results in three specific time points for this LED
times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
% this turns the times.xLEDon into a list of the points when the LED turned
% on ***Use this for making a raster plot****
   %results in two specific time points for this LED


   %% create raster plots-Left Stim
   % Error: Subscript indices must either be real ve integers or
   % logicals.
```

```
    for l = 1:length(times.lLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
    end

    %In original script, variable that's equivalent to 'times.lLEDon' is e.g.
    % a 23x1 double that includes only the start times for light turning
    % on. Maybe be better to use times.lLEDstart?
    %times.lLedon in this script gives the chunks when led was on, aka a
    %1x80299 double

    t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
    t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
    % creates time vector for raster

    times.Lrasterlight = 1:(length(times.lLEDstart));
    t.Lrasterlight = ones(1,(length(times.lLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Lraster = horzcat(ui.Lrastercell{:});
    % concatenates cell array into a double

    Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
    % creates transform for stacking windowed data for the raster plot

    %% create raster plots-Right Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for r = 1:length(times.rLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
    end


    t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
    t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
    % creates time vector for raster
```

```
    times.Rrasterlight = 1:(length(times.rLEDstart));
    t.Rrasterlight = ones(1,(length(times.rLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Rraster = horzcat(ui.Rrastercell{:});
    % concatenates cell array into a double

    Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
    % creates transform for stacking windowed data for the raster plot


%% Before running next part, need to find how many times light goes on,
% do this by checking the variable "times.lLEDstart" and "times.rLEDstart".
% It will either show the exact values for the start times or will indicate
% how many different light on times there are, if trials exceeds ~5.
times


%% this number needs to be input as last value in reshape function below:


     ui.LrasterStack = reshape(ui.Lraster,20001,3);
     ui.RrasterStack = reshape(ui.Rraster,20001,2);
%% Check plot to verify reshape has been applied appropriately to LEFT data:

    figure % creates raster plot
    plot(t.Lraster,ui.LrasterStack+Lstack);
    hold on
    plot(t.Lrasterlight,times.Lrasterlight,'-.black');
    hold off
    ylabel 'trial number'
    xlabel 'time (s)'

%% Check plot to verify reshape has been applied appropriately to RIGHT data:

    figure % creates raster plot
    plot(t.Rraster,ui.RrasterStack+Rstack);
    hold on
    plot(t.Rrasterlight,times.Rrasterlight,'-.black');
    hold off
```

```
    ylabel 'trial number'
    xlabel 'time (s)'
%% Lastly, get spike averages for each eye
     stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Laveon
stats.spikes.Laveoff

%% Lastly, get spike avwerages for each eye
     stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Raveon
stats.spikes.Raveoff
```

10. read_Intan_RHD2000_file.m

```
function read_Intan_RHD2000_file

% read_Intan_RHD2000_file
%
% Version 1.3, 10 December 2013
%
% Reads Intan Technologies RHD2000 data file generated by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
% base workspace.
```

```
%
% Example:
% >> clear
% >> read_Intan_RHD200_file
% >> whos
% >> amplifier_channels(1)
% >> plot(t_amplifier, amplifier_data(1,:))

[file, path, filterindex] = uigetfile('*.rhd', 'Select an RHD2000 Data File',
'MultiSelect', 'off');

% Read most recent file automatically.
%path = 'C:\Users\Reid\Documents\RHD2132\testing\';
%d = dir([path '*.rhd']);
%file = d(end).name;

tic;
filename = [path,file];
fid = fopen(filename, 'r');

s = dir(filename);
filesize = s.bytes;

% Check 'magic number' at beginning of file to make sure this is an Intan
% Technologies RHD2000 data file.
magic_number = fread(fid, 1, 'uint32');
if magic_number ~= hex2dec('c6912702')
    error('Unrecognized file type.');
end

% Read version number.
data_file_main_version_number = fread(fid, 1, 'int16');
data_file_secondary_version_number = fread(fid, 1, 'int16');

fprintf(1, '\n');
fprintf(1, 'Reading Intan Technologies RHD2000 Data File,
Version %d.%d\n', ...
    data_file_main_version_number, data_file_secondary_version_number);
fprintf(1, '\n');

% Read information of sampling rate and amplifier frequency settings.
```

```
sample_rate = fread(fid, 1, 'single');
dsp_enabled = fread(fid, 1, 'int16');
actual_dsp_cutoff_frequency = fread(fid, 1, 'single');
actual_lower_bandwidth = fread(fid, 1, 'single');
actual_upper_bandwidth = fread(fid, 1, 'single');

desired_dsp_cutoff_frequency = fread(fid, 1, 'single');
desired_lower_bandwidth = fread(fid, 1, 'single');
desired_upper_bandwidth = fread(fid, 1, 'single');

% This tells us if a software 50/60 Hz notch filter was enabled during
% the data acquisition.
notch_filter_mode = fread(fid, 1, 'int16');
notch_filter_frequency = 0;
if (notch_filter_mode == 1)
    notch_filter_frequency = 50;
elseif (notch_filter_mode == 2)
    notch_filter_frequency = 60;
end

desired_impedance_test_frequency = fread(fid, 1, 'single');
actual_impedance_test_frequency = fread(fid, 1, 'single');

% Place notes in data strucure
notes = struct( ...
    'note1', fread_QString(fid), ...
    'note2', fread_QString(fid), ...
    'note3', fread_QString(fid) );

% If data file is from GUI v1.1 or later, see if temperature sensor data
% was saved.
num_temp_sensor_channels = 0;
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 1) ...
    || (data_file_main_version_number > 1))
    num_temp_sensor_channels = fread(fid, 1, 'int16');
end

% If data file is from GUI v1.3 or later, load eval board mode.
eval_board_mode = 0;
```

```
if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 3) ...
    || (data_file_main_version_number > 1))
    eval_board_mode = fread(fid, 1, 'int16');
end

% Place frequency-related information in data structure.
frequency_parameters = struct( ...
    'amplifier_sample_rate', sample_rate, ...
    'aux_input_sample_rate', sample_rate / 4, ...
    'supply_voltage_sample_rate', sample_rate / 60, ...
    'board_adc_sample_rate', sample_rate, ...
    'board_dig_in_sample_rate', sample_rate, ...
    'desired_dsp_cutoff_frequency', desired_dsp_cutoff_frequency, ...
    'actual_dsp_cutoff_frequency', actual_dsp_cutoff_frequency, ...
    'dsp_enabled', dsp_enabled, ...
    'desired_lower_bandwidth', desired_lower_bandwidth, ...
    'actual_lower_bandwidth', actual_lower_bandwidth, ...
    'desired_upper_bandwidth', desired_upper_bandwidth, ...
    'actual_upper_bandwidth', actual_upper_bandwidth, ...
    'notch_filter_frequency', notch_filter_frequency, ...
    'desired_impedance_test_frequency',
desired_impedance_test_frequency, ...
    'actual_impedance_test_frequency', actual_impedance_test_frequency );

% Define data structure for spike trigger settings.
spike_trigger_struct = struct( ...
    'voltage_trigger_mode', {}, ...
    'voltage_threshold', {}, ...
    'digital_trigger_channel', {}, ...
    'digital_edge_polarity', {} );

new_trigger_channel = struct(spike_trigger_struct);
spike_triggers = struct(spike_trigger_struct);

% Define data structure for data channels.
channel_struct = struct( ...
    'native_channel_name', {}, ...
    'custom_channel_name', {}, ...
    'native_order', {}, ...
    'custom_order', {}, ...
```

```
    'board_stream', {}, ...
    'chip_channel', {}, ...
    'port_name', {}, ...
    'port_prefix', {}, ...
    'port_number', {}, ...
    'electrode_impedance_magnitude', {}, ...
    'electrode_impedance_phase', {} );

new_channel = struct(channel_struct);

% Create structure arrays for each type of data channel.
amplifier_channels = struct(channel_struct);
aux_input_channels = struct(channel_struct);
supply_voltage_channels = struct(channel_struct);
board_adc_channels = struct(channel_struct);
board_dig_in_channels = struct(channel_struct);
board_dig_out_channels = struct(channel_struct);

amplifier_index = 1;
aux_input_index = 1;
supply_voltage_index = 1;
board_adc_index = 1;
board_dig_in_index = 1;
board_dig_out_index = 1;

% Read signal summary from data file header.

number_of_signal_groups = fread(fid, 1, 'int16');

for signal_group = 1:number_of_signal_groups
    signal_group_name = fread_QString(fid);
    signal_group_prefix = fread_QString(fid);
    signal_group_enabled = fread(fid, 1, 'int16');
    signal_group_num_channels = fread(fid, 1, 'int16');
    signal_group_num_amp_channels = fread(fid, 1, 'int16');

    if (signal_group_num_channels > 0 && signal_group_enabled > 0)
        new_channel(1).port_name = signal_group_name;
        new_channel(1).port_prefix = signal_group_prefix;
        new_channel(1).port_number = signal_group;
        for signal_channel = 1:signal_group_num_channels
```

```
            new_channel(1).native_channel_name = fread_QString(fid);
            new_channel(1).custom_channel_name = fread_QString(fid);
            new_channel(1).native_order = fread(fid, 1, 'int16');
            new_channel(1).custom_order = fread(fid, 1, 'int16');
            signal_type = fread(fid, 1, 'int16');
            channel_enabled = fread(fid, 1, 'int16');
            new_channel(1).chip_channel = fread(fid, 1, 'int16');
            new_channel(1).board_stream = fread(fid, 1, 'int16');
            new_trigger_channel(1).voltage_trigger_mode = fread(fid, 1,
'int16');
            new_trigger_channel(1).voltage_threshold = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_trigger_channel = fread(fid, 1,
'int16');
            new_trigger_channel(1).digital_edge_polarity = fread(fid, 1,
'int16');
            new_channel(1).electrode_impedance_magnitude = fread(fid, 1,
'single');
            new_channel(1).electrode_impedance_phase = fread(fid, 1,
'single');

            if (channel_enabled)
                switch (signal_type)
                    case 0
                        amplifier_channels(amplifier_index) = new_channel;
                        spike_triggers(amplifier_index) =
new_trigger_channel;
                        amplifier_index = amplifier_index + 1;
                    case 1
                        aux_input_channels(aux_input_index) = new_channel;
                        aux_input_index = aux_input_index + 1;
                    case 2
                        supply_voltage_channels(supply_voltage_index) =
new_channel;
                        supply_voltage_index = supply_voltage_index + 1;
                    case 3
                        board_adc_channels(board_adc_index) = new_channel;
                        board_adc_index = board_adc_index + 1;
                    case 4
                        board_dig_in_channels(board_dig_in_index) =
new_channel;
```

```
                    board_dig_in_index = board_dig_in_index + 1;
                case 5
                    board_dig_out_channels(board_dig_out_index) =
new_channel;
                    board_dig_out_index = board_dig_out_index + 1;
                otherwise
                    error('Unknown channel type');
            end
        end

    end
end
end

% Summarize contents of data file.
num_amplifier_channels = amplifier_index - 1;
num_aux_input_channels = aux_input_index - 1;
num_supply_voltage_channels = supply_voltage_index - 1;
num_board_adc_channels = board_adc_index - 1;
num_board_dig_in_channels = board_dig_in_index - 1;
num_board_dig_out_channels = board_dig_out_index - 1;

fprintf(1, 'Found %d amplifier channel%s.\n', ...
    num_amplifier_channels, plural(num_amplifier_channels));
fprintf(1, 'Found %d auxiliary input channel%s.\n', ...
    num_aux_input_channels, plural(num_aux_input_channels));
fprintf(1, 'Found %d supply voltage channel%s.\n', ...
    num_supply_voltage_channels, plural(num_supply_voltage_channels));
fprintf(1, 'Found %d board ADC channel%s.\n', ...
    num_board_adc_channels, plural(num_board_adc_channels));
fprintf(1, 'Found %d board digital input channel%s.\n', ...
    num_board_dig_in_channels, plural(num_board_dig_in_channels));
fprintf(1, 'Found %d board digital output channel%s.\n', ...
    num_board_dig_out_channels, plural(num_board_dig_out_channels));
fprintf(1, 'Found %d temperature sensors channel%s.\n', ...
    num_temp_sensor_channels, plural(num_temp_sensor_channels));
fprintf(1, '\n');

% Determine how many samples the data file contains.

% Each data block contains 60 amplifier samples.
```

```
bytes_per_block = 60 * 4;  % timestamp data
bytes_per_block = bytes_per_block + 60 * 2 * num_amplifier_channels;
% Auxiliary inputs are sampled 4x slower than amplifiers
bytes_per_block = bytes_per_block + 15 * 2 * num_aux_input_channels;
% Supply voltage is sampled 60x slower than amplifiers
bytes_per_block = bytes_per_block + 1 * 2 * num_supply_voltage_channels;
% Board analog inputs are sampled at same rate as amplifiers
bytes_per_block = bytes_per_block + 60 * 2 * num_board_adc_channels;
% Board digital inputs are sampled at same rate as amplifiers
if (num_board_dig_in_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Board digital outputs are sampled at same rate as amplifiers
if (num_board_dig_out_channels > 0)
    bytes_per_block = bytes_per_block + 60 * 2;
end
% Temp sensor is sampled 60x slower than amplifiers
if (num_temp_sensor_channels > 0)
   bytes_per_block = bytes_per_block + 1 * 2 * num_temp_sensor_channels;
end

% How many data blocks remain in this file?
data_present = 0;
bytes_remaining = filesize - ftell(fid);
if (bytes_remaining > 0)
    data_present = 1;
end

num_data_blocks = bytes_remaining / bytes_per_block;

num_amplifier_samples = 60 * num_data_blocks;
num_aux_input_samples = 15 * num_data_blocks;
num_supply_voltage_samples = 1 * num_data_blocks;
num_board_adc_samples = 60 * num_data_blocks;
num_board_dig_in_samples = 60 * num_data_blocks;
num_board_dig_out_samples = 60 * num_data_blocks;

record_time = num_amplifier_samples / sample_rate;

if (data_present)
```

```
    fprintf(1, 'File contains %0.3f seconds of data.  Amplifiers were sampled
at %0.2f kS/s.\n', ...
        record_time, sample_rate / 1000);
    fprintf(1, '\n');
else
    fprintf(1, 'Header file contains no data.  Amplifiers were sampled
at %0.2f kS/s.\n', ...
        sample_rate / 1000);
    fprintf(1, '\n');
end

if (data_present)

    % Pre-allocate memory for data.
    fprintf(1, 'Allocating memory for data...\n');

    t_amplifier = zeros(1, num_amplifier_samples);

    amplifier_data = zeros(num_amplifier_channels, num_amplifier_samples);
    aux_input_data = zeros(num_aux_input_channels, num_aux_input_samples);
    supply_voltage_data = zeros(num_supply_voltage_channels,
num_supply_voltage_samples);
    temp_sensor_data = zeros(num_temp_sensor_channels,
num_supply_voltage_samples);
    board_adc_data = zeros(num_board_adc_channels, num_board_adc_samples);
    board_dig_in_data = zeros(num_board_dig_in_channels,
num_board_dig_in_samples);
    board_dig_in_raw = zeros(1, num_board_dig_in_samples);
    board_dig_out_data = zeros(num_board_dig_out_channels,
num_board_dig_out_samples);
    board_dig_out_raw = zeros(1, num_board_dig_out_samples);

    % Read sampled data from file.
    fprintf(1, 'Reading data from file...\n');

    amplifier_index = 1;
    aux_input_index = 1;
    supply_voltage_index = 1;
    board_adc_index = 1;
    board_dig_in_index = 1;
    board_dig_out_index = 1;
```

```
    print_increment = 10;
    percent_done = print_increment;
    for i=1:num_data_blocks
        % In version 1.2, we moved from saving timestamps as unsigned
        % integeters to signed integers to accomidate negative (adjusted)
        % timestamps for pretrigger data.
        if ((data_file_main_version_number == 1 &&
data_file_secondary_version_number >= 2) ...
        || (data_file_main_version_number > 1))
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'int32');
        else
            t_amplifier(amplifier_index:(amplifier_index+59)) = fread(fid,
60, 'uint32');
        end
        if (num_amplifier_channels > 0)
            amplifier_data(:, amplifier_index:(amplifier_index+59)) =
fread(fid, [60, num_amplifier_channels], 'uint16')';
        end
        if (num_aux_input_channels > 0)
            aux_input_data(:, aux_input_index:(aux_input_index+14)) =
fread(fid, [15, num_aux_input_channels], 'uint16')';
        end
        if (num_supply_voltage_channels > 0)
            supply_voltage_data(:, supply_voltage_index) = fread(fid, [1,
num_supply_voltage_channels], 'uint16')';
        end
        if (num_temp_sensor_channels > 0)
            temp_sensor_data(:, supply_voltage_index) = fread(fid, [1,
num_temp_sensor_channels], 'int16')';
        end
        if (num_board_adc_channels > 0)
            board_adc_data(:, board_adc_index:(board_adc_index+59)) =
fread(fid, [60, num_board_adc_channels], 'uint16')';
        end
        if (num_board_dig_in_channels > 0)
            board_dig_in_raw(board_dig_in_index:(board_dig_in_index+59)) =
fread(fid, 60, 'uint16');
        end
        if (num_board_dig_out_channels > 0)
```

```
board_dig_out_raw(board_dig_out_index:(board_dig_out_index+59)) =
fread(fid, 60, 'uint16');
        end

        amplifier_index = amplifier_index + 60;
        aux_input_index = aux_input_index + 15;
        supply_voltage_index = supply_voltage_index + 1;
        board_adc_index = board_adc_index + 60;
        board_dig_in_index = board_dig_in_index + 60;
        board_dig_out_index = board_dig_out_index + 60;

        fraction_done = 100 * (i / num_data_blocks);
        if (fraction_done >= percent_done)
            fprintf(1, '%d%% done...\n', percent_done);
            percent_done = percent_done + print_increment;
        end
    end

    % Make sure we have read exactly the right amount of data.
    bytes_remaining = filesize - ftell(fid);
    if (bytes_remaining ~= 0)
        %error('Error: End of file not reached.');
    end

end

% Close data file.
fclose(fid);

if (data_present)

    fprintf(1, 'Parsing data...\n');

    % Extract digital input channels to separate variables.
    for i=1:num_board_dig_in_channels
      mask = 2^(board_dig_in_channels(i).native_order) *
ones(size(board_dig_in_raw));
      board_dig_in_data(i, :) = (bitand(board_dig_in_raw, mask) > 0);
    end
    for i=1:num_board_dig_out_channels
```

```
      mask = 2^(board_dig_out_channels(i).native_order) *
ones(size(board_dig_out_raw));
      board_dig_out_data(i, :) = (bitand(board_dig_out_raw, mask) > 0);
   end

   % Scale voltage levels appropriately.
   amplifier_data = 0.195 * (amplifier_data - 32768); % units = microvolts
   aux_input_data = 37.4e-6 * aux_input_data; % units = volts
   supply_voltage_data = 74.8e-6 * supply_voltage_data; % units = volts
   if (eval_board_mode == 1)
      board_adc_data = 152.59e-6 * (board_adc_data - 32768); % units = volts
   else
      board_adc_data = 50.354e-6 * board_adc_data; % units = volts
   end
   temp_sensor_data = temp_sensor_data / 100; % units = deg C

   % Check for gaps in timestamps.
   num_gaps = sum(diff(t_amplifier) ~= 1);
   if (num_gaps == 0)
      fprintf(1, 'No missing timestamps in data.\n');
   else
      fprintf(1, 'Warning: %d gaps in timestamp data found.  Time scale will
not be uniform!\n', ...
          num_gaps);
   end

   % Scale time steps (units = seconds).
   t_amplifier = t_amplifier / sample_rate;
   t_aux_input = t_amplifier(1:4:end);
   t_supply_voltage = t_amplifier(1:60:end);
   t_board_adc = t_amplifier;
   t_dig = t_amplifier;
   t_temp_sensor = t_supply_voltage;

   % If the software notch filter was selected during the recording, apply
the
   % same notch filter to amplifier data here.
   if (notch_filter_frequency > 0)
      fprintf(1, 'Applying notch filter...\n');

      print_increment = 10;
```

```
        percent_done = print_increment;
        for i=1:num_amplifier_channels
            amplifier_data(i,:) = ...
                notch_filter(amplifier_data(i,:), sample_rate,
notch_filter_frequency, 10);

            fraction_done = 100 * (i / num_amplifier_channels);
            if (fraction_done >= percent_done)
                fprintf(1, '%d%% done...\n', percent_done);
                percent_done = percent_done + print_increment;
            end

        end
    end

end

% Move variables to base workspace.

move_to_base_workspace(notes);
move_to_base_workspace(frequency_parameters);

if (num_amplifier_channels > 0)
    move_to_base_workspace(amplifier_channels);
    if (data_present)
        move_to_base_workspace(amplifier_data);
        move_to_base_workspace(t_amplifier);
    end
    move_to_base_workspace(spike_triggers);
end
if (num_aux_input_channels > 0)
    move_to_base_workspace(aux_input_channels);
    if (data_present)
        move_to_base_workspace(aux_input_data);
        move_to_base_workspace(t_aux_input);
    end
end
if (num_supply_voltage_channels > 0)
    move_to_base_workspace(supply_voltage_channels);
    if (data_present)
        move_to_base_workspace(supply_voltage_data);
```

```
        move_to_base_workspace(t_supply_voltage);
    end
end
if (num_board_adc_channels > 0)
    move_to_base_workspace(board_adc_channels);
    if (data_present)
        move_to_base_workspace(board_adc_data);
        move_to_base_workspace(t_board_adc);
    end
end
if (num_board_dig_in_channels > 0)
    move_to_base_workspace(board_dig_in_channels);
    if (data_present)
        move_to_base_workspace(board_dig_in_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_board_dig_out_channels > 0)
    move_to_base_workspace(board_dig_out_channels);
    if (data_present)
        move_to_base_workspace(board_dig_out_data);
        move_to_base_workspace(t_dig);
    end
end
if (num_temp_sensor_channels > 0)
    if (data_present)
        move_to_base_workspace(temp_sensor_data);
        move_to_base_workspace(t_temp_sensor);
    end
end

fprintf(1, 'Done!  Elapsed time: %0.1f seconds\n', toc);
if (data_present)
    fprintf(1, 'Extracted data are now available in the MATLAB
workspace.\n');
else
    fprintf(1, 'Extracted waveform information is now available in the MATLAB
workspace.\n');
end
fprintf(1, 'Type ''whos'' to see variables.\n');
fprintf(1, '\n');
```

```
return


function a = fread_QString(fid)

% a = read_QString(fid)
%
% Read Qt style QString.  The first 32-bit unsigned number indicates
% the length of the string (in bytes).  If this number equals 0xFFFFFFFF,
% the string is null.

a = '';
length = fread(fid, 1, 'uint32');
if length == hex2num('ffffffff')
   return;
end
% convert length from bytes to 16-bit Unicode words
length = length / 2;

for i=1:length
   a(i) = fread(fid, 1, 'uint16');
end

return


function s = plural(n)

% s = plural(n)
%
% Utility function to optionally plurailze words based on the value
% of n.

if (n == 1)
   s = '';
else
   s = 's';
end

return
```

```
function out = notch_filter(in, fSample, fNotch, Bandwidth)

% out = notch_filter(in, fSample, fNotch, Bandwidth)
%
% Implements a notch filter (e.g., for 50 or 60 Hz) on vector 'in'.
% fSample = sample rate of data (in Hz or Samples/sec)
% fNotch = filter notch frequency (in Hz)
% Bandwidth = notch 3-dB bandwidth (in Hz).  A bandwidth of 10 Hz is
%   recommended for 50 or 60 Hz notch filters; narrower bandwidths lead to
%   poor time-domain properties with an extended ringing response to
%   transient disturbances.
%
% Example:  If neural data was sampled at 30 kSamples/sec
% and you wish to implement a 60 Hz notch filter:
%
% out = notch_filter(in, 30000, 60, 10);

tstep = 1/fSample;
Fc = fNotch*tstep;

L = length(in);

% Calculate IIR filter parameters
d = exp(-2*pi*(Bandwidth/2)*tstep);
b = (1 + d*d)*cos(2*pi*Fc);
a0 = 1;
a1 = -b;
a2 = d*d;
a = (1 + d*d)/2;
b0 = 1;
b1 = -2*cos(2*pi*Fc);
b2 = 1;

out = zeros(size(in));
out(1) = in(1);
out(2) = in(2);
% (If filtering a continuous data stream, change out(1) and out(2) to the
%  previous final two values of out.)
```

```
% Run filter
for i=3:L
   out(i) = (a*b2*in(i-2) + a*b1*in(i-1) + a*b0*in(i) - a2*out(i-2) -
a1*out(i-1))/a0;
end

return


function move_to_base_workspace(variable)

% move_to_base_workspace(variable)
%
% Move variable from function workspace to base MATLAB workspace so
% user will have access to it after the program ends.

variable_name = inputname(1);
assignin('base', variable_name, variable);

return;
```

11. comment_160330_IntanEphysAnalysis.m

```
%% Begin with opening file to be analyzed
clear all %Starting a new analysis so we want to eliminate all old variables
close all
%
%First we import the data using:
%read_Intan_RHD2000_file = Opens the Matlab file browser UI to locate the
%file of interest. Afterward it reads header info and establishes basic
%variables from the .rhd file
%
read_Intan_RHD2000_file
%
%From function info: % Reads Intan Technologies RHD2000 data file generated
by evaluation board
% GUI.  Data are parsed and placed into variables that appear in the base
% MATLAB workspace.  Therefore, it is recommended to execute a 'clear'
% command before running this program to clear all other variables from the
```

```
% base workspace.

%check what variables have been imported, especially if youre unsure
%whether accessory amplifier channels were disabled or not during recording
%% Establishing some basic variables from values pulled in by above function

amplifier_channels(1) %Channel data is being collected on (on
%preamplifier this would be 'A-004'
%Above command gives data output about the channel on which data is being
%collected


%%%%%%%% Can we find out automatically which channel is the one?



%% Variable name changes below to simplify:

tRat = t_amplifier; %time variable for ephys data
tLED = t_dig; %time variable for LED data
ui.ratData = amplifier_data(1,:);
lLED = board_dig_in_data(1,:);
rLED = board_dig_in_data(2,:);

% %% Check variables look right-- plot should be identical to last one
% figure % for spike detection
%     hold on
%     plot(tRat, ui.ratData,'blue')
%     plot(tLED,lLED,'red')  %max makes red lines continue across top half
of vertical axis
%     plot(tLED,rLED,'green')
%     xlabel 'time (s)'
%     ylabel 'amplitude (A.U.)'
%     legend('Raw Data', 'Left Eye LED','Right Eye LED')

%% filter data
     Wn = 300/10000;                % Normalized cutoff frequency
    [b,a] = butter(5,Wn,'high');

    ui.ratData = filtfilt(b,a,ui.ratData);
    %% invert signal data for thresholding
```

```
    ui.ratData = ui.ratData.*(-1);


    %% Setting threshold for spikes and finding light ON times
threshold = 20;
Fs =  20000; %amplifier_sample_rate
windowSize = Fs * 0.05; %creates our time interval by taking the 20k
% sampling rate at which the data was collected and converts it to
% timestamps collected every millisecond, in other words the value of
% windowSize is 1 ms.


% Finding all spikes in recording:
    ui.spikes = diff(ui.ratData > threshold) > 0.1;


%% Plot Again with spikes showing & correct time axis now:

t=length(tRat)-1
figure % for spike detection
    hold on
    plot(tRat(1:t), ui.ratData(1:t),'blue')
    plot(tRat(1:t), ui.spikes*max(ui.ratData),'black')
    plot(tLED(1:t),lLED(1:t)*80,'green')  %max makes red lines continue
across top half of vertical axis
    plot(tLED(1:t),rLED(1:t)*80,'red')
    xlabel 'time (s)'
    ylabel 'amplitude (A.U.)'
    legend('Raw Data', 'Spikes', 'Left Eye LED','Right Eye LED')

%%%%%%%% Can we use better visualization?

%% Count spikes during each LED stimulation

 SpikesL = sum(ui.spikes.*lLED(1:end-1))
 SpikesR = sum(ui.spikes.*rLED(1:end-1))

%%%%%%%% Can we use average spikes as an index?

%%   This creates a whole lot of extra light related variables, but unsure
if they are actually useful
```

```
    lightstim = 199; %change?
       ui.leftLEDon = diff(lLED < -lightstim)>0.1;
       ui.rightLEDon = diff(rLED < -lightstim)>0.1;
%        times.leftLED = find(leftLED == 500);
%        times.rightLED = find(rightLED == 500);
%rewrite:
times.lLEDon = find(lLED == 1);
% Gives all time points that left LED is on
   %result is a vector 1 x 80299
times.lLEDoff = find(lLED == 0);
% Gives all time points that left LED is off
   %result is a vector 1 x 1119941
times.rLEDon = find(rLED == 1);
times.rLEDoff = find(rLED == 0);
% the above code will break out the time points when the LED is on for each
% side and when it is off. Next step:
%         Need to ask it to count how many times ui.spikes takes place
%         during each LEDon segment


%%%%%%%% What can we find in times?


%% gets light "on" times into one array
times.lLEDstart = times.lLEDon(diff(times.lLEDon)>Fs*0.05);
   %results in three specific time points for this LED
times.rLEDstart = times.rLEDon(diff(times.rLEDon)>Fs*0.05);
% this turns the times.xLEDon into a list of the points when the LED turned
% on ***Use this for making a raster plot****
   %results in two specific time points for this LED


%%%%%%%% How can we translate the times in recording into ms?


   %% create raster plots-Left Stim
   % Error: Subscript indices must either be real ve integers or
   % logicals.
   for l = 1:length(times.lLEDstart)
      % collects window of data each time the light stimulus initiated
```

```
        windowSize = round(Fs*0.5); % window size in samples
        ui.Lrastercell{l} = ui.spikes(times.lLEDstart(l) -
windowSize:times.lLEDstart(l) + windowSize);
    end

    %In original script, variable that's equivalent to 'times.lLEDon' is e.g.
    % a 23x1 double that includes only the start times for light turning
    % on. Maybe be better to use times.lLEDstart?
    %times.lLedon in this script gives the chunks when led was on, aka a
    %1x80299 double

    t.Lraster = transpose((1:length(ui.Lrastercell{1}(:)))/Fs);
    t.Lraster = repmat(t.Lraster,1,length(times.lLEDstart));
    % creates time vector for raster

    times.Lrasterlight = 1:(length(times.lLEDstart));
    t.Lrasterlight = ones(1,(length(times.lLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Lraster = horzcat(ui.Lrastercell{:});
    % concatenates cell array into a double

    Lstack = repmat(1:length(times.lLEDstart),length(t.Lraster),1);
    % creates transform for stacking windowed data for the raster plot


%%%%%%%% Can we better visualization?


    %% create raster plots-Right Stim
    % Error: Subscript indices must either be real ve integers or
    % logicals.
    for r = 1:length(times.rLEDstart)
        % collects window of data each time the light stimulus initiated
        windowSize = round(Fs*0.5); % window size in samples
        ui.Rrastercell{r} = ui.spikes(times.rLEDstart(r) -
windowSize:times.rLEDstart(r) + windowSize);
    end


    t.Rraster = transpose((1:length(ui.Rrastercell{1}(:)))/Fs);
```

```
    t.Rraster = repmat(t.Rraster,1,length(times.rLEDstart));
    % creates time vector for raster

    times.Rrasterlight = 1:(length(times.rLEDstart));
    t.Rrasterlight = ones(1,(length(times.rLEDstart)))*windowSize/Fs;
    % creates dashed line for indicating stim onset on raster plot

    ui.Rraster = horzcat(ui.Rrastercell{:});
    % concatenates cell array into a double

    Rstack = repmat(1:length(times.rLEDstart),length(t.Rraster),1);
    % creates transform for stacking windowed data for the raster plot


%% Before running next part, need to find how many times light goes on,
% do this by checking the variable "times.lLEDstart" and "times.rLEDstart".
% It will either show the exact values for the start times or will indicate
% how many different light on times there are, if trials exceeds ~5.
times

%%%%%%%% Can we combine different trials?

%% this number needs to be input as last value in reshape function below:


     ui.LrasterStack = reshape(ui.Lraster,20001,3);
     ui.RrasterStack = reshape(ui.Rraster,20001,2);
%% Check plot to verify reshape has been applied appropriately to LEFT data:

    figure % creates raster plot
    plot(t.Lraster,ui.LrasterStack+Lstack);
    hold on
    plot(t.Lrasterlight,times.Lrasterlight,'-.black');
    hold off
    ylabel 'trial number'
    xlabel 'time (s)'

%% Check plot to verify reshape has been applied appropriately to RIGHT data:

    figure % creates raster plot
    plot(t.Rraster,ui.RrasterStack+Rstack);
```

```
    hold on
    plot(t.Rrasterlight,times.Rrasterlight,'-.black');
    hold off
    ylabel 'trial number'
    xlabel 'time (s)'
%% Lastly, get spike averages for each eye
     stats.spikes.Laveon =
sum(sum(ui.LrasterStack(windowSize:end,:)))/length(times.lLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Laveoff =
sum(sum(ui.LrasterStack(1:windowSize,:)))/length(times.lLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Laveon
stats.spikes.Laveoff


%% Lastly, get spike avwerages for each eye
     stats.spikes.Raveon =
sum(sum(ui.RrasterStack(windowSize:end,:)))/length(times.rLEDstart);
%     % calculates average number of spikes after light turned on
     stats.spikes.Raveoff =
sum(sum(ui.RrasterStack(1:windowSize,:)))/length(times.rLEDstart);
%     % calculates average number of spikes preceding light onset


stats.spikes.Raveon
stats.spikes.Raveoff


%%%%%%%% What can we do with these averages? Combine?
```

Bibliography:

Intan Technologies, http://intantech.com/RHD2000_evaluation_system.html

Driscoll, T. A. (2009). *Learning MATLAB*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Feng, J. (2004). *Computational neuroscience: Comprehensive approach*. Boca Raton: Chapman & Hall/CRC.

Special Acknowledgement:

Thank Prof. Jaime Olavarria and Dr. Adrian Andelin for giving me the opportunity to analyze his data! In order to present his the most convincing data, I am lucky to review, utilize and summarize my ability of information searching, computational modeling and conception actualization. In the entire process of developing ideas into software, I learnt a lot from the questions and reflections at every step and had the fortune to enjoy the excitement of tackling the challenges one by one! His tolerance on the deadline and encouragement on the difficulty give me the strongest fortitude to proceed!

Thank Prof. Jaime Olavarria for introducing me with insightful lectures into this dynamic field of biopsychology and neuroscience and letting me getting involved in the challenging projects of his warm lab! (It was also my honor to be able to be peer TA in his class. Thank him to give me courage to challenge Psychology Honors Program.)

Thank University of Washington for giving us the platform to scientifically explore practical academic problems interdisciplinarily!

I will continue the voyage of exploring the infinite realm of computational neuroscience in my academic career fearlessly.

Baihan Lin
April 2016