

# Acquipix User Manual

Version 0.3, 20210727

By Dr. Jorrit Montijn

Cortical Structure and Function group

Netherlands Institute for Neuroscience

E-mail: [j.montijn@nin.knaw.nl](mailto:j.montijn@nin.knaw.nl)

# Contents

Overview.....	3
Installation instructions.....	4
Before you start.....	4
Installing the external libraries for the recording computer.....	4
Installing the external libraries for preprocessing your data .....	4
Installing Acquipix and its dependencies .....	4
Workflow summary .....	5
User guide.....	6
Running an experiment .....	6
Using the online analysis tools .....	6
Preprocessing your data.....	7
Setting the variables and parameters .....	7
Compiling the library and pre-processing the data.....	9
Adjusting the probe coordinates and exporting the data .....	10
Technical descriptions.....	12
Experiment scripts.....	12
Overview.....	12
Description of inputs and structure of an experiment script.....	12
Structure of the experiment script & output file .....	14
Integrating your experiment file into Acquipix .....	16
Online analysis functions.....	16
Overview.....	16
Troubleshooting .....	17

## Overview

Multi-stream data recorded while performing a visual or optogenetic experiment, can be difficult and laborious to process. If your data includes stimulation, multi-channel spiking, LFPs, running speed, and pupil tracking, you will likely already have to deal with 5 independent data streams that need to be synchronized (fig. 1). Moreover, keeping track of all these different files can be a pain. That's why we created the Acquipix repository, which does all the synchronizing and synthesizing for you. All our code is modular, open source and fully customizable, but we made sure you can also run everything out-of-the-box using only graphical user interfaces without having to write a single line of code.

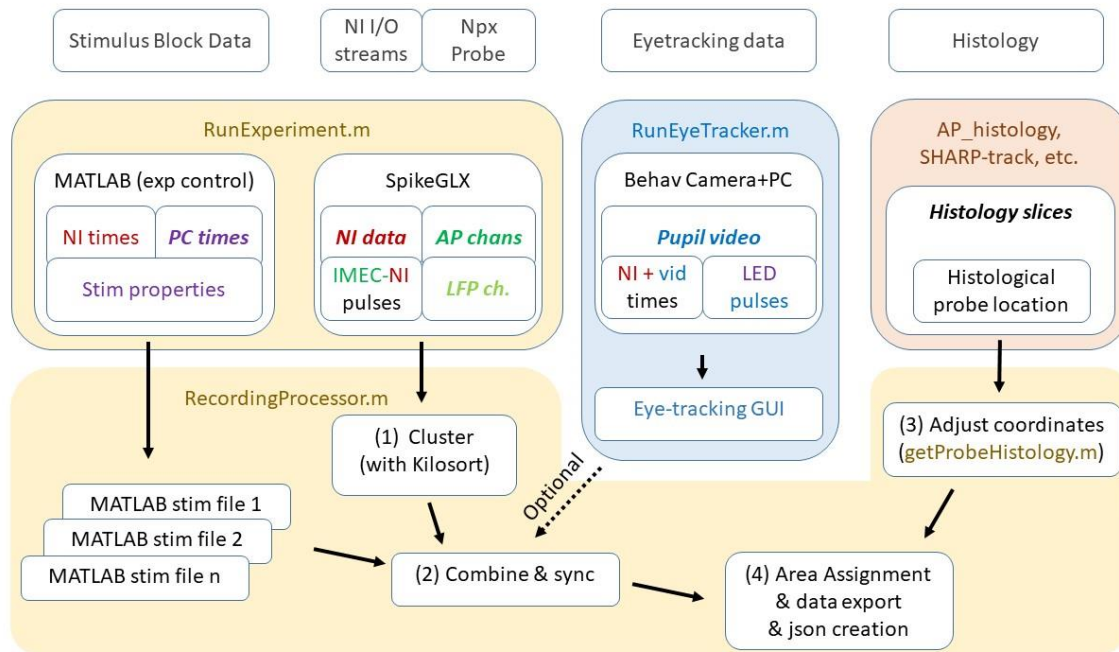


Figure 1. Technical data flowchart showing a single recording. First column from left (*MATLAB (exp control)*): each recording consists of one or more stimulus blocks that contain onset/offset times and stimulus data that are generated using `RunExperiment.m` and its dependency functions. The stimulation scripts automatically record the time stamps of the national instruments (NI) I/O card and internal stimulus times (PC times) generated by Psychtoolbox. Second column (*SpikeGLX*): although the NI I/O streams and IMEC neuropixels (Npx) data may appear to be monolithic, they are entirely separate data streams with independent clock times. SpikeGLX produces data streams at three different sampling frequencies: the NI I/O data, the AP channels, and the downsampled LFP channels. To synchronize these data streams, you need common pulses; i.e., connect the IMEC pulse generator to an NI I/O stream. Third column (*RunEyeTracker*): optionally, you can use the eye-tracker program to perform online eye-tracking and automatically synchronize your pupil tracking movie to the rest of your data. While recording, the eye tracker program periodically queries SpikeGLX for NI data stream time stamps and saves a log file containing synchronous video time stamps and NI time stamps. Moreover, you can define an ROI for luminance-based high-resolution synchronization (LED pulses). Last column (*Histology*): by registering your histology slices to the Allen Brain Atlas using `AP_histology`, `SHARP-track`, or a similar pipeline, you can extract your probe's approximate location. Using the coordinate adjustment GUI you can then fine-tune the coordinates based on electrophysiological markers, such as responsiveness to experimental treatment and the across-channel spiking correlation. The `RecordingProcessor` allows you to easily cluster your spikes with Kilosort (1), combine and synchronize data from multiple sources (2), adjust probe coordinates (3) and automatically extract which brain area each cell was located, using the ABA API (4).

## **Installation instructions**

### **Before you start:**

1. Make sure your computer is up-to-date, and has compatible (not necessarily the newest) CUDA drivers for your Nvidia GPU. If you don't have an Nvidia GPU, you'll have to buy one, because multiple Acquipix functions, as well as the external libraries it uses (Kilosort, Psychtoolbox) make use of CUDA-specific GPU-accelerated computing.
2. You will want a fast SSD for data buffering for various pre-processing operations. You can run everything without an SSD, but it will be noticeably slower.

### **Installing the external libraries for the recording computer:**

1. If you want to run visual experiments on this computer, download Psychtoolbox and follow the installation instructions: <http://psychtoolbox.org/download>
2. If you want to record Neuropixels on this computer, install SpikeGLX: <https://github.com/billkarsh/SpikeGLX>
3. If you want to perform pupil-tracking specifically, or generally want to have accurate & automatic synchronization between your recorded videos, stimulation and Neuropixels data, you can install this Acquipix-independent module on the computer where you're recording the video's: <https://github.com/JorritMontijn/EyeTracker>

### **Installing the external libraries for preprocessing your data:**

1. Download the Kilosort (v3 or v2.5 are both supported), Npy-matlab and spikes repositories and follow their installation instructions: <https://github.com/MouseLand/Kilosort/>, <https://github.com/kwikteam/npymatlab> and <https://github.com/cortex-lab/spikes>. To compile Kilosort's GPU functions, you'll first need to install a compiler. You can for example use visual studio, but note that only specific versions will work in combination with specific versions of matlab: <https://visualstudio.microsoft.com/vs/older-downloads/>. In all cases, make sure you get the **Community** version, and make sure you also install the **C++ compiler**.
2. Compiling Kilosort's CUDA functions with **Visual Studio Community 2015** with Update 3 seems to work with Matlab R2019b, but the version compatibility seems somewhat arcane and random. If your matlab is R2019b or later, try VSC2015 first. If it doesn't work, you can try a different version (i.e., VSC2013).
3. Rename and edit the config and chanmap files to match your preferred settings and copy them to a folder outside the git repositories if you want to make sure they don't get overwritten accidentally

### **Installing Acquipix and its dependencies:**

1. Download the Acquipix repository at <https://github.com/JorritMontijn/Acquipix>
2. Download the GeneralAnalysis and MNCP repositories from <https://github.com/JorritMontijn/>
3. Optional: If you wish to automatically compute a responsiveness metric for your putative single cells, download <https://github.com/JorritMontijn/ZETA>

4. Optional: If you wish to save higher quality figures, download <https://github.com/JorritMontijn/exportFig>

### **Workflow summary:**

To run an experiment:

1. Start SpikeGLX and begin a new acquisition
2. Start “RunExperiment” in MATLAB
3. Select your stimulation and parameters & start the experiment (spikeglx will start recording automatically)

Optional steps for pre-processing:

- A) Pre-process your eye-tracking data
- B) Pre-process your probe coordinates

To pre-process your spiking data with six mouse clicks:

1. Start “RunRecordingProcessor” in MATLAB and compile your data library
2. Select the recording you wish to preprocess
3. Cluster your data by clicking the button in the GUI
4. Combine data from multiple sources by clicking another button using the GUI
5. Adjust the probe coordinates and make some changes if necessary
6. Then export with click #6

## **User guide**

### **Running an experiment**

To run an experiment, run matlab and execute “RunExperiment”

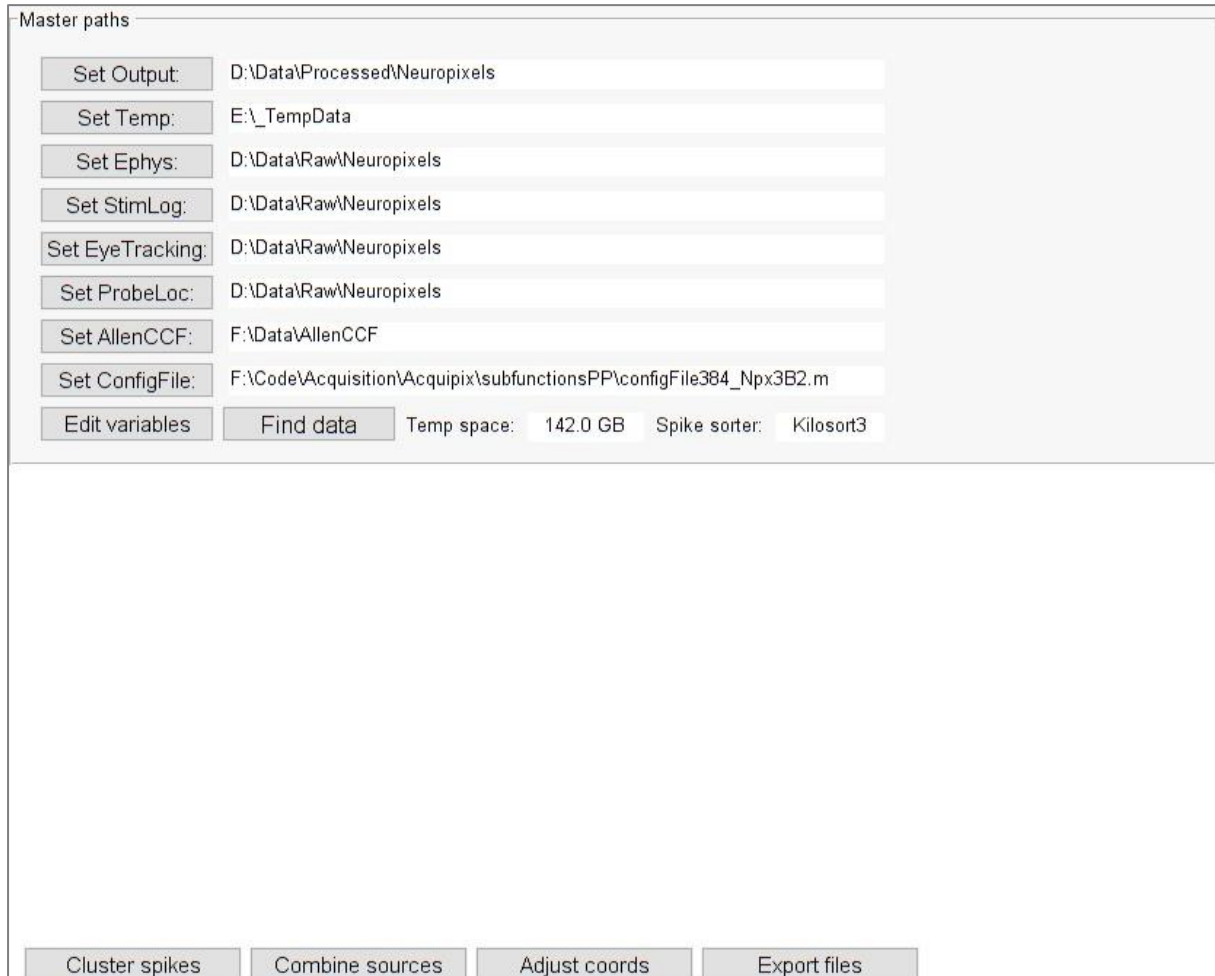
### **Using the online analysis tools**

Acquipix comes with a set of online analysis tools that allow you to stream SpikeGLX data in real-time and calculate MUA tuning curves per channel: runOnlineOT for Orientation Tuning, runOnlineNM for Natural Movies, and runOnlineRF for Receptive Field mapping.

## Preprocessing your data

### *Setting the variables and parameters*

To start preprocessing, run matlab and execute “runRecordingProcessor”. The startup screen will look like this:



The screenshot shows a MATLAB GUI titled "Master paths". It contains several input fields for setting paths and parameters. The fields are arranged in a list, each with a label and a text box. The labels are: "Set Output:", "Set Temp:", "Set Ephys:", "Set StimLog:", "Set EyeTracking:", "Set ProbeLoc:", "Set AllenCCF:", and "Set ConfigFile:". The corresponding text boxes contain the following paths: "D:\Data\Processed\Neuropixels", "E:\\_TempData", "D:\Data\Raw\Neuropixels", "D:\Data\Raw\Neuropixels", "D:\Data\Raw\Neuropixels", "D:\Data\Raw\Neuropixels", "F:\Data\AllenCCF", and "F:\Code\Acquisition\Acquipix\subfunctionsPP\configFile384\_Npx3B2.m". Below these fields are two buttons: "Edit variables" and "Find data". To the right of these buttons are two labels: "Temp space:" and "Spike sorter:". The "Temp space:" label is followed by the text "142.0 GB". The "Spike sorter:" label is followed by the text "Kilosort3". At the bottom of the GUI are four buttons: "Cluster spikes", "Combine sources", "Adjust coords", and "Export files".

To start pre-processing your data, you will first need to set the correct paths:

“Set Output:” specifies where the aggregated and pre-processed data files will be stored.

“Set Temp:” path to fast SSD for data buffering (e.g., used by KiloSort).

“Set Ephys:” root path of your SpikeGLX data files. The actual data files can be in subdirectories. The recommended path structure is something like /root/experiment-set/recordingdate/. To avoid having to compile huge libraries on startup, it is also recommended to use a different root directory for each user and project, but that is of course all up to you.

“Set StimLog:” Same as above, but for the stimulus logs produced by RunExperiment’s subsidiary stimulation functions (e.g., RunDriftingGrating). If you wish, you can put these files together with the SpikeGLX files in the same folder and set the same root. The library compiler combines files from the same recording based on the SpikeGLX run name if they are stored as a variable in the file. If not, it defaults to using the experiment ID in the file name. It is therefore recommended to not change the file names yourself, as this might potentially cause incorrect file attributions.

“Set EyeTracking:” Same as above, but for the output of the EyeTracker repository. Use of the EyeTracker is optional.

“Set ProbeLoc:” Default path to select a probe coordinate file. As there is no way to automatically match the output of AP\_histology, SHARP-track, or others to a specific recording, you will have to manually link them by selecting the correct probe coordinate file.

“Set AllenCCF:” Path to the Allen Brain Atlas Common Coordinate Framework data files.

“Set ConfigFile:” Path to the preprocessing configuration file used by Kilosort. A default file for the Neuropixels 1.0 probe is supplied in the /Acquipix/subfunctionsPP/ folder (configFile384\_Npx3B2.m). You can edit the file yourself to match your probe properties, but it is recommended to save this file to a separate directory so you don’t make changes to files the /Acquipix/ directory. This way you can easily update the code with github if a new version is released. If you start locally overwriting the repository files, you might have to create a fork and merge commits manually.

Next, you will want to edit the variables using the “Edit variables:” button. This will make another screen appear, where you can set some metadata that you want to be exported as a json file along with your aggregate data file:

D:\Data\Raw\Neuropixels\Acquipix\Metavars.m		
Rem	syncCh	1
Rem	version	1.0
Rem	dataset	Neuropixels_data
Rem	investigator	Jorrit_Montijn
Rem	project	MontijnNPX2020
Rem	setup	Neuropixels
Rem	stimulus	VisStimAcquipix
Rem	condition	none
Rem	mousetype	BL6
<input type="button" value="New field"/>		
<input type="button" value="Accept"/>		

You can add and remove variables as you wish. There is only one “special” variable here, and that is “syncCh”. syncCh specifies the channel in your National Instruments data file (nidq) that can be used to synchronize stimulus onsets. E.g.: on my setup, we use a photodiode stuck to the upper right corner of the screen where a white rectangle is presented at the beginning of each stimulus presentation. This photodiode then sends a voltage signal to NI channel #1 so we know exactly when the visual stimulus appeared on screen. It is not required to use this additional synchronization, but if



you don't, there will likely be multiple milliseconds of uncorrected lag+jitter between when your stimulus PC logs the stimulus onset and when it actually appears on the screen.

### *Compiling the library and pre-processing the data*

Now that all parameters have been set, you can compile the data library by clicking "Compile data library". The program will now compile a list of all your data files and group them by experiment. The leading file is the SpikeGLX nidq meta file:

Master paths

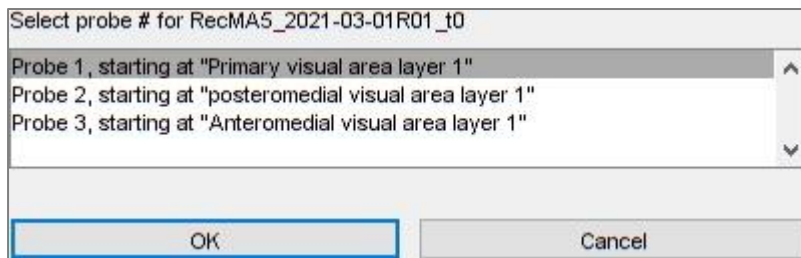
Set Output: D:\Data\Processed\Neuropixels
Set Temp: E:\\_TempData
Set Ephys: D:\Data\Raw\Neuropixels
Set StimLog: D:\Data\Raw\Neuropixels
Set EyeTracking: D:\Data\Raw\Neuropixels
Set ProbeLoc: D:\Data\Raw\Histology
Set AllenCCF: F:\Data\AllenCCF
Set ConfigFile: F:\Code\Acquisition\Acquipix\subfunctionsPP\configFile384\_Npx3B2.m
Edit variables Find data Temp space: 142.0 GB Spike sorter: Kilosort3

Run?	Synthesized	Clustered	Pupil data	Probe coords	Stim files	AP Ephys	LFP Ephys	NI Ephys	File name	Date	Total size
<input type="checkbox"/>	Y	Y	1	2	2	Y	Y	Y	RecMA5_2021-03-01R01_t0	20210618	1.1GB

Cluster spikes
Combine sources
Adjust coords
Export files

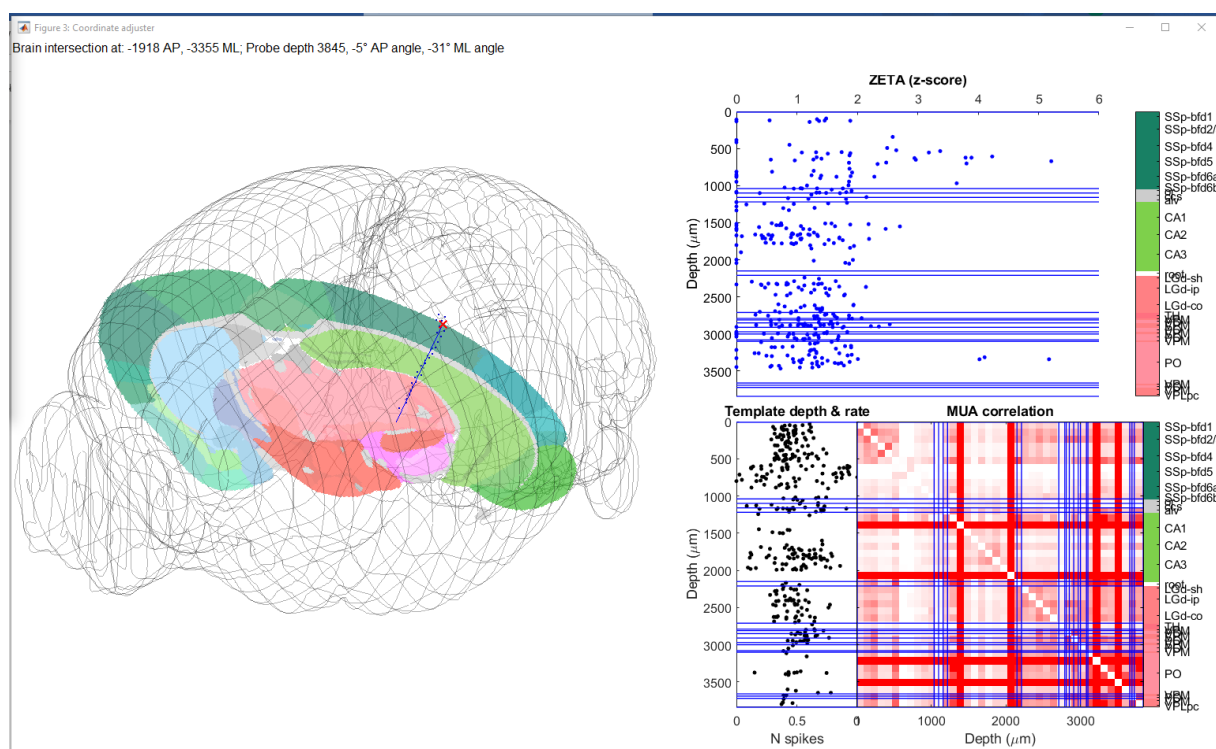
This will show you a list of all experiments, their dates and sizes, and whether certain (preprocessed) files are present or not. Hovering over the different indicators will give you further detailed tooltip information.

You can now select multiple recordings (mark the check box under the "Run?" column) and click "Cluster spikes", after which the program will run Kilosort on all your selected files in serial. When the clustering is finished, you can combine all files of one recording and synchronize the timestamps of the various data into a single synthesized aggregate by clicking "Combine sources". Now you can link the probe coordinate file by clicking on the button under "Probe coords", which will give you another pop-up showing the probe number and area of entry:



### *Adjusting the probe coordinates and exporting the data*

Now that all data has been gathered and pre-processed, you can fine-tune the location of your probe based on your histological data and electrophysiological responses by clicking “Adjust coords”. This will open up a new GUI, which you can also access independently by running `getProbeHistology.m`:



This shows you four types of information. Clockwise starting at the left:

- 1) Similar to the SHARP-track brain browser, it shows you an outline of the mouse brain, the current location of your probe (blue line), the probe's entry point into the brain (red cross), the histological locations you selected for this probe (blue dots), and a transparent slice through the brain showing the location of the ABA-annotated brain areas.
- 2) The plot on the upper right shows the responsiveness of your clusters (blue dots) as a function of depth (y-axis) and ZETA score (x-axis). The higher the score, the more likely your cell is responding to any of your stimuli (see <https://doi.org/10.1101/2020.09.24.311118> for more information on the ZETA-test). The blue lines are area boundaries along the probe's trajectory, as shown on the right.
- 3) The MUA correlation plot shows how correlated the spiking activity is between different channels on the probe. Usually channels that are in the same area will show a higher correlation with each other than with other channels.

4) Finally, the template depth & rate plot shows the normalized firing rate of all clusters (i.e., kilosort templates) as a function of depth. Certain areas may show higher firing rates than others, which can give you another clue of where the area boundaries might lie.

When you are done, you can close the GUI and click “save and exit” to save the adjusted coordinates.

Finally, you can click “export files” in the RecordingProcessor to assign areas to each cluster, save an aggregate recording file, and export a json file with metadata to your output directory.

## Technical descriptions

If you want to make your own stimulation and/or online analysis scripts, you can use the modules present in Acquipix. I've tried to make everything as modular as possible, so it should be easy to plug in experiments of your own design.

### Experiment scripts

#### *Overview*

We will use “RunDriftingGratings.m” as the example here, but other files work the same. All experiment scripts are built so they can be run as a script in the base workspace. The idea here is that your variables will be available in the base workspace even in the case you get an error, forget to save your data, there's a network drive malfunction, etc. I've also built the experiment script so it can be run both independently with default parameters, and as a subfunction of the RunExperiment GUI with supplied inputs.

#### *Description of inputs and structure of an experiment script*

When you press “Start Experiment!” in RunExperiment, it runs the following code block:

```
%run!
assignin('base','sExpMeta',sExpMeta);
assignin('base','sStimParamsSettings',sStimParamsSettings);
assignin('base','sStimPresets',sStimPresets);

evalin('base',strStimType);
```

As mentioned above, you can see that this assigns three variables to the base workspace: sExpMeta, sStimParamsSettings, and sStimPresets. The experiment you wish to run is contained in strStimType; for example runDriftingGratings.

sExpMeta has fixed fields filled by RE\_genGUI:

```
sExpMeta.boolUseSGL = boolUseSGL;
sExpMeta.boolUseNI = boolUseNI;
sExpMeta.dblPupilLightMultiplier = dblPupilLightMultiplier;
sExpMeta.dblSyncLightMultiplier = dblSyncLightMultiplier;
if boolUseSGL
    sExpMeta.strHostAddress = sRE.strHostAddress;
    sExpMeta.hSGL = sRE.hSGL;
    sExpMeta.sParamsSGL = sRE.sParamsSGL;
    sExpMeta.strRunName = sRE.strRunName;
end
if boolUseNI
    sExpMeta.objDaqOut = sRE.objDaqOut;
end
```

It contains some required static information about this recording; whether you wish to use SpikeGLX (boolUseSGL), the National Instruments I/O box (boolUseNI), gain values for the pupil and synchronization LEDs. If SpikeGLX is connected, it supplies the server's IP address (strHostAddress), a handle to the SpikeGLX object (hSGL), a parameter structure supplied by

SpikeGLX (`sParamsSGL`), the name of the current recording (`strRunName`); and if the NI I/O box is used, it supplies a Matlab Daq Interface object `objDaqOut`.

`sStimParamsSettings` is a structure with parameters that are dynamically extracted from the experiment script code itself. If you look in `runDriftingGratings.m` you can see the following block:

```
%% input params
fprintf('Loading settings...\n');
if ~exist('sStimParamsSettings','var') || isempty(sStimParamsSettings)
    %general
    sStimParamsSettings = struct;
    sStimParamsSettings.strStimType = 'SquareGrating';
    sStimParamsSettings.strOutputPath = 'C:\_Data\Exp'; %appends date

    %visual space parameters
    sStimParamsSettings.dblSubjectPosX_cm = 0; %cm; relative to c of scr
    sStimParamsSettings.dblSubjectPosY_cm = -2.5;%cm; rel to c of scr

    (...)
else
    % evaluate and assign pre-defined values to structure
    cellFields = fieldnames(sStimParamsSettings);
    for intField=1:numel(cellFields)
        try
            sStimParamsSettings.(cellFields{intField}) = ...
                eval(sStimParamsSettings.(cellFields{intField}));
        catch
            sStimParamsSettings.(cellFields{intField}) = ...
                sStimParamsSettings.(cellFields{intField});
        end
    end
end
```

The file `RE_getParams.m` reads the experiment script file and loads the field name, value and comment as tooltip information. Any default value you put/change in your experiment file will therefore automatically pop up in the `RunExperiment` GUI. Moreover, if you change values in the GUI, it will supply them in the `sStimParamsSettings` structure and load them in the `else` block. These are usually semi-static variables, like the experiment script's stimulus type (i.e., "SquareGrating"), the location of the mouse relative to the screen, etc. Finally, `sStimParamsSettings.strRecording` is always added by `RunExperiment` to identify the current recording.

`sStimPresets` has dynamic parameter values that you might want to change often and save as a separate pre-set; e.g., the number of repetitions, orientations, stimulus duration, etc. You can change these values easily in the GUI, and on first run, it generates pre-set files in `assertPresets.m`. For example, you can see the following block here:

```
elseif strcmp(strExp, 'RunDriftingGratings')
    for intSet=1:3
        if intSet == 1
            sStimPresets = struct;
            sStimPresets.strExpType = strExp;
            sStimPresets.intNumRepeats = 100;
            sStimPresets.vecOrientations = 0:15:345;
```

```

sStimPresets.vecOrientationNoise = 0;
sStimPresets.dblSecsBlankAtStart = 3;
sStimPresets.dblSecsBlankPre = 0.4000;
sStimPresets.dblSecsStimDur = 1;
sStimPresets.dblSecsBlankPost = 0.1000;
sStimPresets.dblSecsBlankAtEnd = 3;
(...)

```

If you wish to make your own experiment file, you will have to edit `assertPresets` and add your own default values here. Note that you can make it into anything you wish here, with the exception of the `strExpType` field: this is compulsory to differentiate the different experiment scripts. It must match the name of your experiment. For example, because the experiment script is called `RunDriftingGratings.m`, `strExpType` takes the value `'RunDriftingGratings'`.

### *Structure of the experiment script & output file*

You can basically do anything here, as long as you observe the restrictions imposed by the three input variables described above, and the structure of the output file. Looking in `RunDriftingGratings`, we can see that the output file is saved as:

```

save(fullfile(strLogDir, strFilename), 'structEP', 'sParamsSGL');

```

The `sParamsSGL` structure is the same as supplied by `sExpMeta`, so you can use the following:

```

sParamsSGL = sExpMeta.sParamsSGL;

```

The `structEP` structure has four obligatory fields:

`structEP.ActOnSecs` is the actual stimulus onset time in seconds (e.g., the PsychToolBox Screen flip time of the first frame of your stimulus)

`structEP.ActOffSecs` is the actual stimulus offset time in seconds (e.g., the PsychToolBox Screen flip time of the first blanking frame after your stimulus ended)

`structEP.ActOnNI` is the actual stimulus onset in National Instruments (nidq) time stamp

`structEP.ActOffNI` is the actual stimulus offset in National Instruments (nidq) time stamp

You can obtain the National Instruments time stamp using:

```

hSGL = sExpMeta.hSGL;
intStreamNI = -1;
dblSampFreqNI = GetSampleRate(hSGL, intStreamNI);
dblTimeStampNI = GetScanCount(hSGL, intStreamNI)/dblSampFreqNI;

```

That said, it is of course wise to save more information than that; in the case of drifting gratings this would be the orientation during each trial, the spatial frequency, etc. To ensure I always have all meta information, I also include the `sStimParams` structure before saving:

```

structEP.sStimParams = sStimParams;
structEP.sStimObject = sStimObject;
structEP.sStimTypeList = sStimTypeList;
save(fullfile(strLogDir, strFilename), 'structEP', 'sParamsSGL');

```

But of course you can decide differently for yourself. A minimal experiment script could therefore look something like this:

```
% evaluate and assign pre-defined values to structure
if ~strcmpi(sStimParamsSettings.strStimType, 'StupidExperiment')
    sStimParamsSettings = struct;
    sStimParamsSettings.strStimType = 'StupidExperiment';
    sStimParamsSettings.strOutputPath = 'C:\Data\';
    sStimParamsSettings.strTempObjectPath = 'C:\Temp\';
else
    cellFields = fieldnames(sStimParamsSettings);
    for intField=1:numel(cellFields)
        try
            sStimParamsSettings.(cellFields{intField}) = ...
                eval(sStimParamsSettings.(cellFields{intField}));
        catch
            sStimParamsSettings.(cellFields{intField}) = ...
                sStimParamsSettings.(cellFields{intField});
        end
    end
end

% set output locations for logs
strRecording = sStimParamsSettings.strRecording;
strOutputPath = sStimParamsSettings.strOutputPath;
strTempPath = sStimParamsSettings.strTempObjectPath;
strThisFilePath = mfilename('fullpath');
[strFilename, strLogDir, strTempDir, strTexDir] = ...
    RE_assertPaths(strOutputPath, strRecording, strTempPath, strThisFilePath);

%get SGL data
hSGL = sExpMeta.hSGL;
strRunName = sExpMeta.strRunName;
sParamsSGL = sExpMeta.sParamsSGL;
intStreamNI = -1;
dblSampFreqNI = GetSampleRate(hSGL, intStreamNI);

%run trials
structEP = struct;
hStartTic = tic;
for intThisTrial = 1:sStimPresets.intNumRepeats
    %"inter-trial" interval
    pause(0.5);
    structEP.ActOnSecs(intThisTrial) = toc(hStartTic);
    structEP.ActOnNI(intThisTrial) = GetScanCount(hSGL, ...
        intStreamNI)/dblSampFreqNI;

    %"stimulus" period
    pause(1);
    structEP.ActOffSecs(intThisTrial) = toc(hStartTic);
    structEP.ActOffNI(intThisTrial) = GetScanCount(hSGL, ...
        intStreamNI)/dblSampFreqNI;
end

%save data
save(fullfile(strLogDir, strFilename), 'structEP', 'sParamsSGL');
```

### *Integrating your experiment file into Acquipix*

If you've followed the descriptions above, the only thing you need to do now is edit the following files:

- `RE_defaultValues.m` creates the default values, including which files are experiment scripts. You want to add your new experiment to the `sRE.cellStimSets` field.
- Add the default parameters and values for an experiment preset to `assertPresets.m`.
- `RE_evaluateStimPresets.m` calculates how long your experiment will take based on your preset parameters and ensures there are no incompatible values

### **Online analysis functions**

#### *Overview*

Acquipix comes with three real-time data stream analysis GUIs: `runOnlineOT` for orientation tuning with drifting gratings, `runOnlineNM` for natural movie response analysis, and `runOnlineRF` for receptive field mapping. We will use “`runOnlineOT`” as the example.

To be continued...



## **Troubleshooting**

Question (“actually, it’s more of a comment”): *It doesn’t work*

Answer: Restart your PC

Q: *I downloaded everything, but it says files are missing*

A: Double check you have added all folders to the path in Matlab, you have the required Matlab toolboxes installed (Curve Fitting, Parallel Computing), and you’re using a supported matlab version: R2019b is tested and works; anything earlier than R2016b will fail for sure; other versions might work. You may also need additional toolboxes, such as Image Processing and Acquisition to perform eye-tracking. If it’s still not working after you’ve tried the above, google the filename and reinstall its source repository. If it still fails, create a report here: <https://github.com/JorritMontijn/Acquipix/issues>.

Q: *I cannot compile Kilosort’s GPU code*

A: First try installing VSC2015 (make sure you have the Visual Studio **Community** version) and make sure you install the C++ compiler. Then **restart your PC** and try again. If it doesn’t work: uninstall and try VSC2013. If neither of them work, look for help here: <https://github.com/MouseLand/Kilosort/>

Q: *I cannot run any GPU code in matlab (i.e., gpuArray() fails)*

A: Make sure that you have the correct CUDA drivers installed for your GPU. Note that if you’re using anything other than a (modern) Nvidia GPU, you cannot run CUDA.

Q: *I found a bug*

A: Great! Or at least, it’s great that you found it, not that it’s there. If you’ve fixed it, you can make a pull request, otherwise you can create a bug report here: <https://github.com/JorritMontijn/Acquipix/issues>. Please copy/paste the matlab error message and as much detail as you can about what you were doing when it happened. If I cannot recreate the issue, I probably won’t be able to fix it.