E_ROT: a brief guide

The following is to be used as a reference on how to use the program created to run experiments and tasks, and as a guideline to the contents of its files. The program is written in Matlab using the toolbox Psychtoolbox.

Each programmed task has its own function and GUI, plus a GUI to run experiments that can combine different blocks and tasks. Each task will have an output, saved automatically in a folder relative to the Subject and the Block that has been run.

This program is meant to be used with a NETStation to record EEG signals. The EEG recording starts automatically at the beginning of a new block and stops at the end of the block, but can also be started and stopped at the experimenter's discretion directly from the GUI. Each function is programmed to automatically send an event on target activation.

The program is intended to run on a Mac and cannot run on anything else with the current settings without some modifications, but it would be possible to adapt it fairly easily. (see GetCursorData)

INDEX

Before starting

This is a set of instructions on what has to be done to be able to use this program correctly.

- The folder *E_ROT* should be present in its entirety and set as the current path. (for info on which files are necessary and what exactly this folder contains, see: E_ROT: all the files)

(the following should be done only on the first time this program is used on a new computer. If the program isn't working correctly, check if these steps have been performed properly)

- Matlab has to be installed.

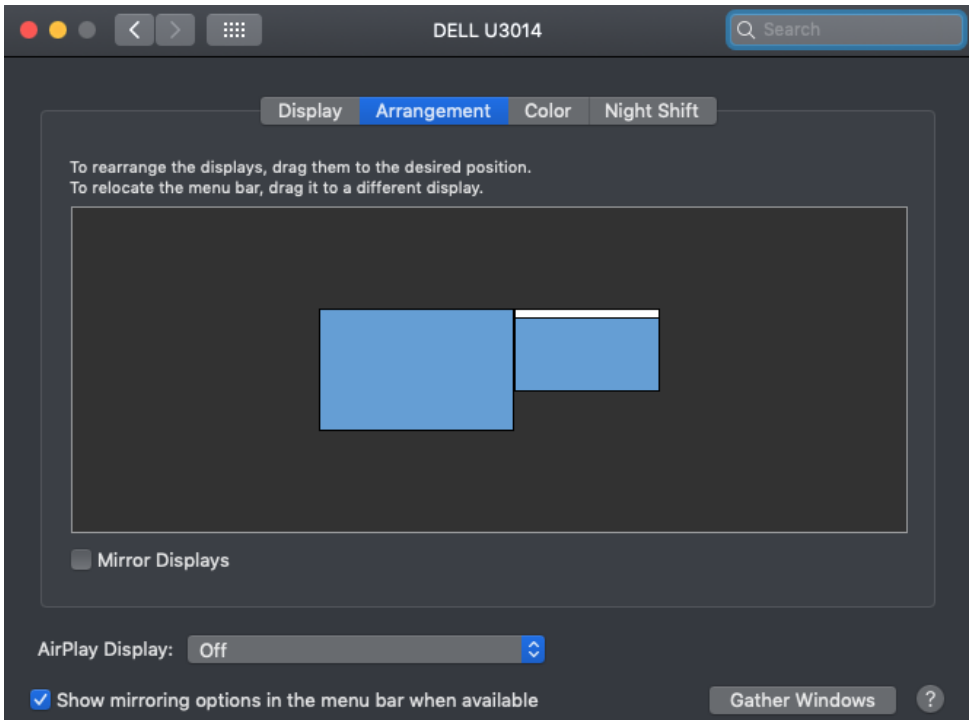
- The Matlab toolbox PsychToolBox has to be downloaded and installed (link)

- The standalone *GetCursorData* has to be installed. To do this, go into the folder *GetCursorData/for_redistribution* and run *MyAppInstaller_mcr.app*. Use the default settings and complete the installation. (you can check if this has been done by looking into the Application directory and checking if the folder *GetCursorData* exists). This will install both the standalone and the Matlab Runtime necessary for this to work. The computer should have an internet connection while performing this step, if the computer doesn't have Runtime on it already.

- The connection between the computer the program runs on (client) and the host computer connected to the NETStation (host) has to be established and set properly. To do this, follow these steps:

    1. Connect the two computers using an ethernet cable
    2. Go to *Finder > Network* on both computers and look for and connect to the other one (*Connect as…*)
    3. Go to *Apple > System Preferences > Sharing* and enable all the services (except internet)
    4. Go to *Apple > System Preferences > Network* and find the ethernet port you're using (if you're not sure which one it is, you can find out easily by disconnecting it and connecting it again and looking at which one becomes available. If you disconnect the cable you may need to perform step 2 again).
    5. Set *Configure IPv4* to *Manually*
    6. Set the IP addresses: the host computer address HAS to be consistent with the one indicated on the acquisition settings from the GUI (see EEG Acquisition). The default one is 192.168.21.35. The client IP can be a variation of this, for example 192.168.21.36.
    7. Set the router to the specified address (in this case, 192.168.21.1) on BOTH computers (the router address has to be the same)

8. If you want to acquire EEG data, run the NETStation Acquisition software and enable the use of port 55513. This is the default port indicated in the acquisition settings, and can be modified (see EEG Acquisition). NB: If the software isn't active or the use of the port is disabled, the program will crash if it's asked to start the acquisition.
9. The second display has to be arranged so it's on the left of the main screen and so the two displays' top edge are aligned (see figure)



Set Up

The correct setup to run these experiments is one Mac computer (client) connected to two screens, and a second computer (host) connected both to the client and to the NETStation. The experimenter launches and controls the experiment from the client's main screen while the subject sits in front of the second screen, not being able to see the main screen. The task gets automatically called fullscreen on the second screen.

E_ROT: all the files

Here follows a list of the folders and files necessary to have all the scripts run correctly, with a brief description of their intended use. All of these have to be in the current folder when using these functions and their hierarchy or names should not be modified.

Scripts: expGUI.fig and expGUI.m

These are the scripts and figure relative to the GUI in which you can define an experiment and run it. This is what will be mainly used to run experiments, after the basic protocols for each block have been defined (for more info, see expGUI).

Folder: _TASK

This folder contains the different tasks, where each task is identified by its own folder. Every task has:
- its main function and its GUI (eg: *MemTask > MemTaskFUN.m, MemTaskGUI.m* and *MemTaskGUI.fig)*: these GUIs can be used on their own and are to be used to define blocks and sets relative to their specific task. (For more info on this folder, go to tasks)
- folder protocols: this contains the blocks that have been defined for each task, where each block is identified by its own folder. (For more info on this folder, go to protocols)
- folder *saving(nameTask)Sets*: this contains the scripts to save single sets relative to the specific task. (Baseline_newSet.m contains specific info on the parameters). These can be used to save sets programmatically instead of using the GUI.
- folder *colorSequences*: this folder (only for Vis and Mem) contain a series of .mat files, each defining a specific color wheel to be used in the mnemonic tasks. The color wheels have to be separately defined and saved, following the syntax shown in the script *colSeq_mem/ colSeq_vis.m* in the same folder.

Folder: _timelines

This folder contains the experiments that have been already defined, in the form of a series of .mat files. A timeline is a series of blocks in a specific order, where each block refers to the ones defined in the folder */protocols* of its task.

Folder: GetCursorData

This folder contains the files relative to the standalone deployed to get the cursor's coordinates from the screen (used in the MotorTask). Its main function is GetCursorData.m. If this function is modified, the project has to be rebuilt and reinstalled in order for the changes to take place.

More info

Protocols

The folder *protocols* for each task contains the settings relative to different blocks. Each block is identified as a folder containing a number of *.mat* files where the settings are saved in a specific format (see: in each task's folder, in the folder *savingSets* the script *Baseline_(nameTask)set* for specific info on the parameters for the different tasks). You can see the settings of a set by loading its .mat file directly into Matlab.
The folder *protocols* must stay in the same folder of its reference task. New blocks can be added by creating a new folder inside *protocols* and saving new sets inside of it. All the sets relative to a block must be in the same folder called *[nameBlock]*.
To create a new set, you can do so in two ways:
-from the GUI, by selecting "*Single Set(Load/Save*)", filling out the desired settings, including the name of the block and the number of the set, and pressing "*Save current settings*". The folder the protocol name specified will be created automatically, you will be again prompted to decide where you want to save the set and how it should be named.
- programmatically, by using either the script *Baseline_newSet* or the function *saveNewSet* (*Baseline_newSet* has a more detailed explanation of the parameters, while *saveNewSet* is more straight-forward) (TIP: If you want to save a lot of sets all at once, use the function in a custom loop code).

IMPORTANT – ABOUT FILE NAMES
The set files MUST be named with the following format:
nameBlock_set#.mat
Where # is the number of the set in the sequence it is going to be presented in the block. This is so it's easy to understand the order of the sets in the protocol. If there is no "set1" in the protocol, there's going to be an error.
eg. The first set of the new block called "block1" will be inside the folder *protocols/block1* and named: "*block1_set1.mat*" and so on.

EEG Acquisition

The EEG acquisition is set to start automatically when launching the first set of a new block (it will not start if you skip the first set). A prompt will appear to inform you the acquisition is starting, at which point you can decide to let it start, cancel the acquisition or edit the parameters (IP address and port). On block completion, the acquisition is set to stop automatically, but again you will be prompted to decide on this. In between sets, you can activate or deactivate the acquisition from the GUI, by pressing the acquisition button. The acquisition can still be controlled independently from its own software (NetStation Acquisition), just remember that the GUI will not have info on what you do outside of it and could therefore operate diffently.

GUIs

You can access each of these by calling them directly from the command window.

expGUI

This is the GUI relative to the whole experiment and the one that is meant to be used to run complete experiments once the protocols have been defined. You can create a timeline of blocks by selecting "edit timeline", selecting a task and its relative block and adding them to the timeline. The tasks available correspond to the folders inside *_TASKS*, the blocks available for each task correspond to the ones defined inside each *protocol* folder for its relative task (for more info, see protocols). You can also save the timeline you defined or load a previously defined one. From here you cannot define the settings for each block separately, this is meant to be used after the blocks have already been programmed. You shouldn't add the same block twice in the same timeline, as in that case the program will overwrite the previous data with the same name; if you need to use the same settings twice, you should create two separate blocks with the same characteristics or construct a single block with multiple sets with the same settings.
When you select or construct a timeline, you can run the experiment from this GUI by pressing START. This will automatically create a folder in the current path named after the *Subject Name* you specified, where all the outputs from the tasks will be saved (e.g. for the MotTask, the output is the behavioral data).
This GUI calls directly on the functions defined for each task. The base for each task is one single set, which means that upon pressing START, the GUI will call the appropriate function with the settings for the current set. Upon set completion, the program switches automatically to the next set of the block, so the user will only have to press START again to launch the next set. The set can be interrupted by pressing PAUSE, at which the program will stop running the current set and not switch to the next one. The output will be created nonetheless, but in this case, it will be saved with the tag "INTERRUPTED" appended to its name.

When you define a timeline or load one, you will be shown how many sets there are in the current block and which set you are currently running; this will be updated thorough the experiment progression. When all the blocks are completed, you will be informed by a message. You can "complete" the experiment manually before having actually run all the defined blocks by pressing STOP.

Tasks

Each one of these tasks (with the exception of the faux task RestingState) has its own GUI from which you can define, load and save new sets and blocks. Info on the parameters for each task can be found in the script *Baseline_(task)Set* in the folder *saving(task)Sets* for each task.
(TIP: A new task can be theoretically added to the folder and be used in the expGUI, provided it has all the same folders and files with the correct syntax relative to the new behavior.)

In every task's main function, the first lines of script have easy debugging controls in case it's necessary to slightly tweak the script. Specifically, there are:

- Control to use only part of the screen during the task instead of full screen mode
- Control not to hide the mouse cursor
- Name of the default event to send to the NETStation
- Manual settings for the display dimensions, in case the figure isn't being printed properly (e.g. The circles of the targets are being displayed as ovals, that means the conversion cm > pixels isn't being calculated correctly, in which case it may be necessary to manually input the display measurements).

Motor Task

This is a reaching task. At the beginning of each set, the subject has to position the cursor in the center point to make the task progress; the center point then flashes three times, and then the actual tasks begins: a number of targets (specified by the user) appear on screen in a set position and get activated in a certain sequence; the subject is prompted to try to reach them with a linear movement and upon reaching the activated target, this will briefly change color.
The output of this function is a *.mat* file with two vectors, *"targetxdistance"* and *"targetydistance"* containing the coordinates of the center of the targets in the order in which they appeared (in cm), and a matrix *"data"* containing the cursor coordinates (in mm with the axis centered on the center of the screen), their timestamp (in ms) and the status of the targets (either 0 if none is active or the number of the one that's currently active).
(specifically: data = [timestamp; x; y; targetStatus]).
The cursor's coordinates are acquired by the standalone *GetCursorData*, deployed automatically by the function (see GetCursorData).
The function sends an event to the NETStation on the appearance of the target. Currently the event is 'MOT1', this can be edited from the MotorTaskFUN, in one of the first lines of the script.

The base for this task is a single set containing one sequence with all the movements relative to the set.
For more info on the parameters for this task, see its *Baseline_motSet* (in the folder *savingMotSets).*

Mem Task

This is mnemonic task. A fixed number of targets (currently 8) appear on screen in a set position and get activated in a certain sequence: the subject is prompted to try to remember the sequence in which they appeared. After each sequence, there is a holding time and then a color wheel appears; after the same holding time, the new sequence begins. The output of this function is a *.mat* file (empty at this point, tbd).
The function sends an event to the NETStation on the appearance of the target. Currently the event is 'MEM1', this can be edited from the *MemTaskFUN*, in one of the first lines of the script.
The base for this task is a single set containing a number of sequences and the same color wheel after each sequence. For more info on the parameters for this task, see its *Baseline_motSet* (in the folder *savingMotSets).*

Vis Task

Similar to MemTask, with the current differences:

- The number of targets is currently 12
- There is no holding time between the sequence and the color wheel
- After each sequence, the experimenter is prompted to decide if the sequence should be repeated another time or not (from the command window)
- The color wheel displayed changes each time between sequences

The function sends an event to the NETStation on the appearance of the target. Currently the event is 'VIS1', this can be edited from the VisTaskFUN, in one of the first lines of the script.

Resting State

This function doesn't have a GUI, and therefore can only be called directly from the command window or from the expGUI, by adding a *RestingState* block. This function simply shows a fixation cross for a set number of seconds and the only input the function has it's the time it runs for. The task already contains a fixed set of protocols; to create a new one you would need to do it programmatically as of now, by saving a .mat file containing the parameter *restTime*.

GetCursorData - basic working principle

The standalone *GetCursorData* is programmed to get coordinates from the screen every 2ms, but the frequency ultimately also depends on the device that's used as a cursor. This standalone gets automatically called and closed by the function, and has an independent timeout of 600s, which means that at the longest it will record data for 10 minutes (if you want to change this, you will have to change the .m function, rebuild and reinstall the standalone). If Matlab gets closed unexpectedly or the function doesn't execute properly, GetCursorData may not get closed automatically. In this case, make sure to close it manually before trying to launch the task again.

It wouldn't be too difficult to make the modifications to have a version compatible with another OS: the main difference would be on how the standalone operates.
First of all, the standalone communicates with the function using a mapped file that they both use to share data. This mapped file is currently saved in the "Application" folder, since it has the same path in every Mac. This path should be therefore modified, both in the GetCursorData.m function and inside each task's main function.
Secondly, currently GetCursorData has been compiled as a Mac executable, as a .app file. This should be recompiled as an executable compatible with the OS you want to use.