Laser and galvanometer control software manual

This software was written in Matlab by Lucas Pinto at the Princeton Neuroscience Institute. Please contact him at lpinto@princeton.edu for further inquiries. Below is quick reference guide for using the software.

If you use this software, please cite “Pinto et al., 2019, Task-dependent changes in the large-scale dynamics and necessity of cortical regions, Neuron”

Overview

This software performs hardware control and contains a GUI interface written to drive a system that scans a laser using 2D galvanometers, used for optogenetic inactivations in the dorsal cortex. Its main design goals were to allow easy integration with behavior setups being run by other machines, to ensure consistent targeting across experimental sessions, and to allow for experimental automation. Its essential associated hardware components are a DAQ card with 4 analog output lines (e.g. NIDAQ PICe-6343), 2D galvanometers with associate generators, a high-performance laser with digital and analog inputs (e.g. Coherent OBIS), and a camera for monitoring and image registration (e.g. pointGrey grasshopper3). Further details about hardware can be found in Pinto et al., 2019 or upon request to the authors.

Function reference

Below I list crucial functions with brief descriptions of their purpose. Please refer to in-function headers and comments for further details.

1. Class objects (~/classes)

There are 3 class objects that must be edited before running laserCtrlGUI(). See .m files for details

- animalList() defines list of experimental animals with associated experiment parameters.
- lsrCtrlParams() lists many parameters required for experiment control and data saving.
- LaserRigParamaters() lists hardware parameters specific to the rig
- ViRMEnStateCodes() is specific to the experiments in Pinto et al 2019 and included as an example for behavior experiment control.

2. Calibration (~/calibration)

galvoCal() and powerCal() provide automated routines to calibrate command voltage-galvanometer angle mapping, and command voltage-laser power mapping, respectively. They create .mat calibration files that must be generated before running the experiments. To calibrate the galvos place a white, opaque sheet of paper at the focal plane of the camera. To calibrate power use a power meter, ideally placed at the level corresponding to the animal's head. The calibration routine asks for user input of the read power, on the command line. These functions are accessible through button presses in the calibration tab of the GUI.

3. Experiment logic (~/experiments)

3.1. Experiment control

Whenever the user turns on the laser, control is taken over by the laserLoop() function. This top-level script initializes the data output and decides on the behavior of the laser and galvanometers by selecting the appropriate laserLoopFn*() function, according to GUI selections. These functions specify what happens within each data output iteration of the software (which by default runs at 400 Hz). For instance, when the system is being manually controlled by the user through the GUI, laserLoop() calls laserLoopManual() 400 times/second (with enforced timing interval).

laserLoopPreSet() provides an example of a loop in which the galvos visit predetermined locations with deterministic timing (e.g. turn the laser on for 1.5 s every 5 s).

laserLoopFnTrig_example() is an example of a triggered function, which turns the laser on and off and moves the galvos following commands from a separate computer (which in this case runs the behavioral experiment). In this particular example, the other computer is running virtual reality behavior using ViRMEn. Both computers run on a loop, ViRMEn at 120 Hz (the projector refresh rate) and Laser at 400 Hz. At every iteration, ViRMEn decides whether the laser should be on or off, and if so, in which location. It sends the 8-bit code to reflect that. On the other side, the laser PC reads the ViRMEn lines at every iteration as well, and compares to its previous state. If the state (location, laser on or off etc) has changed, then it goes to the new location, otherwise it does nothing.

In this case the 8 bits are required because in addition to location, ViRMEn also sends 'trial states' (i.e. ITI, reward, trial setup, experiment ended etc). This primarily happens because a laser log file gets updated at the end of each trial, when the ViRMEn computer is busy rendering the next trial. Because the laser computer needs to run at 400 Hz, you can only save when its services aren't required, so it needs to know when is safe to save.

Also note that even though ViRMEn controls the laser PC, the parameters for the experiment are all set on the laser PC. So right before the experiment starts, the laser PC sends all experiment parameters to the ViRMEn PC via TCP/IP (it is bidirectional, so it goes: 1. laser PC > ViRMEn: location list; 2. ViRMEn > laser PC: OK, received!, 3. laser PC > ViRMEn: laser on probability; 4. ViRMEn PC > laser PC: OK, received! and so forth). This is implemented in sendExptParameters().

The user is of course encouraged to write their own functions that meet their experimental needs.

3.2. Data logging

laserlogger() provides an example of a function that logs laser/galvo data and metadata, in the case of the behavioral experiment outlined above. This function is called only in inter-trial intervals, when timing does not need to be enforced with precision, as explained above. This function should be easily modifiable for user-specific needs.

4. Inactivation grids (~/grid)

Inactivation grids are lists of inactivation (galvo) locations in stereotaxic coordinates with respect to bregma. They are saved as .mat files in this folder and must follow the format below.

Single-location grids (example, fullGrid.mat). If each inactivation location is a unique point, the list of locations is specified as a matrix where rows are locations and columns are [ML AP] coordinates in mm.

Multi-location grids (example, fullGridBilateral.mat). If each inactivation location is comprised of multiple near-simultaneous inactivation locations, the list of locations is specified as a cell array where each cell entry corresponds to near-simultaneous points. Within this cell, locations are specified as matrices where rows are locations and columns are [ML AP] coordinates in mm. Note that while the order of visited locations is not enforced, it is recommended that they follow the order corresponding to the least amount of travel distance. In our setup, the galvos can alternate between locations at 200 Hz, with a measured travel time of ~ 200 μs per 1 mm.

In both cases, the variable should be called *grid*, and an additional *P_on* variable should be saved in the same file. *P_on* specifies the overall 'laser on' probability for that grid. For example, if P_on = 0.3 and the grid contains 3 locations, each location will be visited in (approximately) 10% of instances.

GUI reference (laserCtrlGUI)

General controls (bottom)

'MouseID' dropdown menu: used to select animals from a pre-determined list (animalList() class object). Selecting an animal will automatically load its reference image and headplate outline drawing if available, as well any experiment control parameters associated with the animal. An automatic mouseID_dateTime file name will also be generated. Note that experiment parameters and file name can be overridden by editing the appropriate entries in the GUI.

'File name' edit box: displays current file name and allows for typing custom names.

'Refresh' button: refreshes filename with the current dateTime.

'Set dir' button: opens a file browser to change the current root directory where data are saved. Default directory should be specified in the laserRigParameters() class object.

'RESET' button: closes GUI and reopens a fresh session.

'QUIT' button: closes GUI and all hardware communication.

Camera feedback and image registration panel

Camera display: axis used to display the feed from the camera (camera name must be specified in laserRigParameters())

'Cam ON' button: turns camera feed on/off. This is a toggle button, meaning that camera remains on until button is pushed again.

'Grab frame' button (and following user input prompt cascade): grabs the current frame and saves it following automatic naming conventions, and also turns camera off. Grabbing frame will prompt user input boxes. The first one will ask if user wants to save the grabbed frame as reference, i.e. the reference image for that animal, to which future grabs will be registered. If the answer is *yes*, the frame will be saved in the animals' corresponding folder as animalName_refIm. A new prompt will ask whether the user wants to set the reference pixel, i.e. the [0 0] point in the grid coordinate system (normally bregma). If so, the user will mark the point using mouse-controlled crosshairs (matlab ginput() function). This coordinate is saved with the reference image. If the answer to the "save as reference?" prompt is *no*, the GUI will ask whether the user wants to register the current frame grab to the reference image. If the answer to that new prompt is *yes*, the image registration function will be called and the current

image will be registered. A new box will pop up with the results of the registration, i.e. whether or not image is within user-defined tolerances of the reference (specified in the lsrCtrlParams() class object). Note that currently if the registration is beyond the pre-defined tolerance, the GUI will not respond to button presses to start experiments, and the registration procedure must be repeated until registration passes the quality test. If the user wants to suppress this behavior, edit line 284 in the laserCtrlGUI() m file to suppress the lsr.okFlag output of the registerImage() function, comment line 655, and switch the default okFlag value in in the lsrCtrlParams() to true.

'Register' button: registers current image in memory (i.e. latest frame grab) to the reference, triggering the same prompt cascade as described above.

'Set zero' button: sets the reference pixel, i.e. the [0 0] point in the grid coordinate system (normally bregma). If pressed, the user will mark the point using mouse-controlled crosshairs (matlab ginput() function). This coordinate is saved with the reference image.

'Set grid' button: allows users to draw arbitrary inactivation grids by performing mouse clicks on the image. This is of limited experimental functionality but may help with troubleshooting.

'Draw plate' button: allows user to draw a headplate outline on top of the reference image, using the matlab roipoly() function. This works by creating anchor points with single mouse clicks, and dragging the mouse to create lines until the next desired anchor point. The shape is finalized by closing it onto an existing anchor and double-clicking. If present, this drawn outline is always plotted on top of the camera feed, and is intended to guide mouse placement, thus helping with image registration. Note, however, that it is not used for the registration per se.

'IR LED' button: if an (infra-red) LED is connected to the DAQ card, this button will turn it on. Can be useful for animal monitoring or behavioral sync.

'Green LED' button: if a (green) LED is connected to the DAQ card, this button will turn it on. A green illumination source is recommended for image registration, as it provides the best vascular contrast.

Laser and galvo control panel

'Pulse freq. (Hz)' edit box: displays or allows editing the frequency of the laser square waves in Hz.

'Duration (s)' edit box: displays or allows editing the duration of the laser pulse in seconds. Typically only relevant for manual laser pulses.

‘Duty cycle’ edit box: displays or allows editing the duration of the duty cycle in the interval [0 1]. For example, a duty cycle of 0.8 will cause the square waves to be high (laser on) during 80% of the 1/pulse frequency period.

‘Power (mW)’ edit box: displays or allows editing the laser power in mW.

‘Ramp down dur. (s)’ edit box: displays or allows editing the duration of the linear power ramp down at the end of the laser pulse, in seconds.

‘P (on)’ edit box: overall probability that the laser will be on in behavioral experiments (i.e. in any location), interval [0 1].

‘Trial epochs’ dropdown menu: select from pre-specified list of inactivation epochs in behavioral experiments (specified in the lsrCtrlParams() class object). This information is passed to the behavior computer and not implemented anywhere in this repository.

‘Trial draw’ dropdown menu: select from pre-specified list of trial draw methods in behavioral experiments (specified in the lsrCtrlParams() class object). This information is passed to the behavior computer and not implemented anywhere in this repository.

‘Ramp down method’ dropdown menu: select from pre-specified list of ramp down methods in behavioral experiments (specified in the lsrCtrlParams() class object). This is relevant for VR experiments where trial epoch is defined in maze space, not time. This information is passed to the behavior computer and not implemented anywhere in this repository.

‘Vary laser power’ dropdown menu: check to generate variable laser powers during experiments. This needs to be implemented individually in the user-specific laserLoopFn*.

‘Trigger source’ radio button: Switch between ‘manual' and ‘external’. If set to manual, laser will only work upon GUI button presses and will ignore external triggers. Vice-versa if external is selected.

‘ML’ edit box: displays or allows editing the mediolateral location of the laser beam with respect to the current reference point, in mm.

‘AP’ edit box: displays or allows editing the anteroposterior location of the laser beam with respect to the current reference point, in mm.

‘Cursor’ button: galvos move to location specified by user by mouse button clicks on image display.

‘Single pulse’ button: generate single laser pulse with the parameters specified above.

‘Manual ON’ button: turn on the laser with the parameters specified above, until button is pressed again.

‘Behavior PC’ button: send information and deliver control to behavior PC. For now this results in calling the laserLoopFnTrig_example(), to be modified according to user needs.

Status tab

‘Status’ box: displays current status of the software (e.g. idle, registering image, etc).

‘Console’ box: displays and updates history of commands.

‘Enter notes here’ box: allows input of arbitrary experimental notes. To use it left click first, then start typing.

‘Save console’ button: saves console as text file.

‘Save notes’ button: saves notes as text file.

Calibration tab

‘Calibrate galvos’ button: runs automated galvo calibration routine. See calibration function reference.

‘Sweep galvos’ button: takes your galvos for a test joy ride.

‘Calibrate power’ button: runs semi-automated power calibration routine. See 'Calibration' function reference.

Presets tab

‘Ephys’ button: runs example preset experiment. See ‘Experiment logic' function reference