User Manual

Python-based, open-source software to integrate gaze-contingent behavior experiment for non-human primates with neurophysiology

Developed in Shadmehr Lab in Department of Biomedical Engineering at Johns Hopkins University

Version 0.1

Eye tracker by:

VPixx Technologies
Vision Science Solutions

Version History

Version updated to	Date	Author	Reason
0.1	2023/11/21	J. S. Pi	First draft

Table of Contents

List of Figures

Introduction

Accurate and precise implementation of a behavioral task is critical to conducting good science. Also, it needs to be easily maintained and expanded to design future experiments by any member of the lab. To these ends, we developed a Python-based open-source software to run gaze-contingent experiment and to integrate a host of external devices for a full neurophysiology setup.

We have run extensive testing to ensure sub-ms precision real-time gaze-contingent behavior paradigm. Screen delay from the time of command to display is 10 ± 3 ms and continually verified using a photodiode, which is integrated into the system. The system one clock, and the alignment signal is sent to the external devices for an offline conversion to the system clock. The sampling frequency of the data is constant without variation. The software provides an easy-to-use manual calibration tool that is proven to work with non-human primates and that can calibrate each eye separately and let the user choose which eye to track. It also includes an automatic calibration tool to fine tune the parameters, once a rough calibration is done. It is designed to run in conjunction with neurophysiology by integrating random digital signal to align the data between behavior and ephys times, and it allows an in-app control of OpenEphys software. It controls other peripherals like food pumps, essential for running animal experiment. Lastly, it is Python-based and documented, to allow fast and easy modification and extension.

Support

For questions, you can contact Jay Pi (jay.s.314159@gmail.com) or Reza Shadmehr (shadmehr@jhu.edu).

Installation

Hardware

* One may not need all the parts shown in the layout below, but we include them here to show the full capability of the system.

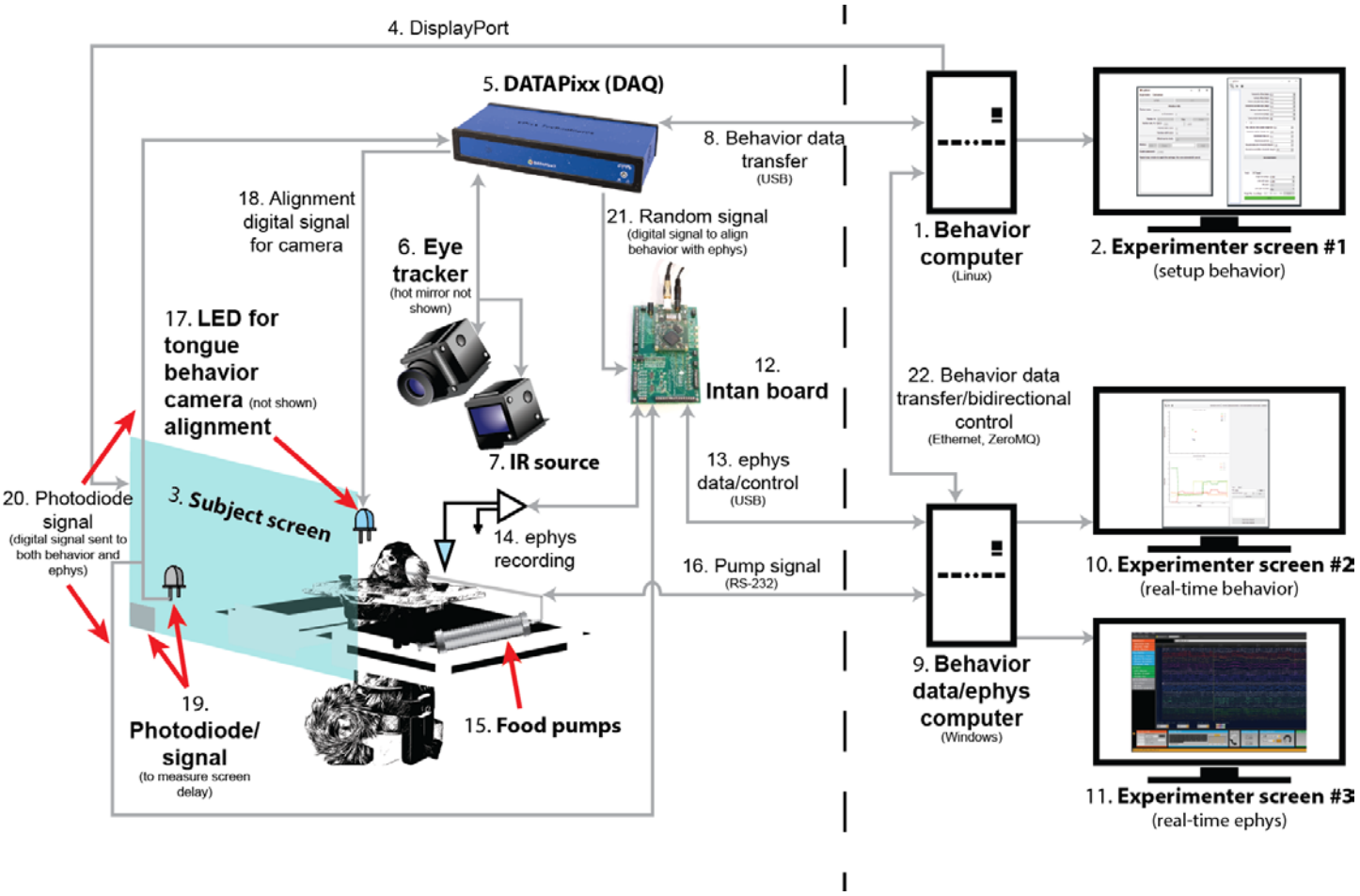
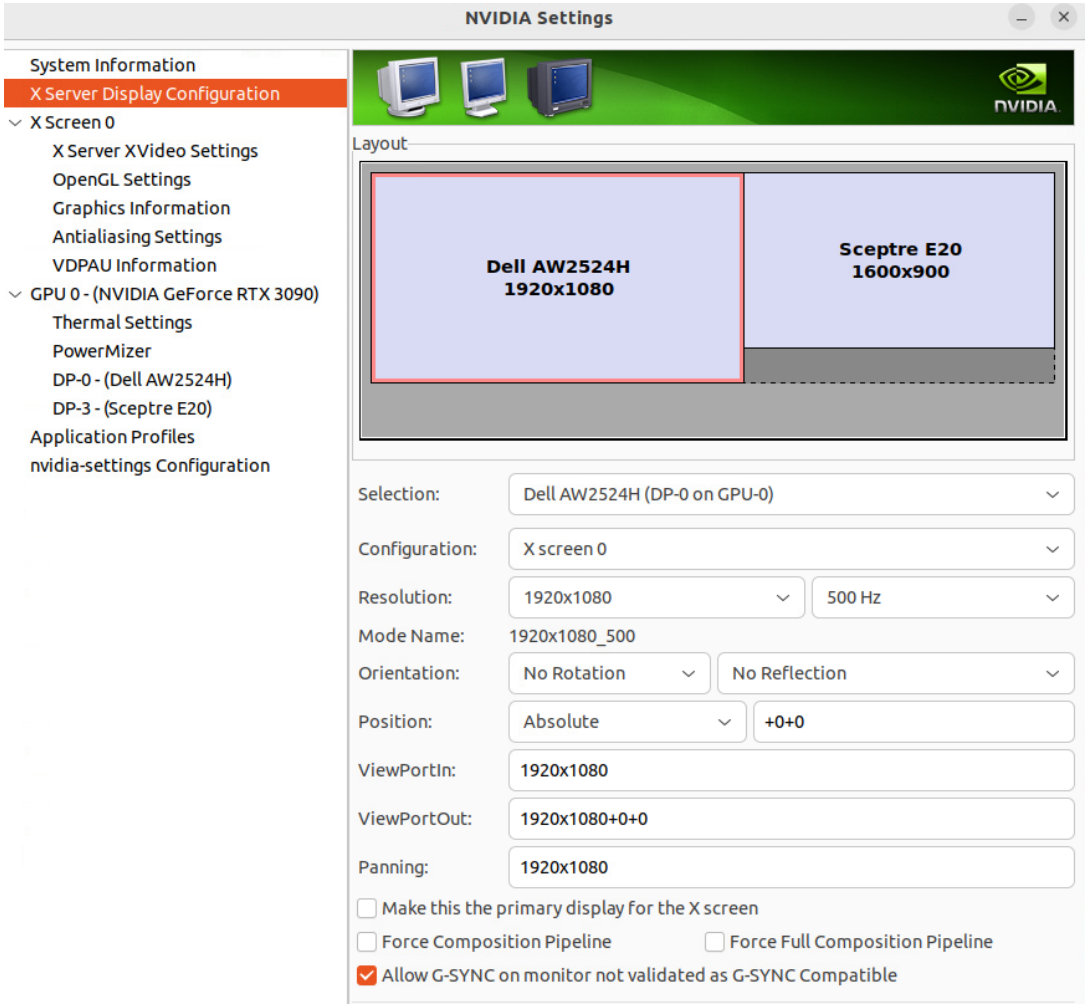


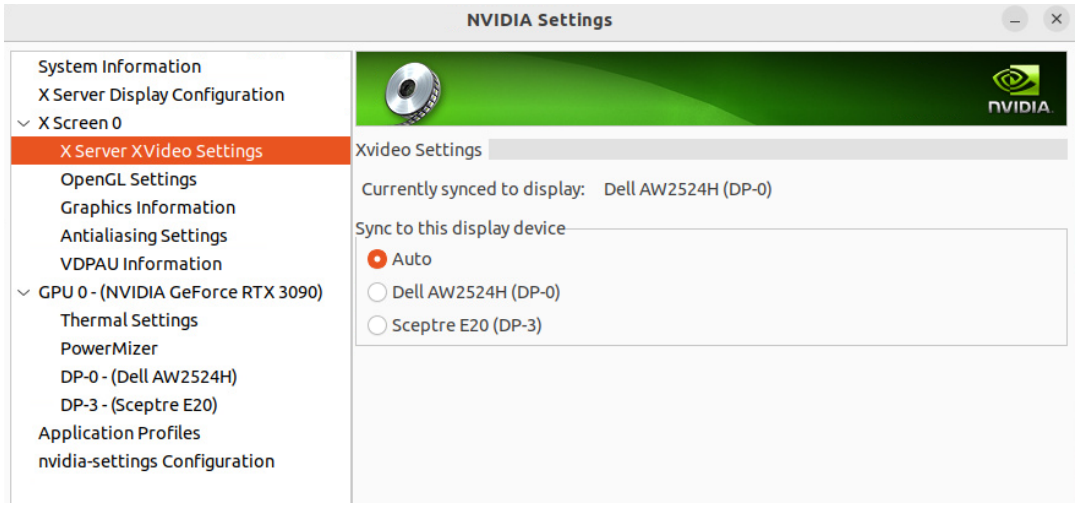
Figure 1. Hardware layout

1. **Behavior computer**: Linux (Ubuntu 22.04.2 LTS) was used, as it was found to be more stable during long behavior task sessions than Windows.
    - Motherboard: ASUS PRIME Z390-A
    - Processor: 12th Gen. Intel Core i9-12900KF x 24
    - Memory: 64 GB
    - Graphics: NVIDIA GeForce RTX 3090

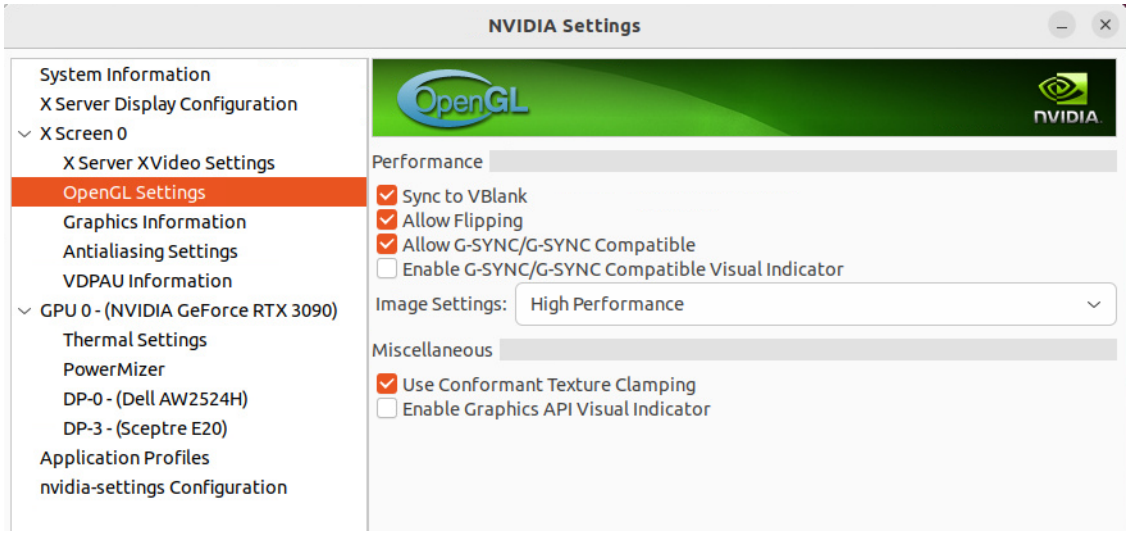
- NVIDIA setting needs to be changed to enable G-SYNC and the correct monitor refresh rate



- This setting can remain as “Auto”



- OpenGL setting to “Sync to VBlank”, “Allow Flipping”, “Allow G-SYNC/G-SYNC Compatible; set to “High Performance”



- Hard drive: 1 TB

2. **Experimenter screen #1**: to open up app. modules, conduct calibration/refinement, and change experimental parameters. Any monitor would work here.
3. **Subject screen:** ALIENWARE 500HZ GAMING MONITOR – AW2524H
    - One of the most critical equipment for running gaze-dependent saccadic task
    - Note that refresh rate is not the same as what we measure as the display latency, which is the time between the command to display the target in code and when it is actually shown on the monitor; increasing refresh rate nonlinearly decreases display latency.
    - Given the system spec. (**1. Behavior computer**), this latency was around 10 $\pm$ 3 ms (mean, std).
    - There is a way minimize the variance (negligible at around 0.1 ms) at the cost of 1 more ms in mean delay, if this suits a particular experiment better; see Target visualization.
4. **DisplayPort**: cable between (**1. behavior computer**) and (**3. subject screen**).
    - Use a quality cable, which is especially important when the computer and screens are placed far apart in different rooms; we found a cable also has a significant effect on the display latency (~3 ms).
    - Note that this setup is different from the VPixx recommendation to send the display signal via (**5. DATAPixx**); this was done to support 500 Hz refresh rate of the monitor
5. **DATAPixx3**: hub for all digital I/O and receives tracker data
6. **Eye tracker**: TRACKPixx3
    - Consult VPixx for proper eye tracker setup
7. **IR source**
    - Consult VPixx for proper eye tracker setup
8. **Behavior data transfer via USB**: common USB 2.0 cable
9. **Behavior data/ephys computer:** (Windows 10 Enterprise)
    - Motherboard: ASUS PRIME Z390-A
    - Processor: 9th Gen. Intel Core i7-9700K
    - Memory: 64 GB
    - Graphics: NVIDIA GeForce GTX 1060
        - Specs. is not as critical as for (**1. behavior computer**)
    - Runs Open Ephys software to collect ephys data
    - Real time behavior data is displayed here

- Behavior data is saved here along with the ephys data

10. **Experimenter screen #2**: non-critical
    - Usually used for display real-time behavior data
11. **Experimenter screen #3**: non-critical
    - Usually used for displaying ephys data
12. **Intan board:** RHD USB Interface Board
    - Receives data from ephys probes
    - Receives digital signal from DATAPixx to align the behavior and ephys data
    - Receives photodiode signal
        - The signal is also sent to DATAPixx as well, so this is more of an insurance than necessity
13. **Ephys data/control via USB**: common USB cable
14. **Ephys recording**: for more information on ephys recording, refer to our lab's published papers on marmoset
15. **Food pumps**: NE-500 Programmable OEM Syringe Pump
    - To feed animal to encourage performing task
    - Controlled by the (**9. behavior data/ephys computer**)
        - Serial communication is relatively slow, and we as experimenters also often manually pump the foods, so to minimize the latency in the task, the signal is sent from this computer
16. **Pump signal:** to control (**15. food pumps**) with serial communication via RS-232.
17. **LED for tongue behavior camera alignment**: common LED
    - Controlled by (**5. DATAPixx**)
    - In the included tasks, the signal is the same as that of (**19. photodiode signal**)
18. Alignment digital signal for camera: digital signal from (**5. DATAPixx**) to control (**17. LED for tongue behavior camera alignment**)
19. **Photodiode/signal**: to observe when the target is actually displayed on the screen
    - The digital signal is displayed on the screen, which is captured by the photodiode attached to it
    - It is sent to both (**5. DATAPixx**) and (**12. Intan board**)
20. Photodiode signal: digital photodiode signal sent to both (**5. DATAPixx**) and (**12. Intan board**)
21. Random signal: a random digital signal is sent from (**5. DATAPixx**) to (**12. Intan board**) to align behavior data with ephys
22. Behavior data transfer/bidirectional control: with ZeroMQ via Ethernet

Software

For both (**1. behavior computer**) and (**9. behavior data/ephys computer**):

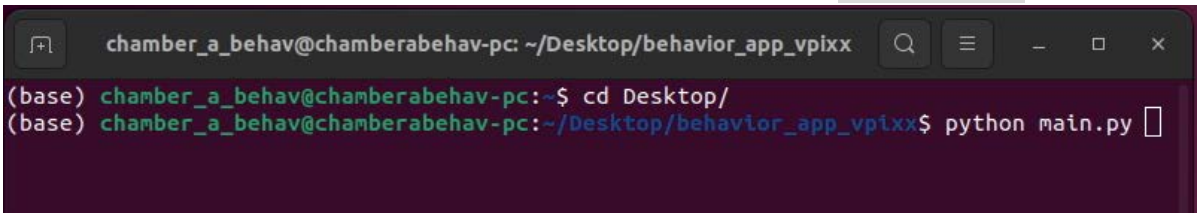
1. Install Anaconda3-2020.11-Windows-x86_64
    - It is assumed and highly recommended the following packages are installed in the base environment and the computer used solely for running the experiment.
    - Packages that do not specify versions will list tested versions, but other versions will most likely work
2. In Anaconda prompt, `conda install python=3.8.5`
3. `conda install -c anaconda pyzmq` (tested version: 19.0.2)
4. `pip install psychopy==2021.1.0`
    - Possible errors and solutions:
        i. `ERROR: Cannot uninstall 'ruamel-yaml'. It is a distutils installed project and thus we cannot accurately determine which files belong to it which would lead to only a partial uninstall.`
            - In Windows PowerShell, `rd -r "C:\.\anaconda3\Lib\site-packages\ruamel*"` (be sure to use your parant path)
        ii. `C:\Users\ChamberB_Behavior\anaconda3\lib\site-packages\pyglet\media\codecs\wmf.py:838: UserWarning: [WinError -2147417850] Cannot change thread mode after it is set warnings.warn(str(err))` when calling `from psychopy import core, visual, event` for example
            - In Anaconda prompt, `pip install pyglet==1.4.10`
5. In Anaconda prompt, `conda install -c anaconda pyqtgraph` (tested version: 0.11.0)
6. `pip install simpleaudio` (tested version: 1.0.4)
7. `conda install -c anaconda h5py` (tested version: 2.10.0)
8. Install VPixx python library
    - Download and install software package from VPixx after creating account
    - In Anaconda prompt, `pip install -U "(parent path)/pypixxlib.tar.gz"` ; use the most recent 'tar' file
        i. For ex., `pip install -U "C:\Program Files\VPixx Technologies\Software Tools\pypixxlib\pypixxlib-3.11.11156.tar.gz"`
    - All development was made in Spyder IDE; for using VPixx library in this environment to test/debug, "pip install" the library within Spyder command line as well
9. Download the code repository

Running the application

Opening the application

(1. Behavior computer)

- On Linux terminal, go to directory where the repository is located and python main.py



- The main control panel should show up:

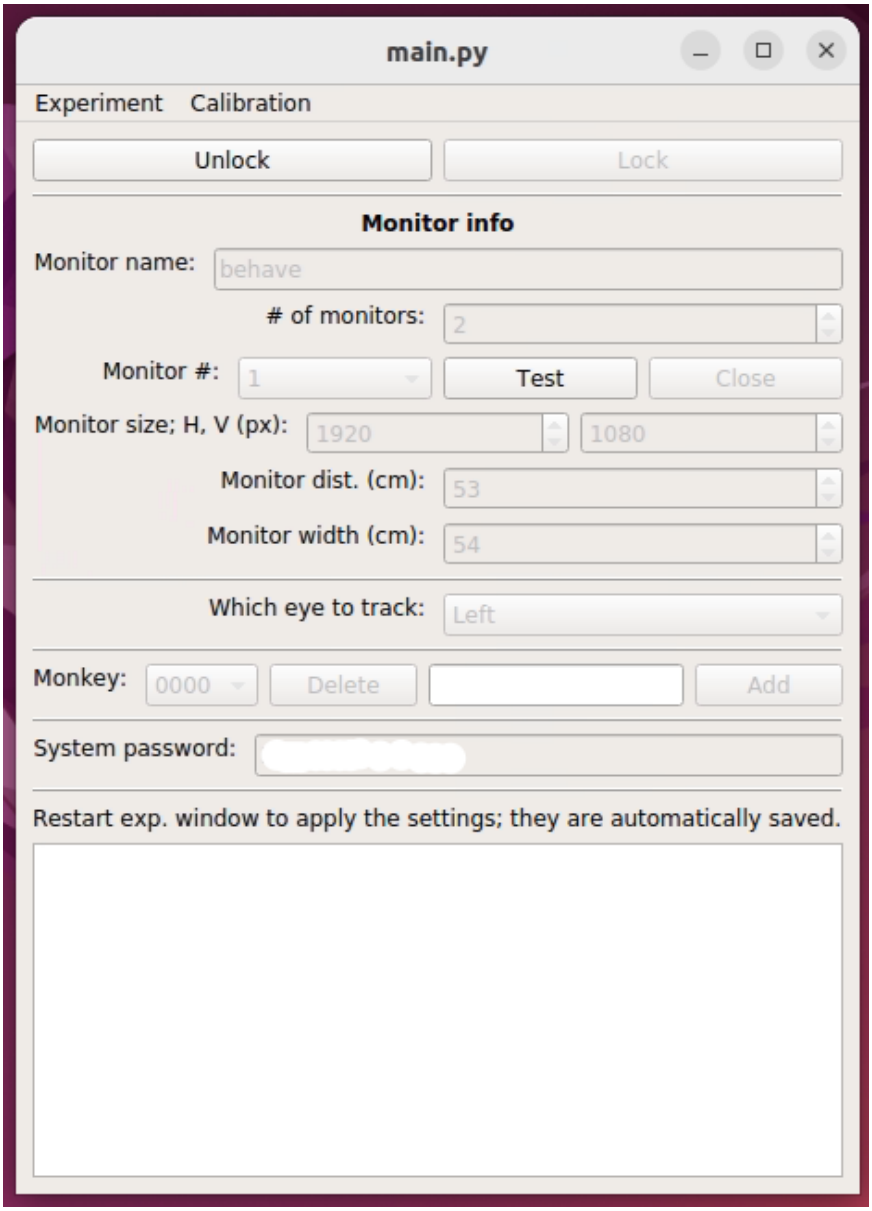
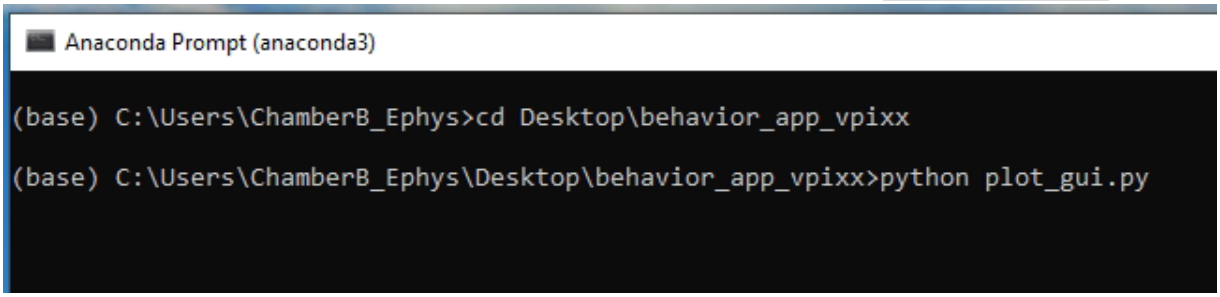


Figure 2. Main control panel

(**9. Behavior data/ephys computer**)

- On Anaconda Prompt, go to directory where the repository is located and python plot_gui.py



- GUI that displays real-time behavior and sets pump setting should show up:

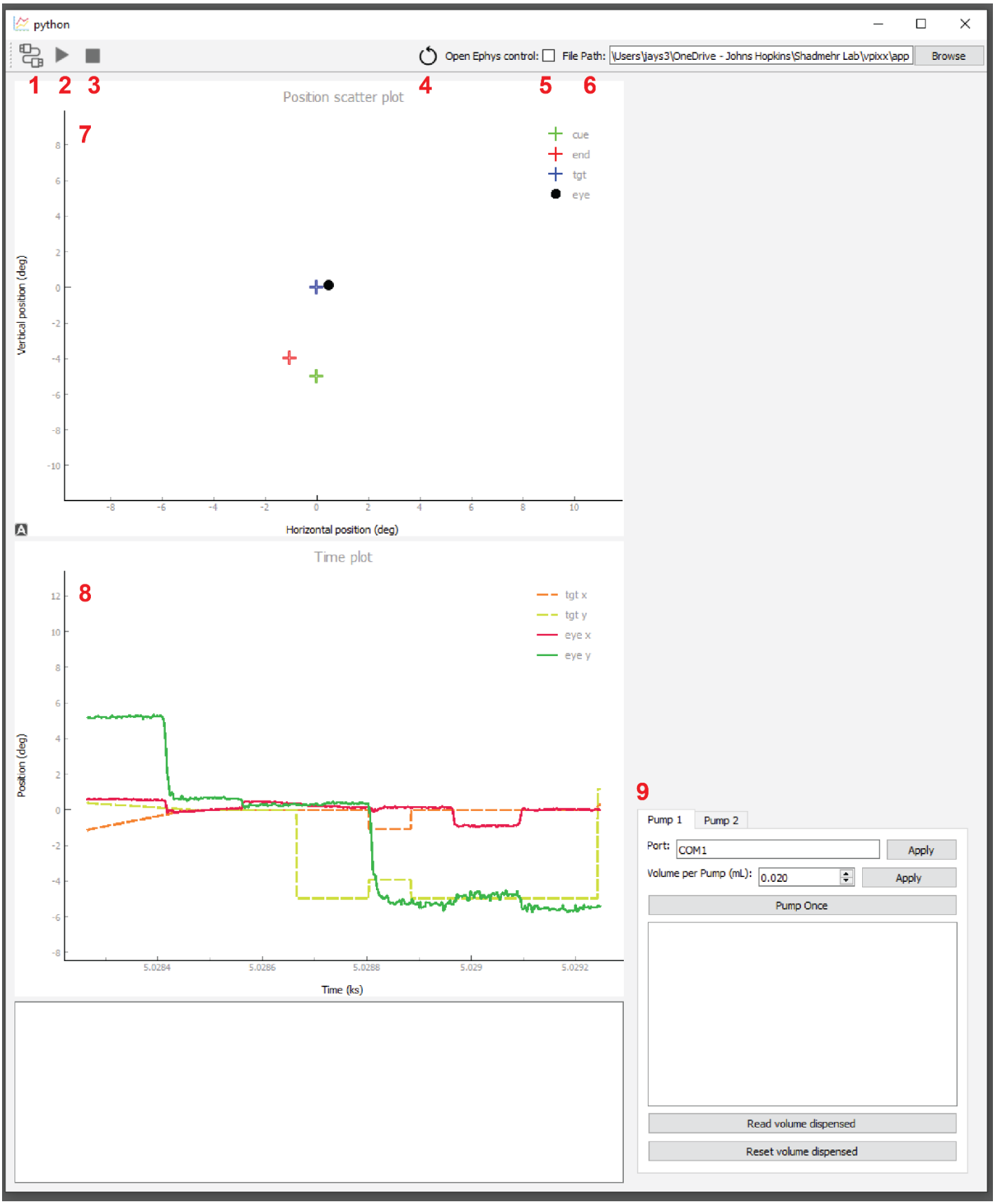
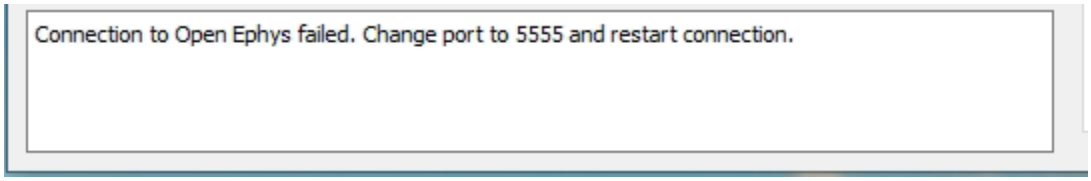


Figure 3. Real-time plot GUI

1. Click this button to receive command from (**1. Behavior computer**). This is to receive real-time behavior data, pump commands, and etc. from the other computer.

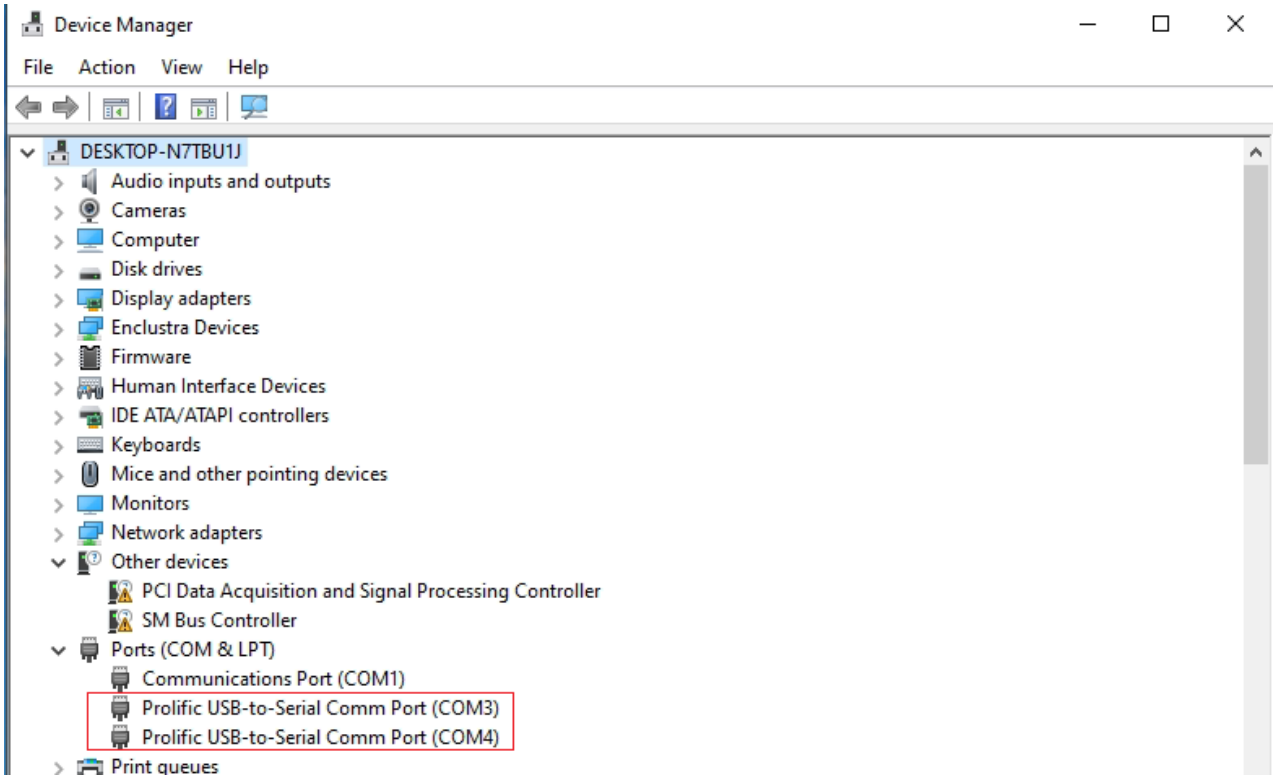
2 – 3. Start/stop the experiment; not applicable for Calibration or Refinement module
   - App. on (**1. Behavior computer**) should be enabled to receive the commands from this computer
   - Upon stopping, behavior data are saved in the 'data' folder of the repo. directory
4. Tries to restart the connection to Open Ephys; usually after changing the port number on Open Ephys to the correct number, after getting the following error:



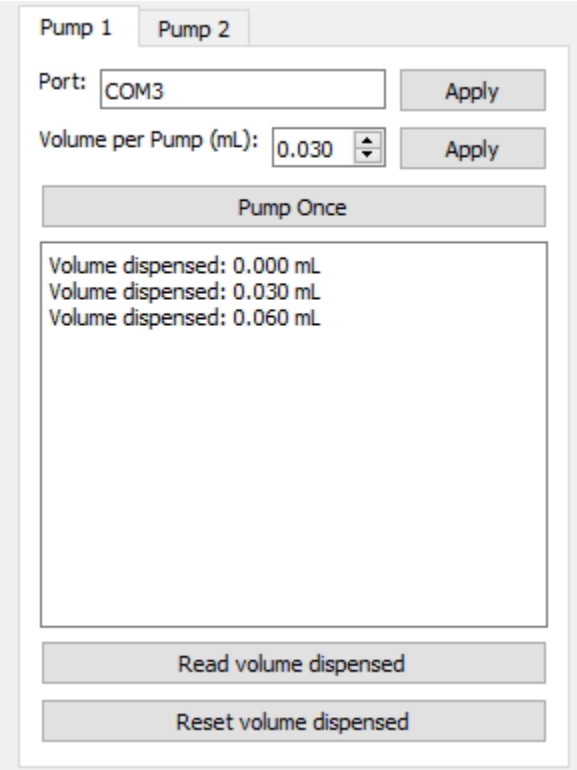
5. Check to start the recording of Open Ephys at the same time as the behavior experiment.
   - Behavior and ephys data are saved in the same directory (6. File Path)
     - This is in addition to the behavior data saved in the 'data' folder of the repo. directory
6. File path to save the behavior and ephys data if the option to control the Open Ephys is enabled (5. Checkbox)
7. Scatter plot of target positions and current eye position
   - 'tgt' – current target position
   - 'eye' – current eye position
   - Others – target positions of the current trial
8. Time plot of current target and eye positions
9. Pump setting; see Setting up pump parameters

Setting up pump parameters

- The default is two pumps, but more pumps can be added (see **Internals** section on pump).
    - Click different tab to change the setting of the desired pump
- Set port address by finding the pump on *Device Manager*; hit *Apply* to set



- *Volume per Pump*: sets the rate of pump per command; hit *Apply* to set
    - Hitting *Apply* freezes GUI to apply the change; this is normal and would not affect the ongoing experiment
    - There are many other settings for the pump that can be set according to the manufacturer, and those additional functions can be added by the user if he or she wishes them.
- *Pump Once*: sends one command signal to pump
    - This action opens up another thread to accomplish its task, so it will not freeze the GUI
- *Read volume dispensed*: clicking this shows how much pump has been dispensing since last hitting *Reset volume dispensed*

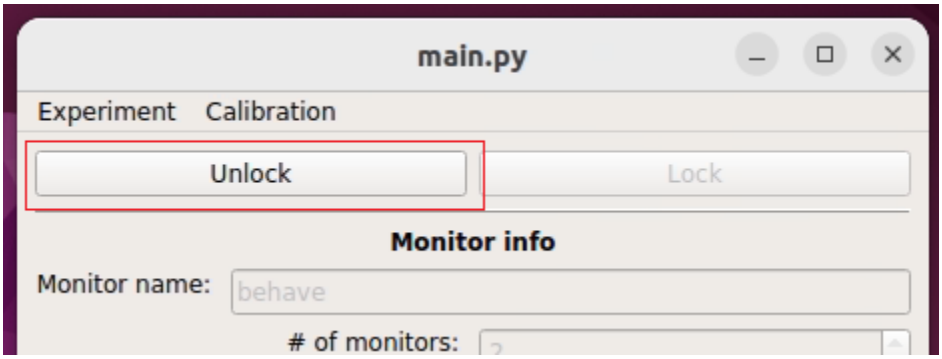


- Clicking this temporarily freezes the GUI but does not affect the ongoing experiment

- *Reset volume dispensed*: clicking this resets the amount of volume dispensed tracked by the pump

Setting up *main* parameters

- After clicking *Unlock* button, fill in parameters.

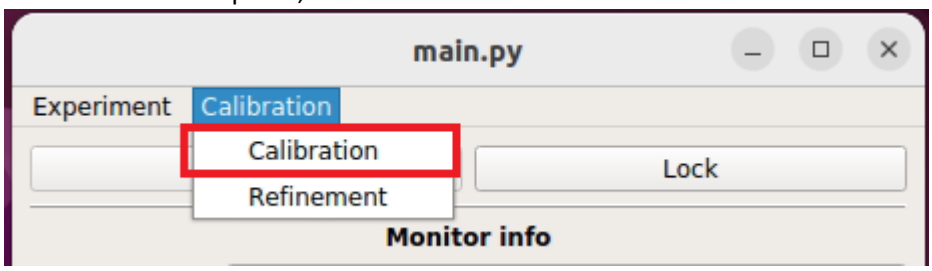


- *# of monitors*: number of monitors connected to (**1. Behavior computer**).
  - At least more than 2 monitors including (**2. Experimenter screen #1**) and (**3. Subject screen**)
- *Monitor #*: to select which monitor to display subject window; click *Test* to see where it will be displayed
- *Monitor size*: number of pixels in (**3. Subject screen**)
- *Monitor dist.*: distance from the eye to the (**3. Subject screen**)
- *Monitor width:* horizontal width of viewable area of (**3. Subject screen**) in cm
- *Which eye to track*: to select which eye to calibrate and track for the experiment
  - User can separately save the calibration of each eye
- *Monkey*: to keep the calibration of multiple subjects
  - After typing in the subject name, click *Add* to save the subject name.
- *System password*: type in the system password of (**1. Behavior computer**).

- Because Linux saves all the ethernet communication messages, which are used to send messages to and from (**9. Behavior data/ephys computer**), eventually the internal storage will run out and crash the program.
- Having the password in here will enable the app. to automatically input *sudo* command to delete the log files.
- If unsure, this feature can be disabled, and the log files can be manually deleted periodically.

Calibration

- In the main control panel, click *Calibration*.

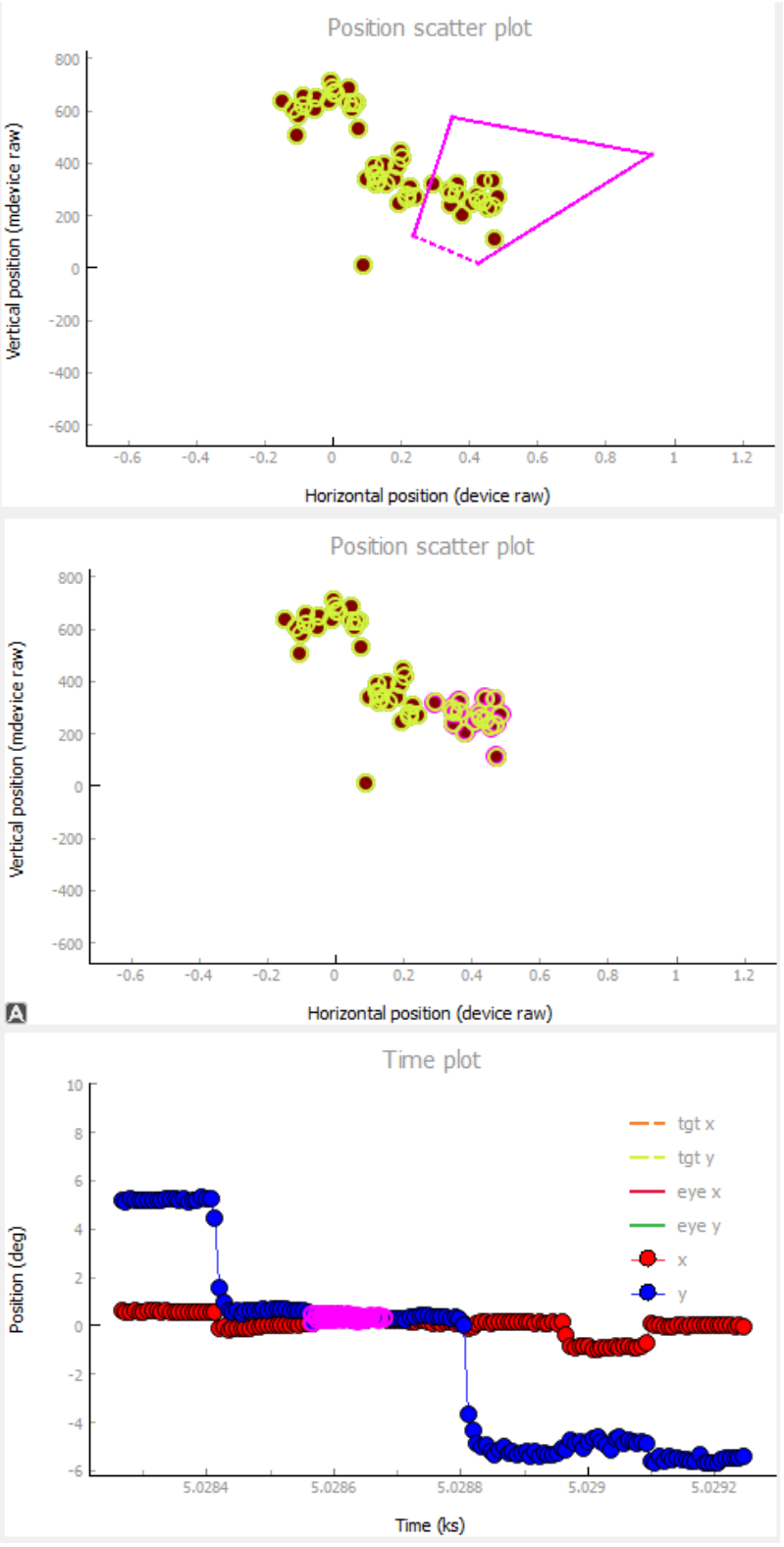


- Subject window will show up on (**3. Subject screen)** and calibration control panel will show up on (**2. Experiment screen #1**).
- Open real-time behavior GUI on (**9. Behavior data/ephys computer**); set it to receive data from the other computer

Figure 4. Calibration GUI

1. Top plot: scatter plot of raw x and y positions as measured by the tracker
2. Bottom plot: time plot of raw x and y positions
3. Run (shortcut **R**): display the target as specified by the parameters
4. Stop (shortcut **CTRL+S**): stop displaying targets
5. Shows current status of calibration; it changes the color to green when there is a saved calibration
6. Calibrate (shortcut **CTRL+C**): when calibration points for more than 5 targets are selected, the button will be enabled; clicking it will show the root mean square error of the calibration points

7. Select (shortcut **S**): user can draw the region of interest (ROI) and select the calibration points for a specified target by clicking on the plots (either top or bottom) and then clicking this button: doing so will highlight the selected points in both the top and bottom plots

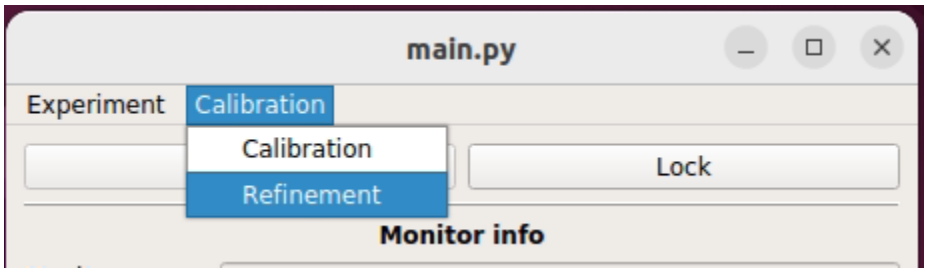


8. Deselect (shortcut **D**): user can deselect the points that were previously selected
9. Clear (shortcut **C**): user can clear the ROI on either plots
10. Delete selected: user can remove selected points
    - Useful when collecting calibration points for more than 2 targets; if the points for one target occlude the sight of points of another target, user can select those points and remove them

11. Delete All: if user wants to delete all points, selected or unselected; basically a resetting button
12. Delete: if user wants to delete all the points for a specified target
13. Target #: shows points of which target is active
    - User can manually change after having presenting multiple targets
    - It also automatically changes as a target is subsequently presented
14. Clear calibration: removes the current calibration
15. Save: after successfully calibrating, the user can click this button save the calibration
16. Target positions: specifies where to present the targets, in eye degrees
    - Check mark specifies which target to present in one run
        - For non-human primate, usually only one target is checked
        - For humans, all targets can be generally checked, and the calibration can be finished in one run
17. Distance from target: targets are generally presented on 9-point grid, in eye degrees
    - Once position for target # 5 is set, clicking *Auto-populate* will automatically populate the positions for all other 8 targets given this parameter
    - If this value is z and the positions for target #5 is (0,0), then position for target # 1 is (+z, +z), #2 is (0, +z)…; the diagonal targets are not technically z distance away from the center target, but it does not practically matter in this case
18. Inter-target interval: in second, how long to wait between presenting targets when more than 1 target is selected
    - Not usually used for non-human primates
19. Cal. target duration: in second, how long to present a target
20. Pursuit target: when this is checked, a pursuit target from the center target (#5) position to the current target is displayed for a specified distance (eye deg) and duration (second), before presenting the target
    - Useful for non-human primates, to grab the attention of the animals
    - If the animals do pursue, the pursuit will be reflected in the bottom plot, and the subsequent jumped eye positions will be that of the desired target
21. Pump: button to activate either first or second pump.
    - Radiobutton is used to indicate which pump to activate with the keyboard shortcut (**P**)
    - Generally, a pump is used to awake the animal, and you wait till animal finished eating, and then present the target

Refinement

- In the main control panel, click *Refinement*.



- Because this module is to refine the existing calibration, an initial calibration of sufficient quality needs to be done first
- Subject window will show up on (**3. Subject screen**) and refinement control panel will show up on (**2. Experiment screen #1**).
- Open real-time behavior GUI on (**9. Behavior data/ephys computer**); set it to receive data from the other computer

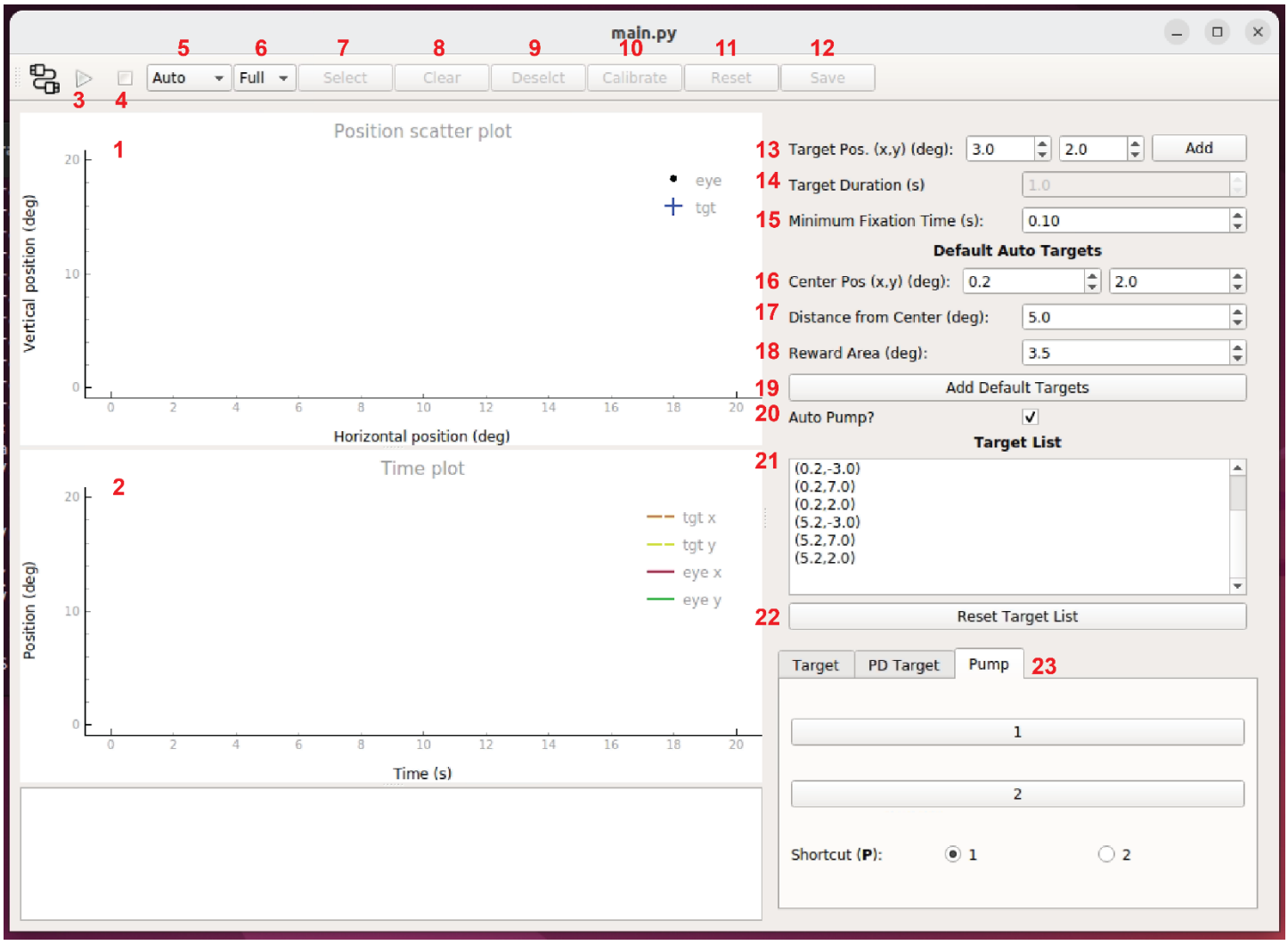
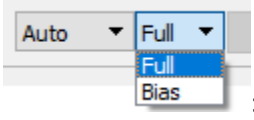


Figure 5. Refinement GUI

1 – 4: see Calibration section

5. Mode:
    a. Auto: sequentially present the target in 20. Target List; subject needs to make saccades to them, fixating for 15. Minium Fixation Time at each target. The fixation point needs to be within a half of 18. Reward Area from the target. After all targets are completed, an automatic calibration will be performed using the fixation points for corresponding targets, showing the user root mean square error of the calibration points
    b. Manual: only one target as specified by 13. Target Position and 14. Target Duration will be shown; then the user selects the calibration points for that target, like in Calibration
    c. Test: same sequence as Auto mode, but there is no re-calibration at the end; this is to see how refined the current calibration is
6. 

:
    a. Full: after running Auto mode, full calibration is performed
    b. Bias: after running Auto or Manual mode, only bias correction is performed

7 – 12. See Calibration section

13. Target Position: the position of the Manual mode target, in eye degrees
    - Clicking Add will add a target of this position to the 20. Target List for either Auto or Test mode
14. Target Duration: how long to display a target, in second; only applicable for Manual mode
15. Minimum Fixation Time: in second; see 5. Mode. a

16 – 17, 19. As in Calibration, adding 9-point grid targets around the center target; the target list is shown in 21. Target List

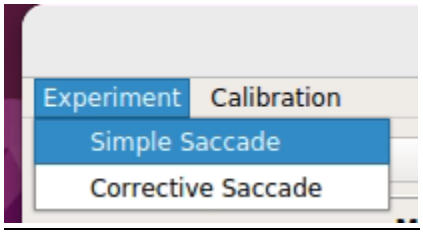
18. Reward Area: in eye degrees; see 5. Mode. a

22. Reset Target List: clicking this removes all the targets in 21. Target List

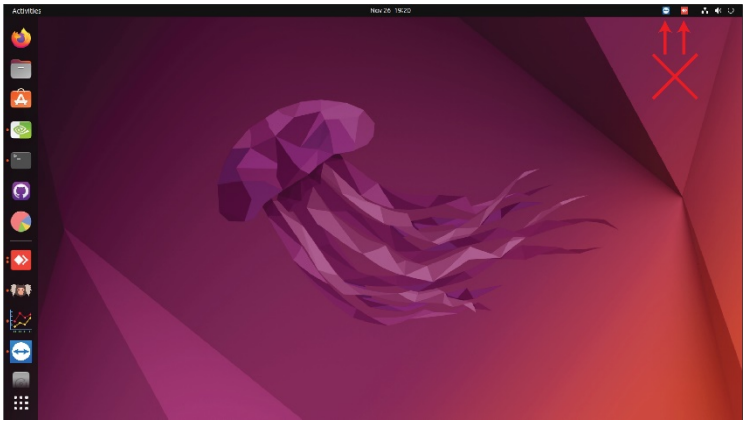
23. Pump: See Calibration section

Experiment

- In the main control panel, click a desired experiment



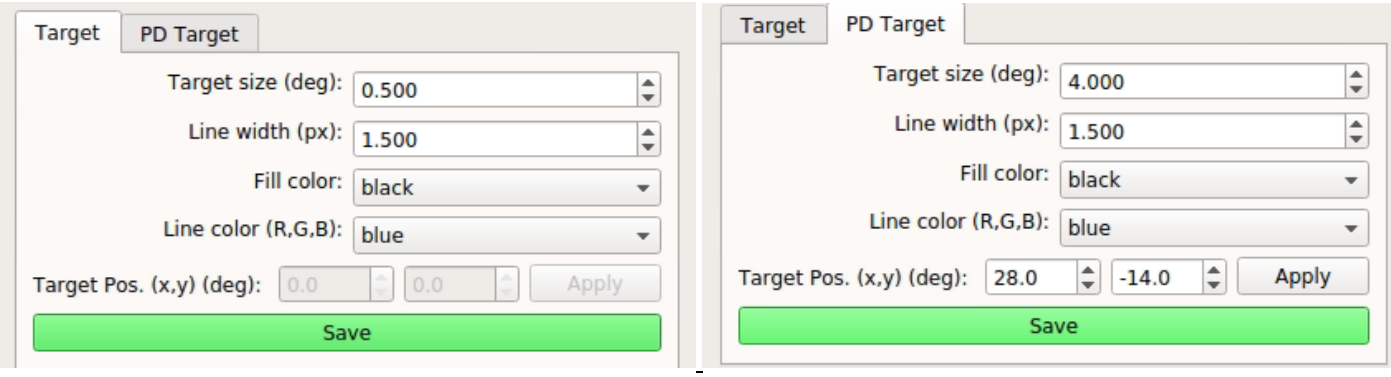
- Subject window will show up on (**3. Subject screen**) and experiment control panel will show up on (**2. Experiment screen #1**).
- While running experiment, minimize all windows; the user should be seeing a blank window like so:



- Remote desktop apps like AnyDesk or TeamViewer (see arrows) need to be completely closed, as they were found to hugely increase the display latency
- The user should not move mouse during the experiment as doing so was found to hugely increase the display latency
- A set of requirements above is one of the reasons for displaying real-time behavior and starting/stopping the task on the (**9. Behavior data/ephys computer**)

- Open real-time behavior GUI on (**9. Behavior data/ephys computer**); set it to receive data from the other computer
- Before running the experiment, be sure to set pump parameters on (**9. Behavior data/ephys computer**)

Target settings



- Target: parameters (self-explanatory) for the experimental target; for more customization, see PsychoPy
- PD Target: parameters (self-explanatory) for the (**19. photodiode signal)**
    - Target Pos.: set the position to where the photodiode is attached to the monitor; test to see if the target is detected well by the (**19. photodiode signal)**

Simple saccade

- The subject fixates at the center target, and a primary target is presented at a specified distance away from the center target. The subject makes a primary saccade to the primary target.

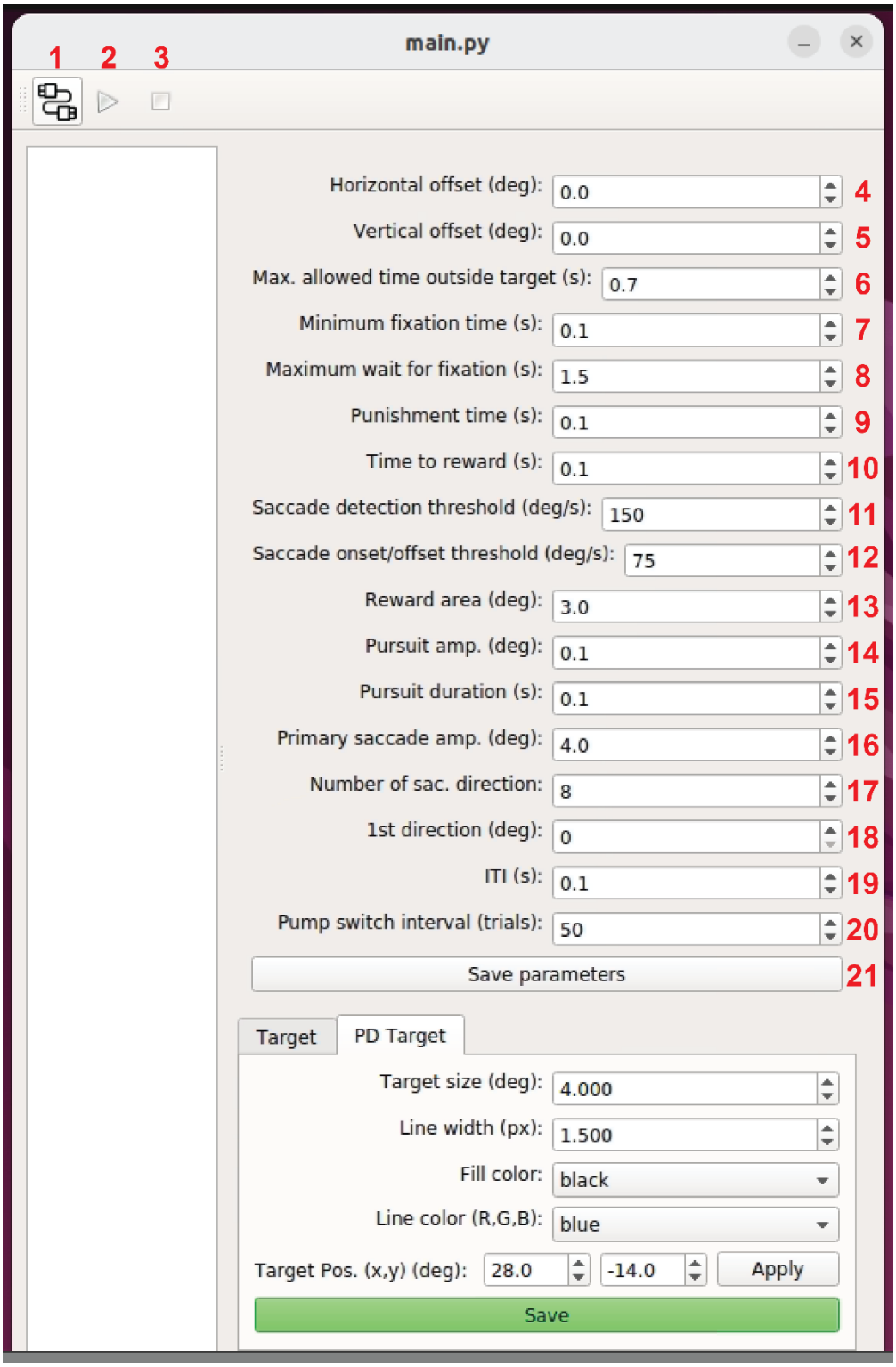


Figure 6. Simple saccade parameter panel

1. Clicking this will set the (**1. Behavior computer**) to receive commands from another computer.
    - Click this, set parameters, minimize this window, and run the experiment from the (**9. Behavior data/ephys computer**)

2 – 3. Start/stop the experiment; doing so from the (**9. Behavior data/ephys computer**) is recommended (see above 1.)

4. *Horizontal offset* (eye degrees): horizontal position of the center target relative to the center of the screen

5. *Vertical offset* (eye degrees): vertical position of the center target relative to the center of the screen
6. *Max. allowed time outside target* (second): unused
7. *Minimum fixation time* (second):
   - Minimum fixation time at center target before displaying the primary target
   - Time delay between 'END_TARGET_FIXATION' and 'TRIAL_SUCCESS'
     - No actual fixation required
8. *Maximum wait for fixation* (second):
   - During several states, maximum amount of time finite state machine will wait until the animal makes a correct action; when the time elapses, the state reverts to the initial state of the trial
9. *Punishment time* (second):
   - How long to wait in 'INCORRECT_SACCADE' state after animal makes an incorrect action
   - *Time to reward* (second): unused
10. *Saccade detection threshold* (eye degrees/second):
    - Speed of the primary saccade that will cause the state to change from 'SACCADE' to 'DETECT_SACCADE_START'
11. *Saccade onset/offset threshold* (eye degrees/second):
    - A saccade speed needs to be less than this value to be counted as the end of the saccade, and the finite state machine will check the offset position at that time
12. *Reward area* (eye degrees):
    - During several states, an eye position is required to be within a half of this value from a current target position
13. *Pursuit amp.* (eye degrees):
    - Amplitude of the pursuit target that appears before the center target appears.
    - A pursuit target appears a specified distance away from the center target and gradually moves toward the center
14. *Pursuit duration* (second): duration of the pursuit target
15. *Primary saccade amp.* (eye degrees): amplitude of the primary target
16. *Number of sac. Direction*:
    - Number of potential primary target positions, evenly distributed around the center target
    - For each trial, one target will be selected randomly
17. *1st direction* (angular degrees):
    - Direction of the first target that initializes the evenly distributed potential primary target positions
    - E.g., value of 90 degrees here and 2 for the number of saccade direction will result in two targets, up and down
18. *ITI* (second): inter-trial interval
19. *Pump switch interval*
    - Pump that gets activated upon completion of a trial will switch after this number of trials

Corrective saccade

- The subject fixates at the center target, and a primary target is presented at a specified distance away from the center target. The subject makes a primary saccade to the primary target. As soon as the primary saccade is detected, the primary target disappears and then the secondary target appears, at a specified distance away from the primary target position. For more information about the task, see (Sedahat-Nejad et al., 2022)

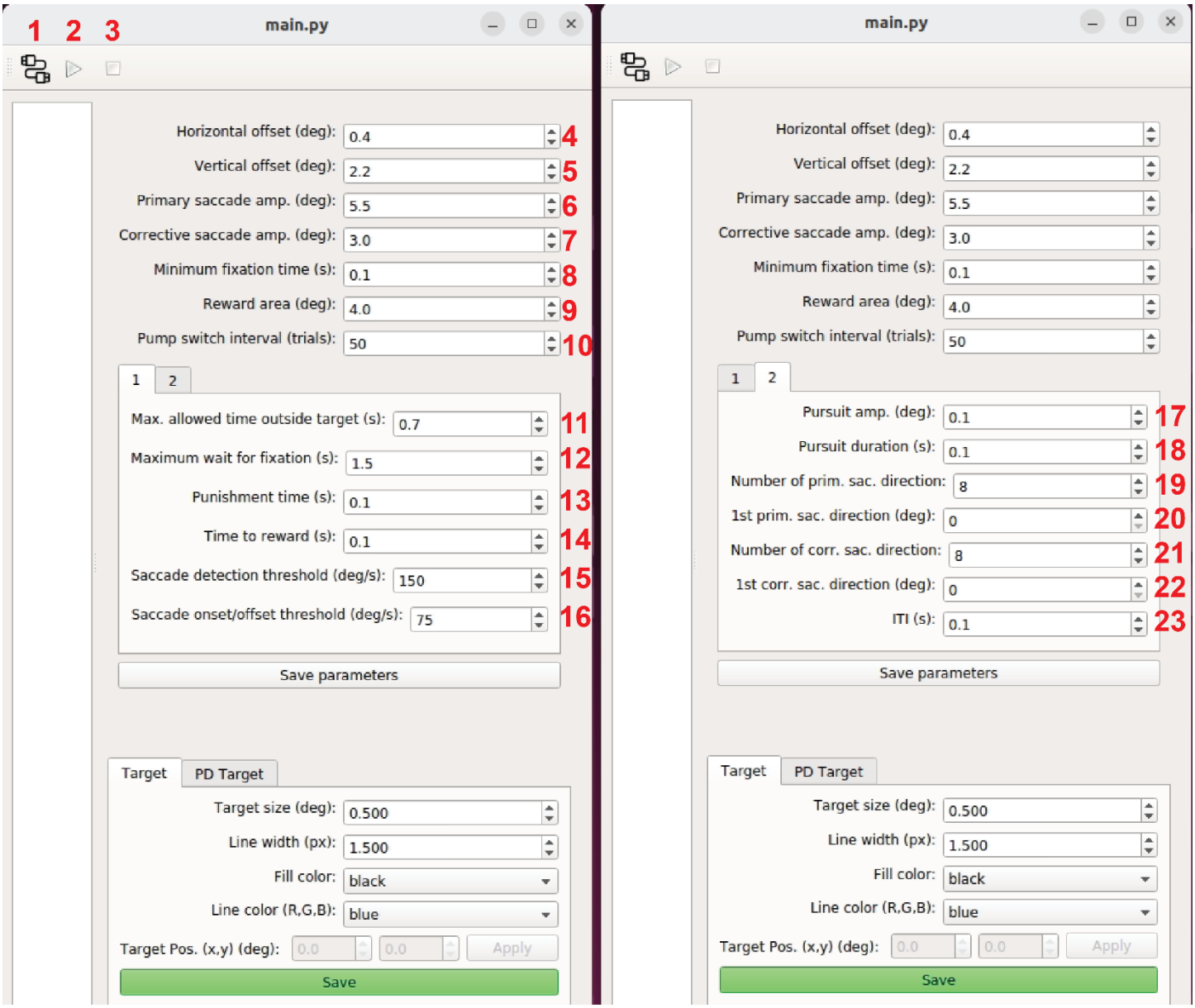


Figure 7. Corrective saccade parameter panel

1 – 6, 8 – 18. See Simple saccade experiment

7. *Corrective saccade amp*. (eye degrees): amplitude of the secondary target
21. *Number of corr. sac. direction*:
    - Number of potential secondary target positions, evenly distributed around the primary target
    - For each trial, one target will be selected randomly
22. *1st corr. sac. direction* (angular degrees):
    - Direction of the first target that initializes the evenly distributed potential secondary target positions

Internals

Overview of architecture

- The GUI was based on PyQt5
- As an example, an internal architecture when the corrective saccade experiment is running is shown below:

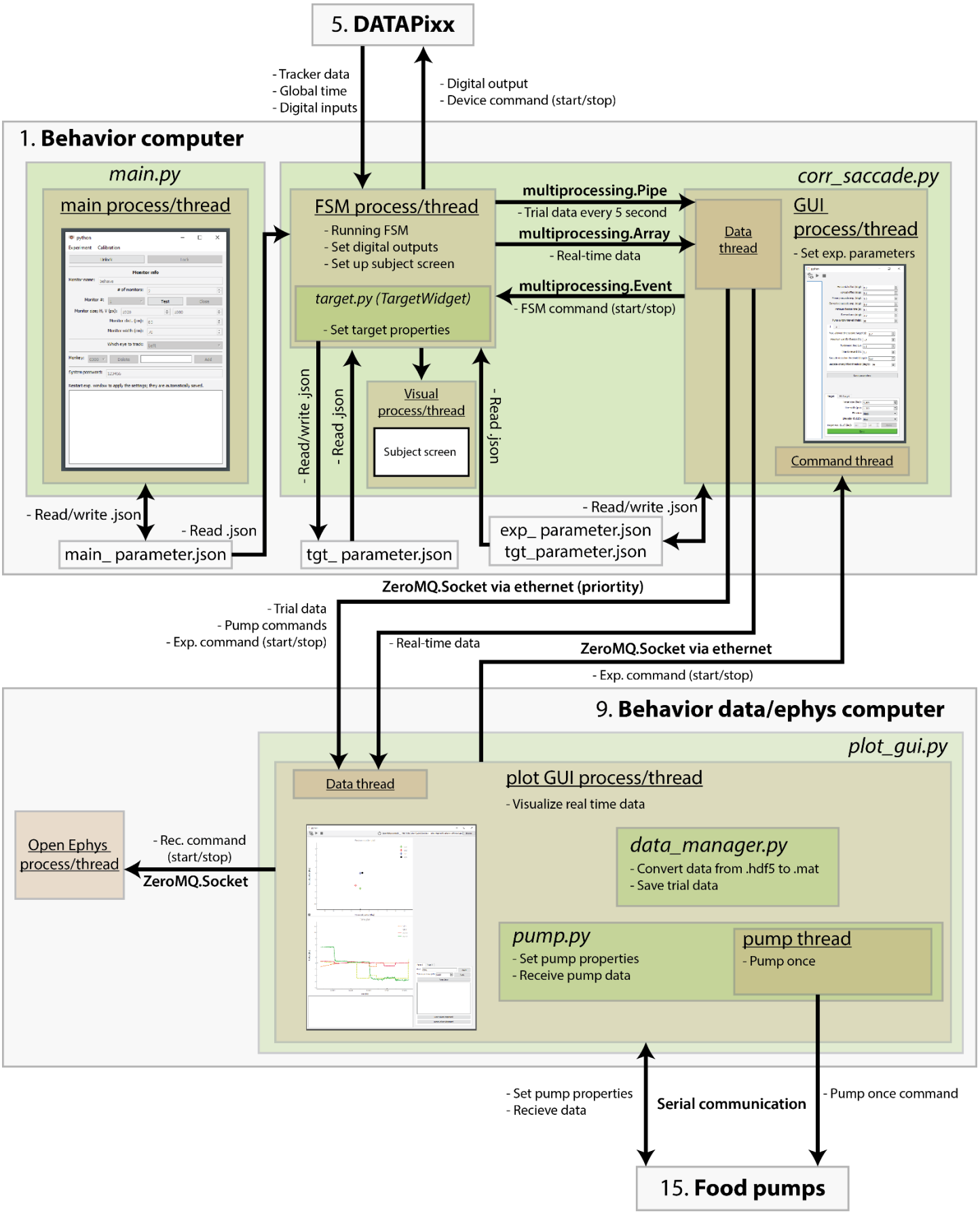


Figure 8. Internal architecture

- Within (**1. Behavior computer**), the main process/thread is run to control the main panel, which loads and saves the .json file for the screen parameters and etc. It also starts FSM and GUI processes of the experiment but does not communicate with them.
- Finite state machine (FSM) is run on a separate FSM process/thread, which controls the logic of FSM and set digital output. It also initializes a separate visual process/thread and sends command to display the targets. This thread interacts directly with (**5. DATAPixx**), receiving the incoming tracker data, digital inputs, and the device time, which is used as a global time in the whole system, and sending the digital output and device control commands to start/stop the tracker. This process requests trial data from (**5. DATAPixx**) and sends the data to GUI process/thread every five second via Python Pipe object; the data are chunked this way because requesting the data from the tracker incurs noticeable delay, which was minimized to promote better experience for the subject, which in turn is critical for his or her performance. Five second is not an absolute rule; any time that is a few seconds should work. This request for data occurs during non-critical period of FSM, so the task is not affected by the delay. The real-time data are sent every loop of the FSM via Python Array object.
- Setting the experiment parameters is run on GUI process/thread, which saves the experiment and target parameters in .json files to be read by the FSM thread. This process acts as the intermediary between the FSM process, which handles the incoming tracker data, and (**9. Behavior data/ephys computer**) to save the final data and visualize the real-time data. Also, this process sets the Python Event object that starts and stops the FSM. There is an independent thread that polls the Pipe to see if the trial data is sent over and accesses the real-time data in Array object. This thread sends these data to (**9. Behavior data/ephys computer**) via ZeroMQ Socket object**.** The real-time data are sent with a different Socket from the rest, as they do not need to be processed in time-sensitive manner; sending everything in one Socket was found to result in a significant delay to processing time-sensitive command like pumping. Another independent thread handles the command to start and stop the experiment from the (**9. Behavior data/ephys computer**).
- On (**9. Behavior data/ephys computer**), GUI process/thread that displays the real-time data is run. It also handles the tasks such as 1). sending the command to start and stop the experiment, 2). converting the final data from .hdf5 format to .mat format at the end of the experiment, and 3). setting pump properties and receiving data from the pump via serial communication. Because serial communication takes noticeable time, and setting of pump properties is run on the same thread as GUI, changing pump properties freezes the GUI. But as these changes occur only occasionally in the current paradigms, and the freezing does not affect the actual experiment, the decision was made to not create separate thread to handle these changes. However, sending the command to pump once, which occurs every trial, is handled by the separate thread and does not cause freezing. This thread also handles the communication with Open Ephys process. A separate thread handles the incoming data from the (**1. Behavior computer**).

Code structure of a standard module

- *corr_sac.py*, which runs the corrective saccade experiment is used as an example.
- In *corr_sac.py*, there are three classes: '…FsmProcess', '…GuiProcess', and '…Gui()'; the first two of which are processes.

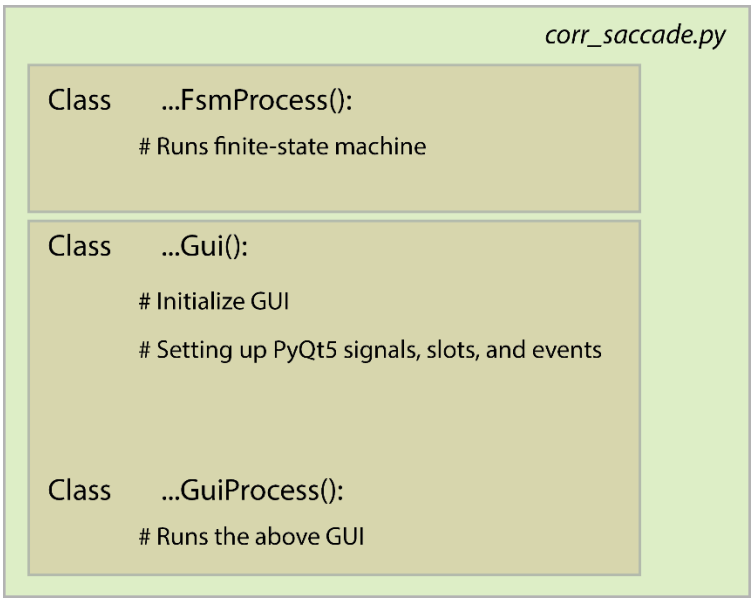


Figure 9. Code structure of a standard module

1) '…FsmProcess' runs the FSM
2) '…GuiProcess' runs the GUI as set up by '…Gui' class

- '…FsmProcess':
  1) After being initialized with various objects to communicate with '…GuiProcess', it runs the 'run()' function.
  2) It sets up the experiment screen according to the monitor parameter

```
# Set up exp. screen
this_monitor = monitors.Monitor(self.mon_parameter['monitor_name'], width=self.mon_parameter['monitor_width'], distance=self.mon_parameter['monitor_distance'])
this_monitor.save()
this_monitor.setSizePix(self.mon_parameter['monitor_size'])
self.window = visual.Window(size=self.mon_parameter['monitor_size'],screen=self.mon_parameter['monitor_num'], allowGUI=False, color='white', monitor=this_monitor,
                            units='deg', winType='pyglet', fullscr=True, checkTiming=False, waitBlanking=True)
self.window.flip()
```

     - Significant amount of the time was spent in testing different parameters for 'visual.Window' class
     - More details on the parameters
       - Unfortunately, not all parameters and how they affect the performance metrics like screen delay and stability are explained adequately, so a trial & error is required if one wants to test a different value for the parameter
       - Parameter list varies depending on the version of PsychoPy
       - Only thing that should definitely not change is 'units='deg'' as it specifies we want to use eye rotation coordinate for setting target positions
  3) Initialization of the targets by running 'update_target()'

```
576     def update_target(self):
577         tgt_parameter, _ = lib.load_parameter('','tgt_parameter.json',True,False,lib.set_default_tgt_parameter,'tgt')
578         pd_tgt_parameter,_ = lib.load_parameter('','tgt_parameter.json',True,False,lib.set_default_tgt_parameter,'pd_tgt')
579         self.tgt = visual.Rect(win=self.window, width=tgt_parameter['size'],height=tgt_parameter['size'], units='deg',
580                     lineColor=tgt_parameter['line_color'],fillColor=tgt_parameter['fill_color'],
581                     lineWidth=tgt_parameter['line_width'])
582         self.tgt.draw() # draw once already, because the first draw may be slower - Poth, 2018
583         self.pd_tgt = visual.Rect(win=self.window, width=pd_tgt_parameter['size'],height=pd_tgt_parameter['size'], units='deg',
584                     lineColor=pd_tgt_parameter['line_color'],fillColor=pd_tgt_parameter['fill_color'],
585                     lineWidth=pd_tgt_parameter['line_width'])
586         self.pd_tgt.pos = pd_tgt_parameter['pos']
587         self.pd_tgt.draw()
588         self.window.clearBuffer() # clear the back buffer of previously drawn stimuli - Poth, 2018
```

- It loads the target parameters as set by the target widget
- Right now, only square targets are supported, so if a different shape or image is to be used, this part along with the target widget would need to be modified

4) VPixx housekeeping

```
64          # Check if VPixx available; if so, open
65          DPxOpen()
66          tracker.TRACKPixx3().open() # this throws error if not device not open
67          DPxSetTPxAwake()
68          DPxSelectDevice('DATAPIXX3')
69          DPxUpdateRegCache()
70
71          # Get pointers to store data from device
72          cal_data, raw_data = lib.VPixx_get_pointers_for_data()
73
74          # Init. var.
75          random_signal_flip_duration = 0.015 # in sec., how often to flip random signal
76          bitMask = 0xffffff # for VPixx digital out, in hex bit
77          DPxSetDoutValue(0, bitMask)
78          DPxUpdateRegCache()
```

- Refer to VPixx Python library documentation or ask them directly for any questions.
- Briefly, 'DPxUpdateRegCache()' sets the registers in the DATAPixx to affect the changes; without this command, any instruction like opening the device or setting of digital input is not executed. It makes new data from the device available to be read; reading is a separate command; the execution of the command takes around 0.25 ms, so one needs to be mindful of this when implementing a sub-ms routine.
- 'raw_data' is a pointer (like in C/C++) to temporarily store the real-time eye data; we do not use 'cal_data' since we perform the calibration on our side, not on DATAPixx
- 'random_signal_flip_duration' sets how fast we flip the digital random signal for time alignment; through trial & error, we found this value less than 10 ms to be unstable (i.e., setting of digital output is missed on many occasions).
- 'bitMask' sets how many bits we want to use for digital output; by default, it is set to use all the bits

5) Start of the experiment loop

```
        # Process loop
1.      while not self.stop_fsm_process_Event.is_set():
2.          if not self.stop_exp_Event.is_set():
                # Turn on VPixx schedule; this needed to collect data
                lib.VPixx_turn_on_schedule()
                # Update targets
                self.update_target()
                # Load exp parameter
                fsm_parameter, parameter_file_path = lib.load_parameter('experiment','exp_paramet
                cal_parameter, _ = lib.load_parameter('calibration','cal_parameter.json',True,Tru
                # Create target list
                target_pos_list = lib.make_corr_target(fsm_parameter)
                num_tgt_pos = len(target_pos_list)
                # Init. var
                DPxUpdateRegCache()
                self.t = DPxGetTime()
                self.pull_data_t = self.t
                random_signal_t = self.t
                trial_num = 1
                pump_to_use = 1 # which pump to use currently
                vel_samp_num = 3
                vel_t_data = deque(maxlen=vel_samp_num)
                eye_x_data = deque(maxlen=vel_samp_num)
                eye_y_data = deque(maxlen=vel_samp_num)
                eye_pos = [0,0]
                eye_vel = [0,0]
                eye_speed = 0.0
                right_eye_blink = True
                left_eye_blink = True
                # Reset digital out
                dout_ch_1 = 1 # nominal PD
                dout_ch_3 = 0 # random signal
                dout_ch_5 = 1 # LED
                DPxSetDoutValue(dout_ch_1 + (2**2)*dout_ch_3 + (2**4)*dout_ch_5, bitMask)
                DPxUpdateRegCache()

                run_exp = True
            # Trial loop
3.          while not self.stop_fsm_process_Event.is_set() and run_exp:
                if self.stop_exp_Event.is_set():
                    run_exp = False
                    self.t = math.nan
                    # Turn off VPixx schedule
                    lib.VPixx_turn_off_schedule()
                    # Remove all targets
                    self.window.flip()
                    break
                # Init. trial variables; reset every trial
                self.init_trial_data()
                self.trial_data['right_cal_matrix'] = cal_parameter['right_cal_matrix']
                self.trial_data['left_cal_matrix'] = cal_parameter['left_cal_matrix']
                state = 'INIT'

                # FSM loop
4.              while not self.stop_fsm_process_Event.is_set() and run_exp:
                    if self.stop_exp_Event.is_set():
                        run_exp = False
                        self.t = math.nan
                        # Turn off VPixx schedule
```

- (1) Process loop that runs until the user closes the GUI for experimental parameters and sets ‘stop_fsm_process_Event’
- When the user opens the GUI, the (1) Process loop continuously checks the statements (2) and (3), which would be false since ‘stop_exp_Event’ is set and ‘run_exp’ would be false
- When the user presses run button, ‘stop_exp_Event is cleared and enters the conditional block in (2), reads experimental parameters, initializes variables, and etc, and finally sets ‘run_exp’ True. It is run once at the start of the experiment
- When ‘run_exp’ is true, the while block (3) runs until the user stops the experiment and initializes FSM state to be ‘INIT’; this block runs at the start of every trial

- (4) FSM block is constantly being looped during a trial

- '…GuiProcess' initializes GUI with various objects to communicate with '…FsmProcess'

```
1235  class CorrSacGuiProcess(multiprocessing.Process):
1236      def __init__(self, exp_name, fsm_to_gui_rcvr, gui_to_fsm_sndr, stop_exp_Event, stop_fsm_process_Event, real_time_data_Array, main_parame
1237          super(CorrSacGuiProcess,self).__init__(parent)
1238          self.exp_name = exp_name
1239          self.fsm_to_gui_rcvr = fsm_to_gui_rcvr
1240          self.gui_to_fsm_sndr = gui_to_fsm_sndr
1241          self.stop_exp_Event = stop_exp_Event
1242          self.real_time_data_Array = real_time_data_Array
1243          self.stop_fsm_process_Event = stop_fsm_process_Event
1244          self.main_parameter = main_parameter
1245      def run(self):
1246          app = QApplication(sys.argv)
1247          app_gui = CorrSacGui(self.exp_name, self.fsm_to_gui_rcvr, self.gui_to_fsm_sndr, self.stop_exp_Event, self.stop_fsm_process_Event, se
1248          app_gui.setWindowIcon(QtGui.QIcon(os.path.join('.', 'icon', 'experiment_window.png')))
1249          app_gui.show()
1250          sys.exit(app.exec())
1251
```

- '…Gui'
  1) Initializes ZeroMQ socket to communicate with (**9. Behavior data/ephys computer**)

```
669          # Create socket for ZMQ
670          try:
671              context = zmq.Context()
672              self.fsm_to_plot_socket = context.socket(zmq.PUB)
673              self.fsm_to_plot_socket.bind("tcp://192.168.0.2:5556")
674
675              self.fsm_to_plot_priority_socket = context.socket(zmq.PUB)
676              self.fsm_to_plot_priority_socket.bind("tcp://192.168.0.2:5557")
677
678              self.plot_to_fsm_socket = context.socket(zmq.SUB)
679              self.plot_to_fsm_socket.connect("tcp://192.168.0.1:5558")
680              self.plot_to_fsm_socket.subscribe("")
681              self.plot_to_fsm_poller = zmq.Poller()
682              self.plot_to_fsm_poller.register(self.plot_to_fsm_socket, zmq.POLLIN)
683          except Exception as error:
684              self.log_QPlainTextEdit.appendPlainText('Error in starting zmq sockets:')
685              self.log_QPlainTextEdit.appendPlainText(str(error) + '.')
686              self.toolbar_run_QAction.setDisabled(True)
687              self.toolbar_connect_QAction.setDisabled(True)
```

  2) Loads experimental parameters and populate the GUI with the saved values

```
689          # Load exp. parameter or set default values
690          self.exp_parameter, self.parameter_file_path = lib.load_parameter('experiment','exp_parameter.json',True,True,self.set_
691          self.cal_parameter, _ = lib.load_parameter('calibration','cal_parameter.json',True,True,lib.set_default_cal_parameter,'
692          self.update_parameter()
```

  3) Initializes PyQt5 signals, slots, and events. Information specific to the general operation of PyQt5 is found online.
  - When the user presses run, 'toolbar_run_QAction_triggered' is run:

```
734     def toolbar_run_QAction_triggered(self):
735         # Check to see if plot process ready
736      1. self.fsm_to_plot_priority_socket.send_pyobj(('confirm_connection',0))
737         # Wait for confirmation for 5 sec.
738         if self.plot_to_fsm_poller.poll(5000):
739             msg = self.plot_to_fsm_socket.recv_pyobj(flags=zmq.NOBLOCK)
740             if msg[0] == 0:
741                 self.toolbar_run_QAction.setDisabled(True)
742                 self.toolbar_stop_QAction.setEnabled(True)
743                 # Start FSM
744              2. self.stop_exp_Event.clear()
745                 # Disable some user functions
746                 self.sidepanel_parameter_QWidget.setDisabled(True)
747                 self.tgt.setDisabled(True)
748                 self.pd_tgt.setDisabled(True)
749                 # Save parameters
750                 self.save_QPushButton_clicked()
751                 # Init. data
752                 if self.cal_parameter['which_eye_tracked'] == 'Left':
753                     self.exp_parameter['right_eye_tracked'] = 0
754                     self.exp_parameter['left_eye_tracked'] = 1
755                 else:
756                     self.exp_parameter['right_eye_tracked'] = 1
757                     self.exp_parameter['left_eye_tracked'] = 0
758                 self.exp_parameter['version'] = 1.0
759                 self.fsm_to_plot_priority_socket.send_pyobj(('init_data',self.exp_name, self.exp_parameter))
760                 # Start timer to get data from FSM
761              3. self.data_QTimer.start(self.data_rate)
762                 # Tell plot GUI we are starting
763              4. self.fsm_to_plot_priority_socket.send_pyobj(('run',0))
764         else:
765             self.log_QPlainTextEdit.appendPlainText('No connection with plotting computer.')
```

- (1) Sends a message to see if the behavior data/ephys computer is ready and wait for the confirmation; this is the delay that you see when the relevant programs are not run on both computers.
- When you receive the confirmation of successful communication with the other computer, we clear the Event object (2), so that the experiment can start to run in '…FsmProcess', which can read this variable as the Event object is a Python object used for multiprocessing communication
- We start the QTimer object to start to receive any data from FSM; this is the 'Data_thread' of GUI process/thread in the internal architecture figure. This lives on a separate thread, so that the receiving of the data does not freeze the GUI and runs 'data_QTimer_timeout' at the frequency set by 'data_rate'.
- We tell the Behavior data/ephys computer that we are running the experiment now; this allows starting of the experiment in either computer

- When the use clicks Connect button, it starts the 'receiver_QTimer', which periodically runs 'receiver_QTimer_timeout'; this is the 'Command_thread' of GUI process/thread

```
783     def toolbar_connect_QAction_triggered(self):
784         '''
785         when triggered, start to receive messages from another computer
786         '''
787         self.receiver_QTimer.start(10)
788         self.toolbar_connect_QAction.setDisabled(True)
```

- IMPORTANT: 'data_QTimer_timeout' runs periodically when the experiment is running to receive the data from FSM processs/thread and send data to Behavior data/ephys computer to plot the real-time data

```
790     def data_QTimer_timeout(self):
791         '''
792         getting data from fsm process and send them to another computer
793         '''
794         if self.fsm_to_gui_rcvr.poll():
795             msg = self.fsm_to_gui_rcvr.recv()
796             msg_title = msg[0]
797             self.fsm_to_plot_priority_socket.send_pyobj(msg)
798             if msg_title == 'log':
799                 self.log_QPlainTextEdit.appendPlainText(msg[1])
800 
801         with self.real_time_data_Array.get_lock():
802             t = self.real_time_data_Array[0]
803             eye_x = self.real_time_data_Array[1]
804             eye_y = self.real_time_data_Array[2]
805             tgt_x = self.real_time_data_Array[3]
806             tgt_y = self.real_time_data_Array[4]
807         self.fsm_to_plot_socket.send_pyobj((t,eye_x,eye_y,tgt_x,tgt_y))
```

- 'fsm_to_gui_rcvr' is one of the pair that handles communication with '…FsmProcess'; the other one is 'fsm_to_gui_sndr'
  - Naming was chosen for readability of the code
- It is currently set to send any data to the other computer via 'fsm_to_plot_priority_socket'
- It reads Array object that is shared by FSM process/thread to obtain real-time data.

Experiment

- Overall structure

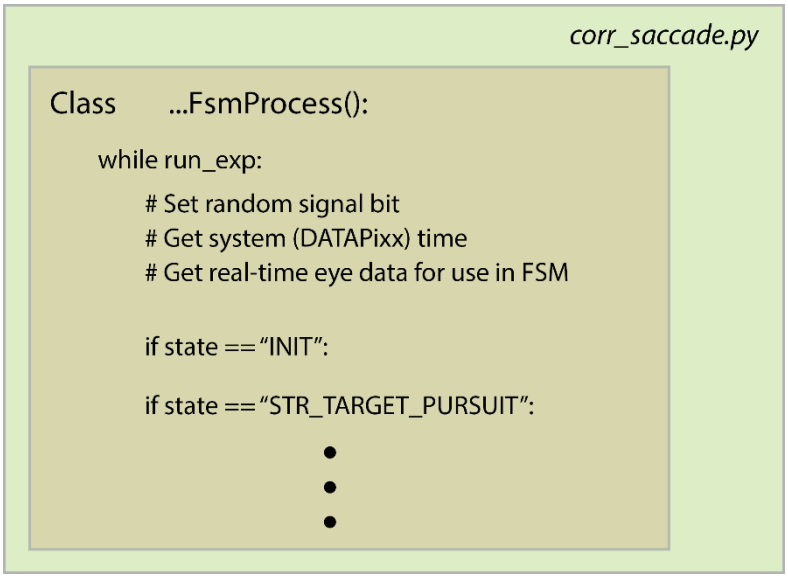


Figure 10. Overall structure of the experiment

1) Set random signal value

```
144                     # Send random signal for alignment
145                     if (self.t - random_signal_t) > random_signal_flip_duration:
146                         random_signal_t = self.t
147                         if random.random() > 0.5:
148                             dout_ch_3 = 1
149                         else:
150                             dout_ch_3 = 0
151                     DPxSetDoutValue(dout_ch_1 + (2**2)*dout_ch_3 + (2**4)*dout_ch_5, bitMask)
```

- Channel 3 is being used for the random signal

2) Get system time

```
                # Get time
                self.t = TPxBestPolyGetEyePosition(cal_data, raw_data) # this calls 'DPxUpdateRegCache' as well
```

- There are several functions to get time, but we use this function to update the real-time eye data as well
- IMPORTANT: this function calls 'DPxUpdateRegCache' by itself; if using different ways to get system time, 'DPxUpdateRegCache' needs to be called manually to update the local register with new device values or change the device registers (like setting the digital output)

3) Get real-time eye data

```
            # Get eye status (blinking)
            eye_status = DPxGetReg16(0x59A)
            right_eye_blink = bool(eye_status & (1 << 0)) # << 0- (animal's) right blink (pink); << 1-left blink (cyan)
            left_eye_blink = bool(eye_status & (1 << 1)) # << 0- (animal's) right blink (pink); << 1-left blink (cyan)
            if cal_parameter['which_eye_tracked'] == 'Right':
                if not right_eye_blink:
                    eye_blink = False
                    raw_data_right = [raw_data[0], raw_data[1],1] # [(animal's) right x, right y (pink), left x, left y (cyan)]
                    eye_pos = lib.raw_to_deg(raw_data_right,cal_parameter['right_cal_matrix'])
                    self.eye_x = eye_pos[0]
                    self.eye_y = eye_pos[1]
                    # Compute eye velocity
                    vel_t_data.append(self.t)
                    eye_x_data.append(self.eye_x)
                    eye_y_data.append(self.eye_y)
                    if len(vel_t_data)==vel_samp_num:
                        eye_vel[0] = np.mean(np.diff(eye_x_data)/np.diff(vel_t_data))
                        eye_vel[1] = np.mean(np.diff(eye_y_data)/np.diff(vel_t_data))
                        eye_speed = np.sqrt(eye_vel[0]**2 + eye_vel[1]**2)
                else:
                    eye_blink = True
                    self.eye_x = 9999 # invalid values; more stable than nan values for plotting purposes in pyqtgraph
                    self.eye_y = 9999
            else:
                if not left_eye_blink:
                    eye_blink = False
                    raw_data_left = [raw_data[2], raw_data[3],1] # [(animal's) right x, right y (pink), left x, left y (cyan)]
                    eye_pos = lib.raw_to_deg(raw_data_left,cal_parameter['left_cal_matrix'])
                    self.eye_x = eye_pos[0]
                    self.eye_y = eye_pos[1]
                    # Compute eye velocity
                    vel_t_data.append(self.t)
                    eye_x_data.append(self.eye_x)
                    eye_y_data.append(self.eye_y)
                    if len(vel_t_data)==vel_samp_num:
                        eye_vel[0] = np.mean(np.diff(eye_x_data)/np.diff(vel_t_data))
                        eye_vel[1] = np.mean(np.diff(eye_y_data)/np.diff(vel_t_data))
                        eye_speed = np.sqrt(eye_vel[0]**2 + eye_vel[1]**2)
                else:
                    eye_blink = True
                    self.eye_x = 9999 # invalid values; more stable than nan values for plotting purposes in pyqtgraph
                    self.eye_y = 9999
```

- Eye data were already received from the device along with the system time
- We access the device registers directly to see whether eyes are blinking.
    - If not blinking, change 'eye_x' and 'eye_y' after transforming the raw data into calibrated data and append eye data to the eye data array to compute the moving average of velocities; a recursive algorithm to filter the data could be used for faster computation, but the difference in computational speed was negligible (a few microseconds), so an explicit computation is used for better readability.
    - If blinking, we set the 'eye_x' and 'eye_y' to 9999, explaining why the real-time plots jump when they are set to auto-range.

4) The following blocks are several If statements to control the FSM.

- Target visualization
  - After target positions are set, the targets are visualized by calling 'draw()' and flipping the 'window', which flips the front and back buffer in the display

```
244                         self.tgt.pos = (self.tgt_x,self.tgt_y)
245                         self.tgt.draw()
246                         self.pd_tgt.draw()
247                         self.window.flip()
```

- Finite state machine (FSM)
  - This part handles the logic of the gaze-contingent paradigm by sequentially changing the 'state', depending on what the eye is doing.
  - The flow of the states is self-explanatory, so only the particular elements, the importance of which may not be obvious, are mentioned here.
    - Flipping the window without drawing anything removes anything current on display
    - Setting of digital outputs depending on the state
      - Channel 1 is used as a photodiode command signal sent to (12) Intan board to indicate when we are turning on the targets
      - The photodiode command signal is used to indicate the onset of target during certain states: 'STR_TARGET_PURSUIT', 'CUE_TARGET_PRESENT', and 'DETECT_SACCADE_END'. It is turned off during other states.
        - To correspond digital output value with the value outputted by the photodiode itself, 0 digital output is used when the photodiode signal is turned on, as our target is colored black, which is indicated as 0 from our photodiode; if using a different photodiode, this value can be changed to match values.
      - Channel 5 is used to control LED signal to align the data from the (17) tongue behavior camera, and it is currently set to follow the channel 1
      - 'DPxUpdateRegCache' is called right away to change the device register:

```
231                         dout_ch_1 = 0
232                         dout_ch_5 = 0
233                         self.pd_tgt.draw()
234                         DPxSetDoutValue(dout_ch_1 + (2**2)*dout_ch_3 + (2**4)*dout_ch_5, bitMask)
235                         DPxUpdateRegCache() # calling this delays fsm by ~0.25 ms
236                         self.window.flip()
237                         state = 'STR_TARGET_PURSUIT'
```

        - Technically, this is not needed as the device register update occurs on every loop of the FSM. But we found that calling this update right after drawing the target is more robust than not calling it. Without it, sometimes we found the actual photodiode signal arrives earlier than its command signal on (12) Intan, which should never happen, probably because there may be some execution of codes that take a significant amount of time (few ms) before the device register update is called. The only cost of this implementation is the delay of about ~0.25 ms, which should not matter in most paradigms.
      - When we turn the photodiode off (value of 1), the window is flipped always without drawing the photodiode target, which extinguishes it.

```
253                         dout_ch_1 = 1
254                         dout_ch_5 = 1
255                         self.tgt.draw()
256                         DPxSetDoutValue(dout_ch_1 + (2**2)*dout_ch_3 + (2**4)*dout_ch_5, bitMask)
257                         DPxUpdateRegCache() # calling this delays fsm by ~0.25 ms
258                         self.window.flip()
```

- IMPORTANT: digital output values are automatically saved in DATAPixx, so the photodiode command signal does not need to be recorded separately. In this specific case, it can be inferred by our state changes, so whether or not digital output values are saved does not matter, but the fact may be important in other cases.

- Every 5 second in 'STR_TARGET_PURSUIT', 'pull_data()' is called to ask DATAPixx for the eye data, which have a constant, specified sampling frequency (2000 Hz in this case), to be saved.
  - This is because calling 'TPxReadTPxData()' takes time that increases with the amount of the data to be received. So we ask for the data periodically and reset the buffer in DATAPixx.

```
555     def pull_data(self):
556         '''
557         to be called every 10 s or when a trial finishes, whichever is earlier
558         and flush the data from VPixx. Reason for this is to prevent the data
559         from accumulating, which will incur a delay when getting data
560         '''
561         tpxData = TPxReadTPxData(0)
562         self.trial_data['device_time_data'].extend(tpxData[0][0::22])
563         self.trial_data['eye_lx_raw_data'].extend(tpxData[0][16::22])
564         self.trial_data['eye_ly_raw_data'].extend(tpxData[0][17::22])
565         self.trial_data['eye_l_pupil_data'].extend(tpxData[0][3::22])
566         self.trial_data['eye_l_blink_data'].extend(tpxData[0][8::22])
567         self.trial_data['eye_rx_raw_data'].extend(tpxData[0][18::22])
568         self.trial_data['eye_ry_raw_data'].extend(tpxData[0][19::22])
569         self.trial_data['eye_r_pupil_data'].extend(tpxData[0][6::22])
570         self.trial_data['eye_r_blink_data'].extend(tpxData[0][9::22])
571         self.trial_data['din_data'].extend(tpxData[0][7::22])
572         self.trial_data['dout_data'].extend(tpxData[0][10::22])
573 
574         TPxSetupTPxSchedule() # flushes data in DATAPixx buffer
```

  - The data, which are read from 'tpxData' array, are saved in 'trial_data' dictionary

- Punishment sound is currently disabled

```
346                     # If time runs out before saccade detected, play punishment sound and reset the trial
347                     elif (self.t - state_start_time) >= fsm_parameter['max_wait_for_fixation']:
348                         ######
349                         # lib.playSound(200,0.1) # punishment beep
350                         ######
351                         state_start_time = self.t
352                         state_inter_time = self.t
353                         self.trial_data['state_start_t_str_tgt_pursuit'].append(self.t)
354                         dout_ch_1 = 0
355                         dout_ch_5 = 0
356                         self.pd_tgt.draw()
357                         DPxSetDoutValue(dout_ch_1 + (2**2)*dout_ch_3 + (2**4)*dout_ch_5, bitMask)
358                         DPxUpdateRegCache() # calling this delays fsm by ~0.25 ms
359                         self.window.flip()
360                         state = 'STR_TARGET_PURSUIT'
```

- Reward is given right after we detect the end of a desired saccade instead of after fixation at the target:

```
393                 if state == 'DETECT_SACCADE_END':
394                     if (eye_speed < fsm_parameter['sac_on_off_threshold']) and (self.t-state_start_time > 0.005):#25):
395                         # Check if saccade made to cue or end tgt.
396                         eye_dist_from_cue_tgt = np.sqrt((self.cue_x-self.eye_x)**2 + (self.cue_y-self.eye_y)**2)
397                         eye_dist_from_end_tgt = np.sqrt((self.end_x-self.eye_x)**2 + (self.end_y-self.eye_y)**2)
398                         if ((eye_dist_from_cue_tgt < fsm_parameter['rew_area']/2) or (eye_dist_from_end_tgt < fsm_parameter['rew_area']/2)):
399                             state_start_time = self.t
400                             state_inter_time = self.t
401                             self.trial_data['state_start_t_deliver_rew'].append(self.t)
402                             state = 'DELIVER REWARD'
```

  - This condition was initially a mistake in the code, but we found that giving reward right away increases the performance of the animal, especially first training him or her, and the animal usually fixates regardless. It may need to be changed if a strict condition is necessary for certain paradigm

- Changing of which pump to use happens during 'DELIVER_REWARD' state:

```
426                     if state == 'DELIVER_REWARD':
427                         eye_dist_from_cue_tgt = np.sqrt((self.cue_x-self.eye_x)**2 + (self.cue_y-self.eye_y)**2)
428                         eye_dist_from_end_tgt = np.sqrt((self.end_x-self.eye_x)**2 + (self.end_y-self.eye_y)**2)
429                         if eye_dist_from_end_tgt < fsm_parameter['rew_area']/2:
430                             if (trial_num % fsm_parameter['pump_switch_interval']) == 0:
431                                 if pump_to_use == 1:
432                                     pump_to_use = 2
433                                 else:
434                                     pump_to_use = 1
435                                 self.fsm_to_gui_sndr.send(('log','Pump switchd to '+str(pump_to_use)))
436                             self.fsm_to_gui_sndr.send(('pump_' + str(pump_to_use),0))
```

- Trial data are sent to 'GUI process/thread' at the end of the trial:

```
508                 if state == 'TRIAL_SUCCESS':
509                     if (self.t-state_start_time) > fsm_parameter['ITI']:
510                         # Pull data
511                         self.pull_data_t = self.t
512                         self.pull_data()
513                         # Send trial data to GUI
514                         self.fsm_to_gui_sndr.send(('log',datetime.now().strftime("%H:%M:%S") + '; trial num: ' + str(trial_num) + ' -> c
515                         self.fsm_to_gui_sndr.send(('trial_data',trial_num, self.trial_data))
516                         trial_num += 1
517                         self.init_trial_data()
518                         self.trial_data['right_cal_matrix'] = cal_parameter['right_cal_matrix']
519                         self.trial_data['left_cal_matrix'] = cal_parameter['left_cal_matrix']
520                         state = 'INIT'
```

  - 'pull_data()' is called once more to get the most recent data, to be concatenated to the existing trial data if one already available.
  - IMPORTANT: any data is sent to GUI process/thread via 'fsm_to_gui_sndr'. Almost always, an array is sent where the first element is the string that describes what is being sent, and the next elements are the data. What happens to the sent data is completely up to '…Gui'

- Adding new experiment
  - To add a new experiment, *main.py* needs to be updated, and a new experiment file that is located in 'experiment' folder is needed.
  - As an example, let's say we want to add an experiment that has something to do how the changes in the reward that animal gets affects his behavior.
  - *main.py*
    - Modifying the GUI
      - Add a new menu option for the new experiment:

```
33          # Actions
34          self.simple_sac_QAction = QAction('Simple Saccade',self)
35          self.menu_exp.addAction(self.simple_sac_QAction)
36          self.corr_sac_QAction = QAction('Corrective Saccade', self)
37          self.menu_exp.addAction(self.corr_sac_QAction)
38          self.cal_QAction = QAction('Calibration',self)
39          self.menu_cal.addAction(self.cal_QAction)
40          self.refine_cal_QAction = QAction('Refinement',self)
41          self.menu_cal.addAction(self.refine_cal_QAction)
```

        - For ex.,
          - self.reward_exp_QAction = QAction('Reward Task', self)
          - self.menu_exp.addAction(self.reward_exp_QAction)
      - Add a signal to be triggered when the new experiment is clicked:

```
216         #%% Signals
217         self.simple_sac_QAction.triggered.connect(self.simple_sac_QAction_triggered)
218         self.corr_sac_QAction.triggered.connect(self.corr_sac_QAction_triggered)
219         self.cal_QAction.triggered.connect(self.cal_QAction_triggered)
220         self.refine_cal_QAction.triggered.connect(self.refine_cal_QAction_triggered)
```

- For ex., `self.reward_exp_QAction.triggered.connect(self.reward_exp_QAction_triggered)`

- Add your own triggered function like below. Not much change needs to be made except, of course, the name of the function and 'exp_name' variable, which will be saved in your data.

```
262     def corr_sac_QAction_triggered(self):
263         # Empty Linux log files; communication with tracker fills up the files, eventually crashing
264         sys_password = self.sys_password_QLineEdit.text()
265         cmd_output = os.system("echo %s | sudo -S sh -c 'echo > /var/log/syslog'" % (sys_password))
266         os.system("echo %s | sudo -S sh -c 'echo > /var/log/syslog.1'" % (sys_password))
267         if cmd_output != 0:
268             self.log_QPlainTextEdit.appendPlainText("Input correct password to clear log files and try again")
269             return
270 
271         self.save_parameter()
272 
273         # Create a separate process for finite state machine (FSM) to run exp. and
274         # for GUI to control it.
275         # Using Pipe to transfer data btwn processes.
276         # Using Event to control stop and start of the experiment
277         stop_exp_Event = multiprocessing.Event()
278         stop_exp_Event.set()
279         stop_fsm_process_Event = multiprocessing.Event()
280         fsm_to_gui_rcvr, fsm_to_gui_sndr = multiprocessing.Pipe(duplex=False)
281         gui_to_fsm_rcvr, gui_to_fsm_sndr = multiprocessing.Pipe(duplex=False)
282 
283         real_time_data_Array = multiprocessing.Array('d', range(5))
284         exp_name = 'random_corrective_saccades'
285         fsm_process = CorrSacFsmProcess(exp_name, fsm_to_gui_sndr, gui_to_fsm_rcvr, stop_exp_Event, stop_fsm_process_Event, real_time_data_Array, self.main_parameter, self.mon_parameter)
286         gui_process = CorrSacGuiProcess(exp_name, fsm_to_gui_rcvr, gui_to_fsm_sndr, stop_exp_Event, stop_fsm_process_Event, real_time_data_Array, self.main_parameter)
287 
288         fsm_process.start()
289         time.sleep(0.25) # without this artificial delay, sometimes causes error
290         gui_process.start()
```

- Import your experiment in the beginning like so:

```
14  from experiment.simple_saccade import SimpleSacGuiProcess, SimpleSacFsmProcess
15  from experiment.corr_saccade import CorrSacGuiProcess, CorrSacFsmProcess
```

  - For ex., `from experiment.reward_task import RewardTaskProcess, RewardTaskFsmProcess`

- *reward_task.py* (to be located in 'experiment' folder)
  - In the new file, copy and paste *corr_saccade.py*, as the overall structure will be similar.
  - Change the class names:

```
29    class CorrSacFsmProcess(multiprocessing.Process):
657   class CorrSacGui(FsmGui):
1235  class CorrSacGuiProcess(multiprocessing.Process):
```

  - In the 'run()' of the `RewardTaskGuiProcess`, change `CorrSacGui` to `RewardTaskGui` (or according to whatever your chosen naming convention)
  - Change FSM as you want

- Be sure to update ‘init_trial_data()’ to save the times that your states are enterd, if you add new states or change the existing ones

```
590     def init_trial_data(self):
591         '''
592         initializes a dict. of trial data;
593         needs to be called at the start of every trial
594         '''
595         self.trial_data = {}
596         self.trial_data['right_cal_matrix'] = [] # may be updated during exp.
597         self.trial_data['left_cal_matrix'] = []
598         self.trial_data['state_start_t_str_tgt_pursuit'] = []
599         self.trial_data['state_start_t_str_tgt_present'] = []
600         self.trial_data['state_start_t_str_tgt_fixation'] = []
601         self.trial_data['state_start_t_cue_tgt_present'] = []
602         self.trial_data['state_start_t_detect_sac_start'] = []
603         self.trial_data['state_start_t_saccade'] = []
604         self.trial_data['state_start_t_detect_sac_end'] = []
605         self.trial_data['state_start_t_deliver_rew'] = []
606         self.trial_data['state_start_t_end_tgt_fixation'] = []
607         self.trial_data['state_start_t_trial_success'] = []
608         self.trial_data['state_start_t_incorrect_saccade'] = []
609         self.trial_data['cue_x'] = []
610         self.trial_data['cue_y'] = []
611         self.trial_data['end_x'] = []
612         self.trial_data['end_y'] = []
613         self.trial_data['start_x'] = []
614         self.trial_data['start_y'] = []
615         self.trial_data['tgt_time_data'] = []
616         self.trial_data['tgt_x_data'] = []
617         self.trial_data['tgt_y_data'] = []
618         self.trial_data['eye_x_data'] = []
619         self.trial_data['eye_y_data'] = []
620          # 2000 Hz data
621         self.trial_data['eye_lx_raw_data'] = []
622         self.trial_data['eye_ly_raw_data'] = []
623         self.trial_data['eye_l_pupil_data'] = []
624         self.trial_data['eye_l_blink_data'] = []
625         self.trial_data['eye_rx_raw_data'] = []
626         self.trial_data['eye_ry_raw_data'] = []
627         self.trial_data['eye_r_pupil_data'] = []
628         self.trial_data['eye_r_blink_data'] = []
629         self.trial_data['device_time_data'] = []
630         self.trial_data['din_data'] = []
631         self.trial_data['dout_data'] = []
```

- You can append the time to your variable like so, which usually happens right before you assign a different state to ‘state’

```
473                 if state == 'END_TARGET_FIXATION':
474                     eye_dist_from_tgt = np.sqrt((self.tgt_x-self.eye_x)**2 + (self.tgt_y-self.eye_y)**2)
475                     if ((self.t - state_inter_time) >= fsm_parameter['min_fix_time']):
476                         state_start_time = self.t
477                         state_inter_time = self.t
478                         self.trial_data['state_start_t_trial_success'].append(self.t)
479                         self.window.flip() # remove all targets
480                         state = 'TRIAL_SUCCESS'
```

  - As a rule, you can save whatever variable you want by first initalizing the empty dictionary and appending to it in FSM.
  - For ex., if we want to keep track of which pump was active in a given trial, first make a variable in ‘init_trial_data()’ like `self.trial_data[‘which_pump_active’] = []`
  - And then append to it at an appropriate place, which in this case would at ‘DELIVER_REWARD’

```
426                     if state == 'DELIVER_REWARD':
427                         eye_dist_from_cue_tgt = np.sqrt((self.cue_x-self.eye_x)**2 + (self.cue_y-self.eye_y)**2)
428                         eye_dist_from_end_tgt = np.sqrt((self.end_x-self.eye_x)**2 + (self.end_y-self.eye_y)**2)
429                         if eye_dist_from_end_tgt < fsm_parameter['rew_area']/2:
430                             if (trial_num % fsm_parameter['pump_switch_interval']) == 0:
431                                 if pump_to_use == 1:
432                                     pump_to_use = 2
433                                 else:
434                                     pump_to_use = 1
435                                 self.fsm_to_gui_sndr.send(('log','Pump switchd to '+str(pump_to_use)))
436                             self.fsm_to_gui_sndr.send(('pump_' + str(pump_to_use),0))
```

- Right after ‘pump_to_use’ is newly assigned;
  self.trial_data[‘which_pump_active’].append(pump_tp_use)

- Change ‘set_default_parameter()’ found in both ‘…FsmGuiProcess’ and ‘…Gui’:

```
633     def set_default_parameter(self):
634         parameter = {
635                     'horz_offset':0.0,
636                     'vert_offset':0.0,
637                     'max_allow_time':0.7,
638                     'min_fix_time':0.1,
639                     'max_wait_for_fixation':1.5,
640                     'pun_time':0.1,
641                     'time_to_reward':0.1,
642                     'sac_detect_threshold':150.0,
643                     'sac_on_off_threshold':75.0,
644                     'rew_area':3.0,
645                     'pursuit_amp':0.1,
646                     'pursuit_dur':0.1,
647                     'prim_sac_amp':4.0,
648                     'num_prim_sac_dir':8,
649                     'first_prim_sac_dir': 0,
650                     'corr_sac_amp':2.0,
651                     'num_corr_sac_dir':8,
652                     'first_corr_sac_dir': 0,
653                     'ITI':0.1,
654                     'pump_switch_interval':50}
655         return parameter
```

- Change/add new parameters to be controlled in GUI:

```
913     #%% GUI
914     def init_gui(self):
915         # Disable plots
916         self.plot_1_PlotWidget.deleteLater()
917         self.plot_2_PlotWidget.deleteLater()
918         # Disable pumps
919         self.pump_1.deleteLater()
920         self.pump_2.deleteLater()
921         # Side panel tabs for extra params.
922         self.sidepanel_params_TabWidget= QTabWidget()
923         self.sidepanel_params_1_tab_QWidget = QWidget()
924         self.sidepanel_params_1_tab_QVBoxLayout = QVBoxLayout()
925         self.sidepanel_params_1_tab_QWidget.setLayout(self.sidepanel_params_1_tab_QVBoxLayout)
926         self.sidepanel_params_2_tab_QWidget = QWidget()
927         self.sidepanel_params_2_tab_QVBoxLayout = QVBoxLayout()
928         self.sidepanel_params_2_tab_QWidget.setLayout(self.sidepanel_params_2_tab_QVBoxLayout)
929         self.sidepanel_params_TabWidget.addTab(self.sidepanel_params_1_tab_QWidget, '1')
930         self.sidepanel_params_TabWidget.addTab(self.sidepanel_params_2_tab_QWidget, '2')
931         # Side panel
932         self.horz_offset_QHBoxLayout = QHBoxLayout()
933         self.horz_offset_QLabel = QLabel('Horizontal offset (deg):')
934         self.horz_offset_QHBoxLayout.addWidget(self.horz_offset_QLabel)
935         self.horz_offset_QLabel.setAlignment(Qt.AlignRight)
936         self.horz_offset_QDoubleSpinBox = QDoubleSpinBox()
937         self.horz_offset_QDoubleSpinBox.setValue(0)
938         self.horz_offset_QDoubleSpinBox.setMinimum(-50)
939         self.horz_offset_QDoubleSpinBox.setMaximum(50)
```

- Add the signal for your parameter and the slot
- Change ‘update_parameter()’ to load the saved parameters to your GUI parameter panel at the start of the experiment

Misc.

- *main.py*
  - Any general parameter applicable for all experiments is added here
    - Change ‘set_default_parameter’ accordingly
  - Any module initializes Python objects required for communication between processes (1) and start two processes, one for FSM and the other for parameter GUI (2).

```
262 def corr_sac_QAction_triggered(self):
263     # Empty linux log files; communication with tracker fills up the files, eventually crashing
264     sys_password = self.sys_password_QLineEdit.text()
265     cmd_output = os.system("echo %s | sudo -S sh -c 'echo > /var/log/syslog'" % (sys_password))
266     os.system("echo %s | sudo -S sh -c 'echo > /var/log/syslog.1'" % (sys_password))
267     if cmd_output != 0:
268         self.log_QPlainTextEdit.appendPlainText("Input correct password to clear log files and try again")
269         return
270
271     self.save_parameter()
272
273     # Create a separate process for finite state machine (FSM) to run exp. and
274     # for GUI to control it.
275     # Using Pipe to transfer data btwn processes.
276     # Using Event to control stop and start of the experiment
277 1.  stop_exp_Event = multiprocessing.Event()
278     stop_exp_Event.set()
279     stop_fsm_process_Event = multiprocessing.Event()
280     fsm_to_gui_rcvr, fsm_to_gui_sndr = multiprocessing.Pipe(duplex=False)
281     gui_to_fsm_rcvr, gui_to_fsm_sndr = multiprocessing.Pipe(duplex=False)
282
283     real_time_data_Array = multiprocessing.Array('d', range(5))
284     exp_name = 'random_corrective_saccades'
285 2.  fsm_process = CorrSacFsmProcess(exp_name, fsm_to_gui_sndr, gui_to_fsm_rcvr, stop_exp_Event, stop_fsm_process_Event, real_time_data_Array, self.main_parameter, self.mon_parameter)
286     gui_process = CorrSacGuiProcess(exp_name, fsm_to_gui_rcvr, gui_to_fsm_sndr, stop_exp_Event, stop_fsm_process_Event, real_time_data_Array, self.main_parameter)
287
288     fsm_process.start()
289     time.sleep(0.25) # without this artificial delay, sometimes causes error
290     gui_process.start()
```

- *data_manager.py*
  - Data manager is initialized in *plot_gui.py*

- It starts a data file and saves trial data, according to the messages the plot gui process receives from FSM process. See 'receiver_QTimer_timeout()' in *plot_gui.py*, msg_title == 'trial_data' and msg_title == 'init_data'
  - The functions of the QTimer was described elsewhere. Briefly, it manages "Data_thread" in (9). Behavior data/ephys computer, plot GUI process/thread
- Note that it inherits from QRunnable class and has a 'run()' function, so that it supports a creation of separate thread to handle saving of the data, to avoid freezing of the real-time plotting GUI, but chunking the trial data turned out to be enough to avoid noticeable delay, so this functionality is not currently being used

- *app_lib.py*
  - It contains useful functions used by more than one module such as playing sound, loading parameters, and making a list of target positions
- *pump.py*
  - The main classes are 'Pump' and 'PumpWidget'
    - 'Pump' inherits from QRunnable to support the creation of a separate thread to command the pump to start pumping; without a separate thread, there would be a delay in the real-time plot GUI every time pump is to be used.
      - It also has miscellaneous functionalities like reading the volume dispensed so far, as computed by the pump device, and resetting that volume.
    - 'PumpWidget' is the pump panel you see in the real-time plot GUI, and it manages all the button presses within that panel. It sends several serial commands to the pump for setup. Finer control of the pump is possible; see manufacturer documentation.
- *sound.py*
  - Currently not being used. Originally, it was created to manage all aspects of playing a sound, but since our sound requirement is so simple, consisting of beeps, the development stopped. If a more elaborate sound system is required, it may be developed further.
- *target.py*
  - Manages the GUI that you see to change target properties. Its only job is to let the user change the target parameters and save them, so that they could be read by FSM process when creating targets. If a more elaborate target is required, like an image, then this GUI would need to be modified along with 'update_target()' function within '…FsmProcess' class of the modules.