

User Manual
Version 1.0 (01/08/19)

1

Contents

1.	Introduction	5
2.	Contact	5
3.	Citation	5
4.	The REC-GUI framework	6
5.	Installation and Setup	7
5.1.	Required Packages	7
5.2.	Files Included with the REC-GUI framework	8
5.2.1.	Configuration Files.....	8
5.2.2.	Python Files	8
5.2.3.	Example MATLAB Scripts (Useful Learning Tools).....	9
6.	Opening the GUI.....	10
7.	Using the GUI: Functions and General Instructions	11
7.1.	Non-Configurable Core Widgets.....	12
7.1.1.	Subject Name Panel: Creating and Loading Subject Configuration.....	12
7.1.2.	Subject Configuration Panel	13
7.1.3.	Communication Panels between the GUI and its Counterparts.....	13
7.1.3.1.	Sending Panel	14
7.1.3.1.1.	Variable Name.....	14
7.1.3.1.2.	Identifier	14
7.1.3.1.3.	Value.....	14
7.1.3.1.4.	Submit Button	15
7.1.3.2.	Receiving Panel.....	15
7.1.3.2.1.	Variable Name.....	15
7.1.3.2.2.	Identifier	15
7.1.3.2.3.	Value.....	15
7.1.3.3.	Task Control Panel.....	15
7.1.3.3.1.	Creating a Task Configuration File.....	16
7.1.3.3.2.	Start, Stop, Pause and Exit	16
7.1.3.4.	Monitoring Panel	16
7.1.3.4.1.	Feedback Checkbox	17
7.1.3.4.2.	Dragging Checkbox	17
7.2.	Configurable Widgets	17
7.2.1.	Eye Control Configuration	17
7.2.1.1.	Enable/Disable the Eye Control Configuration Panel	18
7.2.1.2.	Eye Checkboxes.....	18

7.2.2.	Display Control Configuration	18
7.2.3.	Manual Eye Calibration (Example Code)	19
7.2.3.1.	Enable/Disable Manual Calibration	19
7.2.3.2.	Performing Manual Eye Calibration	20
7.2.4.	Receptive Field Mapping Panel (Example Code)	20
7.2.4.1.	Enable/Disable Receptive Field Mapping	20
7.2.4.2.	Using the Receptive Field Mapping Panel	21
7.2.4.2.1.	Grating Specific Properties	22
7.2.4.2.1.1.	Spatial Frequency	22
7.2.4.2.1.2.	Temporal Frequency	22
7.2.4.2.2.	Bar Specific Properties	22
7.2.4.2.3.	Dot Specific Properties	22
7.2.4.2.3.1.	Density	22
7.2.4.2.3.2.	Dot Size	22
7.2.4.2.3.3.	Speed	22
7.2.4.2.4.	General Stimulus Properties	22
7.2.4.2.4.1.	Diameter	22
7.2.4.2.4.2.	Depth	22
7.2.4.2.4.3.	Orientation	23
7.2.4.2.4.4.	Contrast	23
7.2.4.2.4.5.	Pulse Duration	23
7.2.4.2.4.6.	Inter-Pulse Interval	23
7.2.4.2.4.7.	Fixation Point Position	23
7.2.5.	User Programmable Buttons	23
7.2.5.1.	Enable/Disable User Programmable Buttons Panel	23
7.2.5.2.	Changing Button Labels	24
7.2.5.3.	Adding Callback Functions to Programmable Buttons	25
8.	Writing Code for the GUI Counterpart	25
8.1.	Basic coding structure.	25
8.1.1.	Coding Structure in MATLAB for a gaze fixation task	26
8.1.2.	Establishing UDP connections between MATLAB and the GUI.	27
8.1.3.	Main while-loop in MATLAB Script	27
8.2.	Network Configuration	29
8.3.	UDP Communication	29
8.3.1.	Sending UDP Packets from MATLAB to the GUI	30
8.3.2.	Sending UDP Packets from the GUI to MATLAB	30
8.3.2.1.	Predefined Outgoing UDP Packets	31
9.	Saved Data File Structure - GUI	31

9.1.	<i>Header Structure</i>	31
9.2.	<i>Configuration Structure.....</i>	32
9.3.	<i>Data Structure.....</i>	32
10.	<i>Reading Saved REC-GUI Data: Example Code for Reading Data Files</i>	33
11.	<i>Basic Coding Structure in MATLAB</i>	33
12.	<i>Phototransistor Circuit.....</i>	33
12.1.	<i>Parts</i>	33
12.2.	<i>Circuit Diagram.....</i>	33

1. Introduction

This document describes coding schemes and system configurations for the ‘Real-time Experimental Control with Graphical User Interface’ (REC-GUI) framework. Included are examples explaining how the REC-GUI framework works, as well as how to modify the framework to fit your experimental design.

The graphical user interface (GUI) was written in Python using the Tkinter package. The source code for the GUI is provided.

Python code and MATLAB scripts for implementing the REC-GUI framework are also provided.

IMPORTANT:

Depending on your experimental needs, additional resources may be required. This document describes example experimental procedures and provides instructions on how to modify the REC-GUI framework, but you will most likely need to modify and/or create new scripts to apply the REC-GUI framework to your application. The REC-GUI online forum is a valuable resource for customization: <https://recgui2018.wixsite.com/rec-gui/forum>

The REC-GUI framework is an open-source project. We encourage you and others to contribute to its continued development. We do not claim any copyright on the REC-GUI framework code, or responsibility for the results of others obtained using the framework. To contribute, please contact us via the forum.

2. Contact

You can visit us at the following sites:

Lab Website: <https://rosenberg.neuro.wisc.edu/>

REC-GUI Website: <https://recgui2018.wixsite.com/rec-gui>

REC-GUI GitHub: <https://github.com/rec-gui>

Please send questions to recgui2018@gmail.com, or visit the [FAQ page](#) on the REC-GUI website. The REC-GUI website includes a variety of resources including a forum to share ideas and solutions.

You can directly reach Byoungsoon Kim at bkim10@wisc.edu or Ari Rosenberg at ari.rosenberg@wisc.edu.

3. Citation

If you use the REC-GUI framework in your research, please include a citation to:

Byoungsoon Kim, Shobha Channabasappa Kenchappa, Adhira Sunkara, Ting-Yu Chang, Lowell Thompson, Raymond Doudlah, and Ari Rosenberg (2018) Real-Time Experimental Control using Network-Based Parallel Processing. bioRxiv 392654; doi: <https://doi.org/10.1101/392654>.

4. The REC-GUI framework

Modern neuroscience research often requires the coordination of multiple processes such as stimulus generation, real-time experimental control, as well as behavioral and neural measurements. The technical demands required to simultaneously manage these processes with high temporal fidelity limits the number of labs capable of performing such work. The Real-Time Experimental Control with Graphical User Interface (REC-GUI) framework is an open-source, network-based parallel processing framework that lowers this barrier. The framework offers multiple advantages, including: (i) a modular design that is agnostic to coding language(s) and operating system(s) to maximize experimental flexibility and minimize researcher effort, (ii) simple interfacing to connect multiple measurement and recording devices, (iii) high temporal fidelity by dividing task demands across CPUs, and (iv) real-time control using a fully customizable and intuitive GUI.

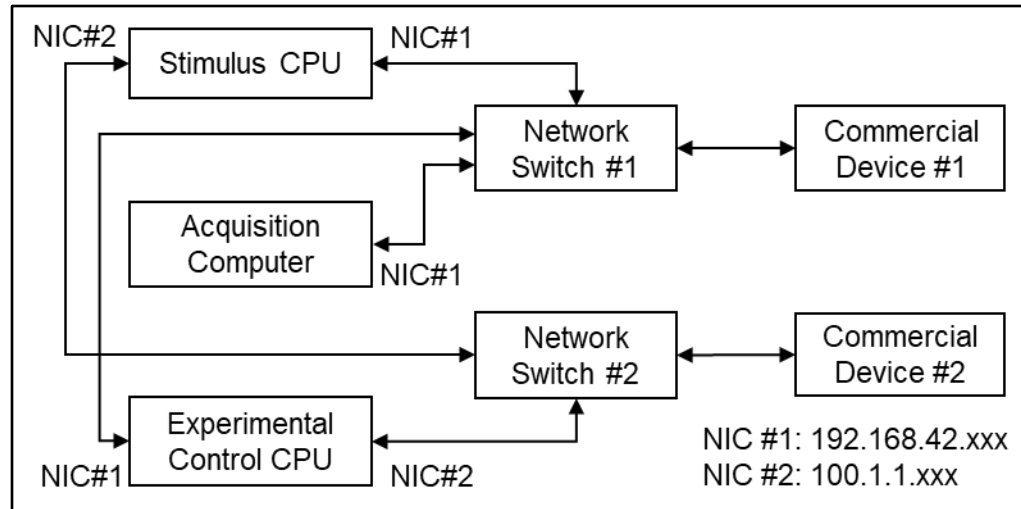


Figure 4.1: Network Communication Schematic

Experimental systems often require multiple pieces of specialized hardware produced by different companies (e.g., commercial devices 1 and 2). Network-based communications with such hardware must often occur over non-configurable, predefined subgroups of IP addresses. In **Figure 4.1**, commercial devices 1 and 2 have non-configurable, predefined subgroups of IP addresses that cannot be routed over the same network interface card (NIC) without additional configuration of the routing tables. In such cases, a simple way to set up communication with the hardware is to use multiple parallel networks with a dedicated network switch for each piece of hardware. For this hypothetical configuration, this requires two network switches, and both the stimulus and experimental control CPUs require two NICs assigned to different subgroups of IP addresses. With this configuration, the stimulus and experimental control CPUs can both directly communicate with commercial devices 1 and 2.

5. Installation and Setup

5.1. Required Packages

Main code for the REC-GUI framework.

- Download/clone code from GitHub (<https://github.com/rec-gui>)

Basic packages required for the REC-GUI framework:

- Python2.7 - base python interpreter
- Numpy - for calculations using arrays or matrices (<http://www.numpy.org>)
- Tkinter - for building the GUI (<https://tkdocs.com/tutorial/install.html>)
- MATLAB (2017a or greater is recommended)
 - MATLAB's Computer Vision System Toolbox
 - MATLAB's Data Acquisition Toolbox (if using USB-1680G for I/O)
 - Psychtoolbox 3 for visual stimulus generation (<http://psychtoolbox.org/>)

IMPORTANT:

The REC-GUI framework is written in Python and MATLAB. To implement the framework, additional software packages run by Python and MATLAB are required. Since these packages have different developers, we recommend checking with them regarding updates and usage. Skipping required packages can cause serious system errors.

Example packages for specific system configurations:

1. To use EyeLink (SR Research, Inc.).
 - Go to: <https://www.sr-support.com/>
 - Download 'EyeLink display software' for your operating system:
 - Linux - <https://www.sr-support.com/forum/downloads/eyelink-display-software/46-eyelink-developers-kit-for-linux-linux-display-software>
 - Windows - <https://www.sr-support.com/forum/downloads/eyelink-display-software/39-eyelink-developers-kit-for-windows-windows-display-software>
 - Mac OS X - <https://www.sr-support.com/forum/downloads/eyelink-display-software/45-eyelink-developers-kit-for-mac-os-x-mac-os-x-display-software>
2. To use USB-1680G (Measurement Computing, Inc.).
 - USB-1680G is a multifunctional DAQ used to handle analog I/O or digital I/O.
 - Linux
 - Ready for use with REC-GUI: <https://pypi.org/project/pydaqflex/>
 - <https://github.com/mccdaq/uldaq>
 - Windows
 - <https://github.com/mccdaq/mcculw>
3. To use a PC Parallel Port.
 - This is a simple solution for digital I/O. Example code is available at the links below.
 - Linux
 - Ready for use with REC-GUI - <https://github.com/pyserial/pyparallel>
 - Windows
 - <https://pypi.org/project/pyparallel/>

5.2. Files Included with the REC-GUI framework

The REC-GUI framework uses a variety of files and file types. The GUI uses a combination of configuration files and Python scripts to communicate with MATLAB. The files included in the GitHub distribution are listed below with a brief explanation. Further sections will use these files to not only teach you about the framework, but also help you get started with your own experiments.

5.2.1. Configuration Files

default.conf – required default system configuration file (see Section 6).

<subject name>.conf – required subject configuration file (see Section 6).

fixation.conf – an example system configuration file for implementing a gaze fixation task. To be used in conjunction with 'Fixation.m' (see Section 8.1.1).

calibration.conf – an example system configuration file for implementing an eye calibration task. To be used in conjunction with 'Calibration.m' (see Section 7.2.3).

receptive_field_mapping.conf – an example system configuration file for implementing a receptive field mapping task. To be used in conjunction with 'Receptive_Field_Mapping.m' (see Section 7.2.4).

stim_default.mat – a MATLAB configuration file that contains default stimulus parameters. It is tailored for particular stimuli rendered in MATLAB.

5.2.2. Python Files

The following is a list of files required for the GUI. Essential classes that need to be modified in order to adapt the REC-GUI framework to new experimental paradigms are marked in bold.

- analog.py – a class for handling analog to digital channels when using an external analog-to-digital converter (e.g., USB-1680G).
- **arbitrator_server.py** – a class for communication between the GUI and its counterparts.
- **button_funs.py** – a class that defines call-back functions for the user programmable buttons presented when a user selects 'None' for the eye tracking system (see Section 7.2.4).
- calibration.py – a class for handling eye calibration procedures.
- **constants.py** – a class that defines all system constants.
- data_collection.py – a class for saving the data file.
- digitalIO.py – a class for input/output digital bits to control TTL pulses using USB-1680G (Measurement Computing, Inc.)
- **eye_interpret.py** – a class for communication between the GUI and an eye tracking system (e.g., EyeLink, analog search coil, etc.).
- file.py – a class for writing the data file.
- **global_parameters.py** – a class that defines all system and configuration variables.
- **gui.py** – a class for building the GUI (in Tkinter). Start here to change any GUI component. A programming reference is found on the Tkinter website: (<https://tkdocs.com/tutorial/firstexample.html>).
- logger.py – a class for updating the data log window.

- **main.py** – Main function to spawn threads for the GUI, eye tracking, and system counterparts (e.g., a stimulus rendering/presentation CPU running MATLAB).
- **read_task_parameters.py** – a class for importing parameters from a saved task configuration file.
- **select_eye_recording.py** – a class that provides a GUI pop-up window for selecting the eye tracking method.
- **staircase.py** – a class for implementing a staircase procedure in a psychophysical task.
- **utility.py** – a class containing conversion routines for calculations.
- **vergence_version.py** – a class for calculating vergence and version of eye position.
- **widget_api.py** – a class for presenting specialized tool panels such as eye calibration, receptive field mapping, etc.

5.2.3. Example MATLAB Scripts (Useful Learning Tools)

Three MATLAB scripts that implement specific tasks are provided as examples for learning the overall coding scheme of the REC-GUI framework. Two additional skeleton scripts are provided as templates for preparing new tasks. At multiple points in the User Manual, these scripts are used as examples for explaining functions and coding strategies.

- **start_coding_closedloop.m** – Script that shows how to establish interactive control between the GUI and MATLAB. The MATLAB script receives computer mouse information (provided by user interactions with the Monitoring Panel) from the GUI, and sends that information back to the GUI. This input-output cycle provides the basis for implementing closed-loop control. The code 'Receptive_Field_Mapping.m' uses similar functions. Use 'start_coding_closedloop.conf' to configure the GUI to run this script.

IMPORTANT:

Run 'start_coding_closedloop.m' first after installing all of the required packages. This script requires no specialized hardware or software, and illustrates how the REC-GUI framework operates. Use this example code to see the coding structure, and how UDP packets are sent between MATLAB and the GUI.

- **Fixation.m** – Example script that implements gaze fixation training using standard operant conditioning procedures. This code presents a visual target at user-specified locations. If a subject fixates their gaze on the target for a specified duration of time, positive reinforcement is provided. This script provides a useful starting point for learning how to use Psychtoolbox with the REC-GUI framework. Use 'Fixation.conf' to configure the GUI to run this script.
- **Calibration.m** – Example script for performing eye calibration. This code implements a manual eye movement calibration procedure using the GUI. The user manually selects the location of a fixation target, and indicates when the subject's gaze is fixed on the target. After the subject has fixated multiple target locations, the GUI calculates offsets and gains to apply to the raw eye movement signals in order to map those signals to the true fixation locations. Use 'Calibration.conf' to configure the GUI to run this script.
- **Receptive_Field_Mapping.m** – Example script for mapping the visual field location of a neuron's receptive field during neurophysiological recordings. In this script, real-time interactive control of a visual stimulus is coordinated between the experimental control and stimulus CPUs. This script also shows how to implement the SDK toolbox (Ripple, Inc.) as an example of how to update event codes in a data acquisition system (e.g., used

for electrophysiological recordings). Use 'Receptive_Field_Mapping.conf' to configure the GUI to run this script.

- start_coding.m – Script that provides a template for preparing a new task. It contains code for initializing a UDP connection and a while-loop in which experiment specific code can be added. This script does not have a GUI configuration file. Follow the structure of the example '.conf' files to create one for this script.

6. Opening the GUI

To launch the GUI, enter the following commands in a command terminal:

```
cd <REC-GUI DIRECTORY HERE>
python main.py
```

The GUI loads 'default.conf' to set system parameters to default values. 'default.conf' is a system configuration text file that can be read or edited in any text editor.

IMPORTANT:

When editing any of the included files, be sure to preserve the structure of the code, including parentheses. Altering the structure of any file may result in serious system failures. We encourage you to customize the provided files, but edit them with caution.

Parameters in 'default.conf'

- a. Parameters for specialized tools that are automatically loaded, such as receptive field mapping, are defined here. Such parameters can be added or removed as necessary. For example, the 'receptive_field_mapping.m' example code uses the following parameters and default values:
 - i. dot_density: dot density in a random dot stimulus.
 - ii. dot_size: dot size (degree).
 - iii. dot_direction: motion direction of dots (degree).
- b. "arbitrator_service": defines IP addresses and UDP port numbers for CPUs (e.g., a stimulus CPU) that communicate with the experimental control CPU. UDP packets are broadcast to a specific IP address with a specific port number. 5001 through 5004 are default port numbers, and should be open in the firewall.
 - i. local_port: port number of the experimental control CPU. This number should match the server port number in the code running on the counterpart CPU (e.g., the stimulus CPU).
 - ii. server_port: port number of the counterpart. This number should match the local port number in the GUI code.
 - iii. local_port_eye: port number of the experimental control CPU. This number should match the server port number in the EyeLink system.
 - iv. server_port_eye: port number in the EyeLink system.
 - v. server_ip: IP address of the counterpart CPU.
 - vi. eyelink_service: IP address of the EyeLink CPU.

- vii. constant: default system parameters.
- viii. number_of_task_config_entries: defines how many lines are needed in the Sending Panel. Default is 30.
- ix. moving_eye_position_mean: defines the number of eye movement samples averaged per eye position estimate. Default is 25.
- x. data_size_length: UDP packet size. Default is 1,024 bytes.
- xi. data_padding_character: filler character for UDP packets. Default is 'q'.
- xii. message_delimiter: delimitator for UDP packets. Default is '/'.

7. Using the GUI: Functions and General Instructions

A core component of the REC-GUI framework is the GUI. It contains both configurable and non-configurable widgets. All widgets are defined in 'gui.py' using the Tkinter library. For the configurable widgets, a detailed description of how to modify them is provided in this section.

The current default configuration of the GUI has two operating modes: "vision research" and "non-vision research". If an eye tracking option is selected when the GUI starts, the GUI enters the vision research mode. This includes three vision research widgets: manual eye offset and gain controls, manual eye calibration, and receptive field mapping. Each of these widgets can be individually enabled/disabled in 'gui.py'. If 'None' is selected for the eye tracking system when the GUI starts, the GUI enters the non-vision research mode. This includes User Programmable Buttons that can trigger user-defined functions to provide specific controls. Users can reconfigure these User Programmable Buttons and add callback functions as needed. New user-defined configurations can also be created by editing 'gui.py' following the structure of these two modes.

The selection of the eye tracking system is defined as a class ('EyeRecord') located in the Python File 'select_eye_recording.py'. The selected method is saved as a variable (eye_recording_method_name) in the Python File 'global_parameters.py', and is used in 'gui.py' to initialize the GUI. The lines below, located in 'gui.py', can be modified as needed. For example, new widget functions can be defined in the highlighted places below (the 'if' case is the non-vision mode, and the 'else' case is the vision mode) to add specialized tools. In this way, new GUI configurations can be built.

Code snippet from 'gui.py':

```
207 globals.eye_recording_method_obj = EyeRecord()
208 globals.eye_recording_method_obj.dialog1.focus_force()
209 globals.gui_root.wait_window(globals.eye_recording_method_obj.dialog1)
210 # Wait here for user to make selection of which eye window to be selected
211 globals.eye_recording_method_obj.eye_recording_win.attributes("-topmost",True)
212 globals.gui_root.wait_window(globals.eye_recording_method_obj.eye_recording_win)
213
214 if globals.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globals.custom_button = []
216     globals.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220     <add new widget function>
221 else:
222     self.draw_subject_viewing_params()
223     # Draw eye offset, gain, and configuration
224     self.draw_eye_and_window_config()
225     # Draw eye calibration widgets
226     self.draw_eye_calibration_task_button()
227     # Do this in the end as it initializes some of the widgets as well
228     self.init_calibration_configuration_parameter(init=True)
229     # Receptive field mapping
230     self.draw_receptive_field_mapping_window()
231     <add new widget function>
232 tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.1. Non-Configurable Core Widgets

It is advised that the core system widgets (Subject Name, Task Control Box, Sending Panel, Receiving Panel, and Monitoring Panel) be treated as non-configurable since they support key system functions.

7.1.1. Subject Name Panel: Creating and Loading Subject Configuration

If the subject configuration file already exists, type in the name of the subject (e.g., "LVS") and select "Load". For a new subject, enter the subject's name and their configuration information into the configuration panel and select "Save" (see Section 7.1.1). This will generate a subject configuration file that can be loaded. The subject name is used as part of the filename together with the time and date that it was saved (e.g., 'LVS_04_28_17_00_02_42' for 'LVS' subject saved on 04/28/2017 at 00:02:42).

As an example, a subject configuration file can contain Eye Calibration and Eye Control Configuration settings for vision tasks, and the information can be broadcast to a counterpart CPU to prepare visual stimuli with the appropriate geometric configuration.

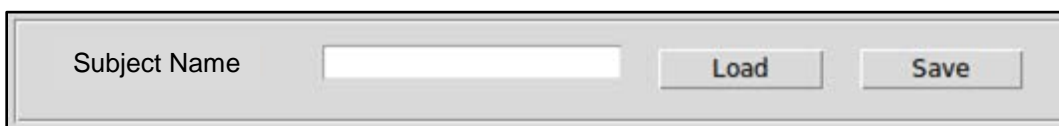


Figure 7.1: Subject Name Control Panel

IMPORTANT:

Note that clicking “Save” will alter the loaded subject configuration file. If you do not wish to overwrite the file, enter a new name and click “Save”.

Since the subject name is used as a file name for saving data, you must load your subject file prior to launching any experimental scripts on the counterpart.

7.1.2. Subject Configuration Panel

Subject configuration parameters are saved in the subject configuration file. Although the Subject Configuration Panel is a configurable widget (e.g., it is not presented in the ‘non-vision’ GUI mode), it is described in this section since it is closely linked to the Subject Name Panel. In the vision research mode of the GUI, the Subject Configuration Panel includes the subject’s inter-ocular distance (IOD), the screen distance, and screen size.

If you alter the configuration parameters, we recommend saving and reloading the subject configuration file to ensure that the correct configuration parameters are sent to the stimulus computer (*restart the GUI*).

To disable the ‘Eye Control Configuration’ Panel, find this section in ‘gui.py’ and comment out the highlighted lines below.

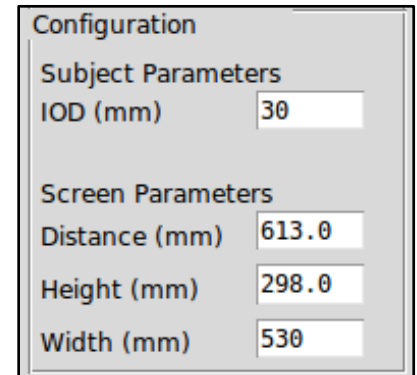


Figure 7.2: Subject Configuration Panel

Code snippet from ‘gui.py’:

```
214 if globls.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globls.custom_button = []
216     globls.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_viewing_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.1.3. Communication Panels between the GUI and its Counterparts

In the REC-GUI framework, communication between CPUs is essential to share the workload across major groups and to synchronize system components. This communication makes it possible to update parameters during runtime. The GUI uses an intuitive structure for real-time communication between CPUs.

The coding strategy for receiving updated parameters in the stimulus/counterpart CPU is described in Section 8.

7.1.3.1. Sending Panel

The GUI provides a sending panel as an interface to send information to code running on counterpart CPUs. For example, in the example gaze fixation training script 'Fixation.m', the duration required to hold fixation on a target in order to receive a reward may be changed. To do so, the experimenter enters the duration in the sending panel and the GUI sends the value to MATLAB by clicking submit. The sending panel has three columns of user input text boxes and a 'submit' button.

Variable Name	Identifier	Value
StimulusDuration	-106	1
InterTrialInterval	-101	1
FixationWindowHoldTime	-102	0.3
FixationAcquireTime	-103	5
MissedTrialDelay	-104	2
SaccadeAcquireTime	-105	0.5
Version_Fixation(deg)	-108	2
VersionSaccade(deg)	-109	3.5
Vergence(deg)	-110	1
VergenceOption	-111	1

Submit

Figure 7.3: Sending Panel

7.1.3.1.1. Variable Name

The variable name for each parameter should be an intuitive label. The GUI does not send the variable name, only the Identifier.

7.1.3.1.2. Identifier

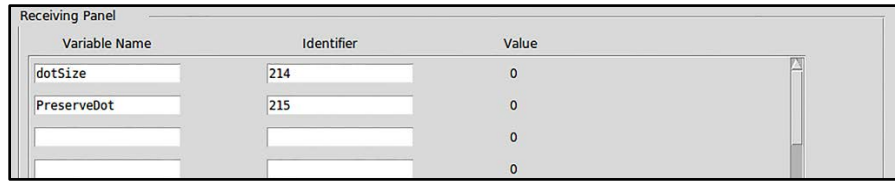
An integer value uniquely associated with the corresponding variable name. The identifier is used by the counterpart as the identity of a particular variable. See the example MATLAB scripts and Section 8.3 for examples of how identifiers can be used as “cases” in a switch command to alter a variable using the Value sent by the GUI.

7.1.3.1.3. Value

Input the value to assign to the associated variable in the counterpart. See the example MATLAB scripts and Section 8.3 for examples of how to extract a value and assign it to the appropriate variable using the Identifier.

7.1.3.1.4. Submit Button

Changes made to Identifiers or Values are not sent until the “Submit” button is pressed. To save the current parameter values in the Sending Panel to the Task Configuration file, click “Save” in the Task Control panel.



Variable Name	Identifier	Value
dotSize	214	0
PreserveDot	215	0
		0
		0

Figure 7.4: Receiving Panel

7.1.3.2. Receiving Panel

The GUI provides a receiving panel to monitor information from the counterparts in real-time. Just as the GUI sends ‘Identifiers’ and ‘Values’ to the counterparts, whenever counterparts send information to the GUI, the GUI updates the receiving panel for real-time monitoring.

7.1.3.2.1. Variable Name

An intuitive label for each parameter received. It does not need to match the variable name assigned to the variable in the counterpart.

7.1.3.2.2. Identifier

An integer value uniquely associated with the corresponding variable. This identifier is used by the counterpart as a signal to send a particular variable. See the example MATLAB scripts and Section 8.3 for examples of how identifiers are used to alter the value displayed by the GUI.

7.1.3.2.3. Value

This is the value associated with the variable, which is sent from the counterpart. See the example MATLAB scripts and Section 8.3 for examples of how to send the GUI these values using the Identifiers.

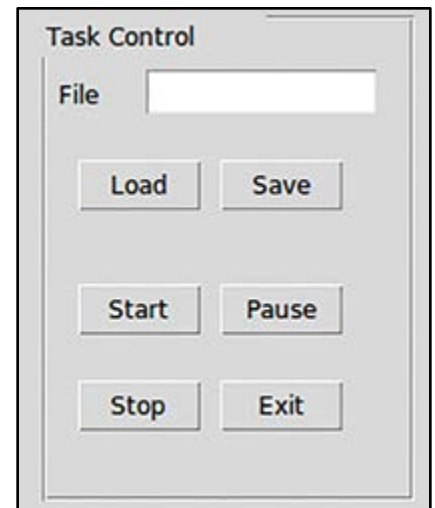


Figure 7.5: Task Control Panel

7.1.3.3. Task Control Panel

All configurations in the sending and receiving panels can be saved to a task configuration file. Dedicated task configuration files are recommended so that loading a specific task configuration file prepares the GUI to control a specific experiment only. As an example, input ‘Fixation.conf’ for the gaze fixation task included with the REC-GUI download.

“Start” begins the experiment after launching the experiment on the counterpart. “Pause” momentarily pauses the experiment until “Start” is selected again. “Stop” ends the experiment and clears the Subject and Task control panels, but leaves the GUI open. “Exit” closes the GUI.

IMPORTANT:

To begin a task, you must load a subject configuration file before starting the experiment script on the counterpart. After loading the subject configuration, you can launch the experiment on the counterpart and hit the “Start” button in the GUI.

7.1.3.3.1. Creating a Task Configuration File

In the Task Control Panel text box, enter the name of a new task configuration file.

Enter the variables, identifiers, and values you wish to manipulate in the Sending Panel, and the variables and identifiers to monitor in the Receiving Panel. Click “Save” to create the configuration file. The configuration file can now be loaded.

7.1.3.3.2. Start, Stop, Pause and Exit

Once the Task Configuration is loaded, you can send start, stop, pause, or exit commands to the counterpart CPUs. When these buttons are pressed, the GUI sends the events to the counterpart(s). See Section 8 for examples of how to handle these events in the counterpart code.

The packet structure of the information sent when these buttons are pressed is defined in ‘constants.py’, as follows:

- Start: Identifier = -2, Value = 100.
 - a. When clicked, the GUI sends the UDP packet:
“-2 100 qqqqqq.....q/” (total 1,024 bytes)
- Stop: Identifier = -2, Value = 101.
 - b. When clicked, the GUI sends the UDP packet:
“-2 101 qqqqqq.....q/” (total 1,024 bytes)
- Pause: Identifier = -2, Value = 102.
 - c. When clicked, the GUI sends the UDP packet:
“-2 102 qqqqqq.....q/” (total 1,024 bytes)
- Exit: Identifier = -2, Value = 104.
 - d. When clicked, the GUI sends the UDP packet:
“-2 104 qqqqqq.....q/” (total 1,024 bytes)

7.1.3.4. Monitoring Panel

The real-time display of continuous signals occurs in the Monitoring Panel. If an eye tracking system is selected when the GUI starts, the monitoring panel shows eye movement signals provided by an EyeLink or search coil system. If ‘None’ is selected, the Monitoring Panel displays signals received with the Identifier equal to ‘-6’. See Section 8.3 for details.

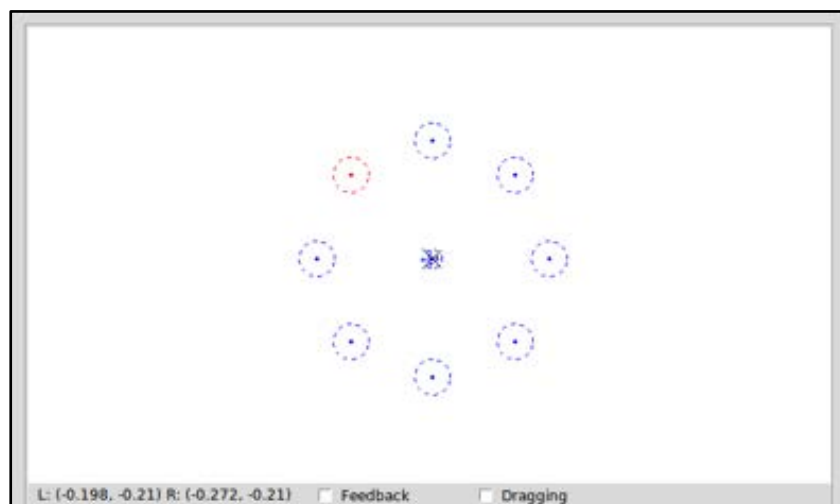


Figure 7.6: Monitoring Panel

The Monitoring Panel can also display behavioral enforcement “windows”. For example, this is a convenient way to monitor if a saccade was made to a specific target, or if a fixation window was violated. Above is an example (application 1 described in the REC-GUI manuscript) with 7 distractor windows (outer blue circles), a correct target window (red), version fixation window (center, blue), left and right eye positions (crosses in the fixation window), vergence fixation window (center, black), and vergence error (black x). Another example is to use the Monitoring Panel to evaluate if an animal entered a specific part of an arena (as in application 3 in the manuscript).

When the GUI receives a request to use “windows” from a counterpart CPU, it displays them at the requested locations using request colors, and in turn, reports if incoming signals have entered the window(s) or not.

7.1.3.4.1. Feedback Checkbox

If selected, when a position in the Monitoring Panel is clicked using the GUI mouse, the position is sent to an established counterpart CPU at the time that the button is released. In the ‘Receptive_Field_Mapping.m’ and ‘Fixation.m’ example codes, a user can use this feature to discretely change the location of a presented visual stimulus.

7.1.3.4.2. Dragging Checkbox

If selected, the position of the mouse cursor within the Monitoring Panel is continuously sent to an established counterpart CPU. In the ‘Receptive_Field_Mapping.m’ example code, a user can use this feature to continuously change the location of a presented visual stimulus.

7.2. Configurable Widgets

The GUI currently has two operating modes. If an eye tracking system is selected, it provides a default configuration for vision research. Under the Monitoring Panel, three panels are provided: Eye Control Configuration, Eye Calibration, and Receptive Field Mapping. If ‘None’ is selected for the eye tracking system, the GUI presents two panels: Display Control Configuration and User Programmable Command Buttons. Each of these panels is described below.

7.2.1. Eye Control Configuration

Adjust the offset and gain of the vertical and horizontal eye position signals, as well as the pupil size threshold (EyeLink specific). The values can be saved by clicking “Save” in the Subject Name Panel. If eye calibration is performed in EyeLink, Offsets should be initially set to 0, Gains to 1, and Pupil Sizes to 1.

The screenshot shows a window titled "Eye Control Configuration". Inside, there are two columns for "Horizontal" and "Vertical" settings, and a "Pupil Size" column. Each column has "Offset" and "Gain" sub-columns. The "Eye" column has checkboxes for "Left" and "Right".

Eye	Horizontal		Vertical		Pupil Size
	Offset	Gain	Offset	Gain	
<input checked="" type="checkbox"/> Left	-5.0	1.0	15.0	1.0	1
<input checked="" type="checkbox"/> Right	1.0	1.0	14.0	1.0	1

Figure 7.7: Eye Control Configuration Panel

7.2.1.1. Enable/Disable the Eye Control Configuration Panel

To disable the 'Eye Control Configuration' Panel, find this section in 'gui.py' and comment out the highlighted lines below.

Code snippet from 'gui.py':

```
214 if globs.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globs.custom_button = []
216     globs.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_viewling_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.2.1.2. Eye Checkboxes

Select the left and/or right eye(s) to monitor eye movements in the Monitoring Panel.

7.2.2. Display Control Configuration

When the GUI starts, the user is prompted to select an eye tracking system. If the user selects 'None', the GUI enters a 'non-vision research' mode which provides a Display Control Configuration Panel rather than an Eye Control Configuration Panel. Any incoming UDP packet with an identifier of '9' will be presented as a graphics object in the Monitoring Panel (see Section 8.3.1). This panel provides manual offset and gain controls for displaying graphics objects in the Monitoring Panel.

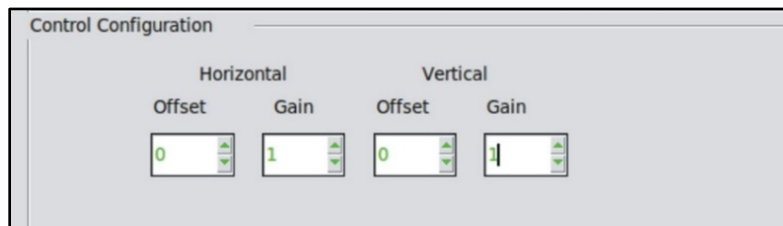


Figure 7.8: Display Control Configuration Panel

To disable the 'Display Control Configuration' Panel, find this section in 'gui.py' and comment out the highlighted lines below.

Code snippet from 'gui.py':

```
214 if globals.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globals.custom_button = []
216     globals.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_view_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.2.3. Manual Eye Calibration (Example Code)

The vision research mode of the GUI provides an 'Eye Calibration' panel for calibrating eye movements. This widget is paired with the MATLAB script 'Calibration.m' which is included as example code. After entering the 'Calibration.conf' task configuration file, and the subject configuration file, the Eye Calibration panel is ready. Select the "Eye Calibration" check box to control the calibration routine.

7.2.3.1. Enable/Disable Manual Calibration

To disable the 'Eye Calibration' panel, find this section in 'gui.py' and comment out the highlighted lines below.

Code snippet from 'gui.py':

```
214 if globals.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globals.custom_button = []
216     globals.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_view_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.2.3.2. Performing Manual Eye Calibration

Follow these steps to perform manual eye calibration.

- 1) Launch the 'Calibration.m' file on the MATLAB computer by typing the following into the command window:

```
Calibration(1)
```

- 2) Select the "Manual" check box to calibrate using the compass positions.

- 3) Once checked, click "Start" to begin the task. Select a compass position to display a fixation point at that location, and when the subject has fixated, hit "Accept".

- You can select "Clear" to delete the last accepted value. Dots are plotted on the Monitoring Panel to show the eye positions whenever "Accept" is pressed.

- 4) Repeat this at a minimum of five locations and then select the "Calibrate" button to calibrate.

- 5) To clear all accepted values, click the "Cancel" button.

- 6) Click "Save" in the Subject Name Panel to save the calibration.

- 7) Click "Stop" in the Task Control Panel to end the calibration.

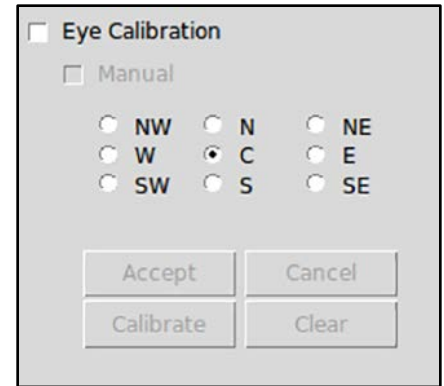


Figure 7.9: Eye Calibration Panel

7.2.4. Receptive Field Mapping Panel (Example Code)

The Receptive Field Mapping tool helps experimenters find the receptive field location of a neuron during electrophysiological recordings. It is paired with the MATLAB script 'Receptive_Field_Mapping.m'. Two mouse control options in the Monitoring Panel ('Feedback' and 'Dragging') allow the user to control the location of the stimulus in real-time. With Feedback, a stimulus will be presented at a location where mouse is clicked in a discrete step. To make the stimulus follow the mouse continuously, the dragging option must be enabled.

The panel provides common receptive field mapping stimuli, and allows the user to alter a variety of stimulus properties.

7.2.4.1. Enable/Disable Receptive Field Mapping

To disable the 'Receptive Field Mapping' panel, find this section in 'gui.py' and comment out the highlighted lines below.

Code snippet from 'gui.py':

```
214 if globals.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globals.custom_button = []
216     globals.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_view_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.2.4.2. Using the Receptive Field Mapping Panel

Follow these steps to run receptive field mapping.

- 1) After entering and loading the 'Receptive_Field_Mapping.conf' task configuration file and the subject configuration file, the Receptive Field Mapping panel is ready.
- 2) Run the 'Receptive_Field_Mapping.m' code. Note that the 'handmapping_default.mat' file also needs to be in the search path.

If connecting to a Scout Processor, use the following line in the MATLAB command window:

```
Receptive_Field_Mapping(1,1)
```

If not, use the following line in the MATLAB command window:

```
Receptive_Field_Mapping(1,0)
```

- 3) In the GUI, select the checkbox next to the Receptive Field Mapping panel title.
- 4) Select the stimulus type to be presented.
- 5) Manipulate the stimulus properties as desired.
- 6) Click "Submit" in the Receptive Field Mapping Panel, as well as in the Sending Panel.
- 7) Click "Start" to begin showing the stimulus. You may alter the stimulus parameters at any point but must click "Submit" in the Receptive Field Mapping Panel for the changes to take effect.
- 8) Select "Feedback" or "Dragging" in the Monitoring Panel to control the position of the stimulus.

☐ Receptive Field Mapping

☐ Grating

☐ Bar
☐ Dots

Spatial Freq.
Temporal Freq.

Bar Height (deg)
Bar Width (deg)
Bar Color (RGB)

Speed (deg/sec)
Density (dots/deg²)
Dot Size (deg)

General Stimulus Properties
Stimulus Position
Fixation Point

Orientation (deg)
Diameter (deg)
Contrast
Depth (mm)
Pulse Duration (sec)
Inter-Pulse Interval (sec)

X (deg): 0
Y (deg): 0

X (deg)
Y (deg)
Z (mm)

Figure 7.10: Receptive Field Mapping Control Panel

7.2.4.2.1. Grating Specific Properties

Displays a drifting black and white sinusoidal grating. Use the controls underneath this check box to alter the stimulus properties.

7.2.4.2.1.1. Spatial Frequency

Frequency of grating in cycles per degree.

7.2.4.2.1.2. Temporal Frequency

Cycles of the grating per second.

7.2.4.2.2. Bar Specific Properties

Displays a bar of the chosen height, width, and RGB color (scaled from 0 to 1).

7.2.4.2.3. Dot Specific Properties

Displays a random dot pattern.

7.2.4.2.3.1. Density

Number of dots per square degree.

7.2.4.2.3.2. Dot Size

Dot size (diameter) in degrees.

7.2.4.2.3.3. Speed

Dot speed, in degrees per second.

7.2.4.2.4. General Stimulus Properties

7.2.4.2.4.1. Diameter

Diameter of a circularly apertured grating or random dot stimulus, in degrees. Not enabled for the bar stimulus.

7.2.4.2.4.2. Depth

The depth of the stimulus relative to the screen (mm). The appropriate disparities are calculated using information in the Subject Configuration file (see Section 7.1.2). Using this parameter requires 3D presentation capabilities (e.g., shutter glasses, polarizers, or anaglyph glasses) and an enabled stereomode in 'Receptive_Field_Mapping.m'.

7.2.4.2.4.3. Orientation

Change the orientation of the bar, or direction that the dots or grating drifts. Units are in degrees, 0° to 360°, counter-clockwise with respect to vertical, with 0° corresponding to a rightward drift/orientation.

7.2.4.2.4.4. Contrast

Stimulus contrast (0%-100%).

7.2.4.2.4.5. Pulse Duration

When the inter-pulse interval is greater than zero, the stimulus will pulse on for the given duration in seconds.

7.2.4.2.4.6. Inter-Pulse Interval

Entering a value greater than zero initiates a pulsing stimulus and determines the length of time, in seconds, between stimulus presentations.

7.2.4.2.4.7. Fixation Point Position

The X (horizontal), Y (vertical), and Z (depth) locations of the fixation point. Altering the Z location requires 3D presentation capabilities (e.g., shutter glasses, polarizers, or anaglyph glasses) and an enabled stereomode in 'Receptive_Field_Mapping.m'.

7.2.5. User Programmable Buttons

When the GUI starts, the user is prompted to select an eye tracking system. If the user selects 'None', the GUI enters a 'non-vision research' mode which provides a User Programmable Buttons Panel rather than the Eye Calibration and Receptive Field Mapping Panels. This panel contains six command buttons which evoke 'click' events that execute user-defined callback functions.

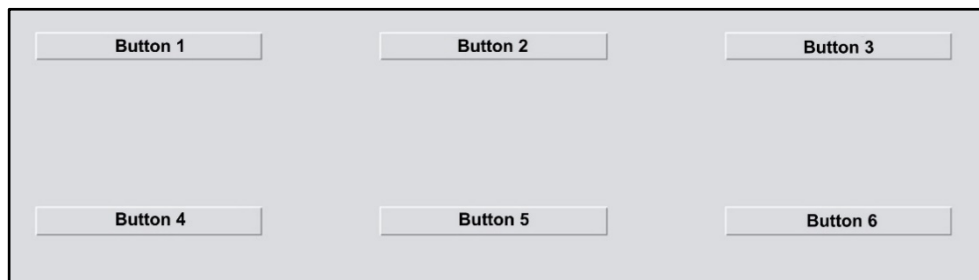


Figure 7.11: User Programmable Buttons Panel

7.2.5.1. Enable/Disable User Programmable Buttons Panel

To disable the User Programmable Buttons Panel, find this section in 'gui.py' and comment out the highlighted lines below.

```

214 if globals.eye_recording_method == EyeRecordingMethods.EYE_NONE:
215     globals.custom_button = []
216     globals.custom_button_callbacks = button_module()
217     tracking_config_dict = GuiWidget['tracker_window_config']['tracker_window_label']
218     self.draw_tracker_window_config()
219     self.draw_callback_buttons()
220 else:
221     self.draw_subject_view_params()
222     # Draw eye offset, gain, and configuration
223     self.draw_eye_and_window_config()
224     # Draw eye calibration widgets
225     self.draw_eye_calibration_task_button()
226     # Do this in the end as it initializes some of the widgets as well
227     self.init_calibration_configuration_parameter(init=True)
228     # Receptive field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']

```

7.2.5.2. Changing Button Labels

Labels for these buttons are defined in 'gui.py'. The labels can be changed by finding the following section of code in 'gui.py' and modifying the text highlighted below.

```

2653 'custom_buttons': {
2653     'custom_buttons': {
2654         'main_frame': {
2655             'relx': 0.01, 'rely': 0.65,
2656             'relheight': 0.34, 'relwidth': 0.48, 'width': 225},
2657         'buttons': [
2658             {'text': 'Start Save', 'state': tk.NORMAL,
2659              'relx': 0.1, 'rely': 0.2,
2660              'relheight': 0.07, 'relwidth': 0.20},
2661             {'text': 'Open Gate', 'state': tk.NORMAL,
2662              'relx': 0.4, 'rely': 0.2,
2663              'relheight': 0.07, 'relwidth': 0.20},
2664             {'text': 'E-Shock', 'state': tk.NORMAL,
2665              'relx': 0.7, 'rely': 0.2,
2666              'relheight': 0.07, 'relwidth': 0.20},
2667             {'text': 'Button 4', 'state': tk.NORMAL,
2668              'relx': 0.1, 'rely': 0.6,
2669              'relheight': 0.07, 'relwidth': 0.20},
2670             {'text': 'Button 5', 'state': tk.NORMAL,
2671              'relx': 0.4, 'rely': 0.6,
2672              'relheight': 0.07, 'relwidth': 0.20},
2673             {'text': 'Button 6', 'state': tk.NORMAL,
2674              'relx': 0.7, 'rely': 0.6,
2675              'relheight': 0.07, 'relwidth': 0.20},
2676         ]
2677     },

```


7.2.5.3. Adding Callback Functions to Programmable Buttons

Callback functions for the User Programmable Buttons are defined in 'button_funs.py'. Find this section of code and add user-defined callback function code here.

```
20 def button_1_callback(self):  
    '''Write your function here if it's simple. '''  
    '''If it is more complex, you might want to import it '''  
    '''from a separate document and call it here.'''
```

8. Writing Code for the GUI Counterpart

This section explains the basic coding structure for communicating with the GUI. It explains how to create a UDP connection and send information. This section also provides examples that highlight coding schemes and structures that can help you implement the REC-GUI framework.

The coding scheme for a counterpart CPU is shown using 'Fixation.m' as an example.

8.1. Basic coding structure.

Since MATLAB allows only one thread, we recommend using one while-loop with case-statements to control individual steps in the experimental control process. In the gaze fixation training example, the GUI has two UDP connections. One for reporting eye tracking results, and the other for sending/receiving commands from/to MATLAB. The MATLAB script thus establishes two UDP connections (MyUDP and MyUDP_eye).

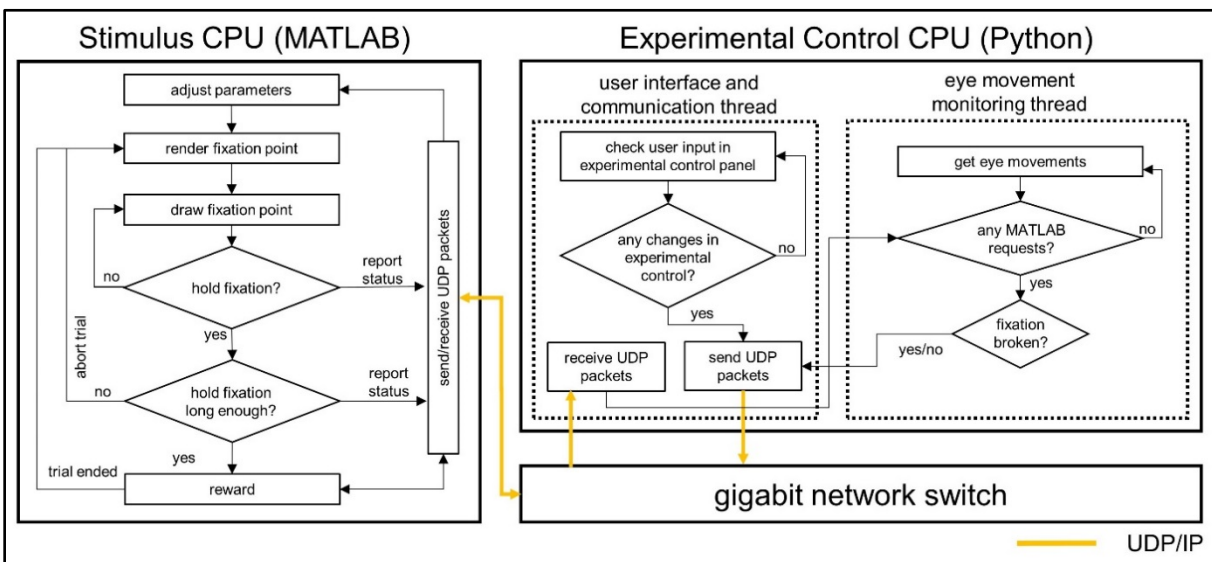


Figure 8.1: Experimental Coding Schematic

8.1.1. Coding Structure in MATLAB for a gaze fixation task.

Below is pseudo-code explaining how MATLAB can communicate with the GUI during real-time feedback. See 'Fixation.m' for the corresponding actual code.

```
FirstStep = 0;
while 1
    check version / vergence error from the GUI
    ex) see yellow highlighted lines in Section 8.1.3
    if ~FirstStep
        send parameters of fixation checking window to the GUI.
    ex) send '50 1 xPosFixation yPosFixation zPosFixation VergenceError LColor RColor' enable
    version/vergence windows
        ex) send '51 qqqq...qq/';
    send '53 qqqq...qq/';
        FirstStep = 1;
    end
    draw fixation point on the screen
    send request for checking version window to the GUI
    ex) send '4 qqqq...q/' to the GUI
    send request for checking vergence window to the GUI
    ex) send '5 qqqq...q/' to the GUI
    if version error or vergence error detected
        abort trial
    else
        continue...
    end
end
```

8.1.2. Establishing UDP connections between MATLAB and the GUI.

Below is a snippet of MATLAB code that establishes a UDP connection between MATLAB and the GUI. Check the provided MATLAB codes for example implementations.

```
packetSize = 1024; packetSize = 1024;
bufferLength = packetSize*1; % UDP packet size
%%% UDP connection sending/receiving commands between MATLAB and the GUI
Myudp = udp('MATLAB COMPUTER IP ADDRESS', 5001, 'LocalPort', 5002); % initiating UDP connection
if ~isempty(Myudp)
    set(Myudp,'ReadAsyncMode','continuous'); % set connection as asynchronous mode
    set(Myudp,'InputBufferSize',bufferLength*2); % prepare buffer for input
    set(Myudp,'OutputBufferSize',bufferLength*2); % prepare buffer for output
    set(Myudp,'DatagramTerminateMode','on'); % enabled using terminator in the packet
    fopen(Myudp); % open connection
    SendUDPGui(Myudp, '-1 8256'); %% send a probe packet to establish initial UDP connection
    readasync(Myudp); % start async. reading to control flow and check eye pos
end
%%% UDP connection dedicating to eye tracking results from the GUI
Myudp_eye = udp('MATLAB COMPUTER IP ADDRESS', 5003, 'LocalPort', 5004);
if ~isempty(Myudp_eye)
    set(Myudp_eye,'ReadAsyncMode','continuous');
    set(Myudp_eye,'InputBufferSize',bufferLength*2);
    set(Myudp_eye,'OutputBufferSize',bufferLength);
    set(Myudp_eye,'DatagramTerminateMode','on');
    fopen(Myudp_eye); % open connection
    readasync(Myudp_eye); % start async. reading to control flow and check eye pos
end
```

8.1.3. Main while-loop in MATLAB Script

Below is a code snippet illustrating how to read UDP packets from the GUI and how to use the identifiers in switch/case statements. This coding structure makes it easy to define identifiers and/or event codes that follow the experimental trial structure that you have created. The highlighted sections provide examples for using a switch/case statement after reading a UDP packet.

```

while 0
    if Myudp_eye.BytesAvailable >= packetSize
        UDP_Pack = char(fread(Myudp_eye,packetSize));
        flushinput(Myudp_eye);
        p=find(UDP_Pack=='q');    %% remove filler character
        if ~isempty(p)
            UDP_Pack(p)='';
        end
        tempIndex = strfind(UDP_Pack,'/');    %% remove terminator
        if ~isempty(tempIndex)
            for i=1:length(tempIndex)
                if i>1
                    tempStr = UDP_Pack(tempIndex(i-1)+1:tempIndex(i)-1);
                else
                    tempStr = UDP_Pack(1:tempIndex(i)-1);
                end
                [CMD, tempWord] = strtok(tempStr, ' ');
                CMD_Word = strrep(tempWord, ' ', '');
                switch CMD
                    case '-14'    %% check whether left eye position is in 'window' or not
                        IsLEyeIn = str2double(CMD_Word);
                    case '-15'    %% check whether right eye position is in 'window' or not
                        IsREyeIn = str2double(CMD_Word);
                    case '-16'    %% check whether vergence error violate threshold or not
                        IsVergenceIn = str2double(CMD_Word);
                end
            end
        end
    end
end
if Myudp.BytesAvailable >= packetSize    %% check UDP packet for eye tracking result.
    tempUDP = fread(Myudp,packetSize);
    flushinput(Myudp);
    for i=1:length(tempUDP)
        UDP_Pack(i) = char(tempUDP(i));
    end
    tempIndex = strfind(UDP_Pack,'/');
    if ~isempty(tempIndex)
        for i=1:length(tempIndex)
            if i>1
                tempStr = UDP_Pack(tempIndex(i-1)+1:tempIndex(i)-1);
            else
                tempStr = UDP_Pack(1:tempIndex(i)-1);
            end
            p=find(tempStr=='q');
            if ~isempty(p)
                tempStr(p)='';
            end
            [CMD, tempWord] = strtok(tempStr, ' ');
            CMD_Word = strrep(tempWord, ' ', '');
            tempI = str2double(CMD);
            switch tempI    %% check identifier from UPD packet
                case -2    %% from buttons in the task control panel
                    tempCMD = str2double(CMD_Word);
                    switch tempCMD
                        case 100    % start button
                            :

```

8.2. Network Configuration

Communication between devices/systems connected with the GUI CPU occurs over parallel networks so each hardware piece has a dedicated network switch. We use application 1 (Neural basis of 3D vision in non-human primates) from the REC-GUI manuscript as an example network configuration. Both the EyeLink and Scout Processor have a non-configurable, predefined subgroup of IP addresses. Two NICs are required for both the stimulus and experimental control systems, with each NIC assigned to a different subgroup of IP addresses: 100.1.1.* for the EyeLink and 192.168.42.* for the Scout Processor (**Figure 8.2**). With this configuration, the stimulus (MATLAB) and experimental control (GUI) CPUs can both communicate with the EyeLink and Scout Processor directly.

The 'Receptive_Field_Mapping.m' and 'Calibration.m' example codes assume the following IP configurations:

Network Switch #1 (*if connecting to the Scout Processor*):

Stimulus CPU IP (MATLAB): 192.168.42.130

Acquisition Computer IP: 192.168.42.129

Network Switch #2:

Experimental Control CPU IP (REC-GUI): 100.1.1.3

Stimulus CPU IP (MATLAB): 100.1.1.2

EyeLink Computer IP: 100.1.1.1

If you use a different IP configuration, you must change the MATLAB files accordingly. The 'Start_Coding.m' file can help direct you to the portions of code that would need to be changed.

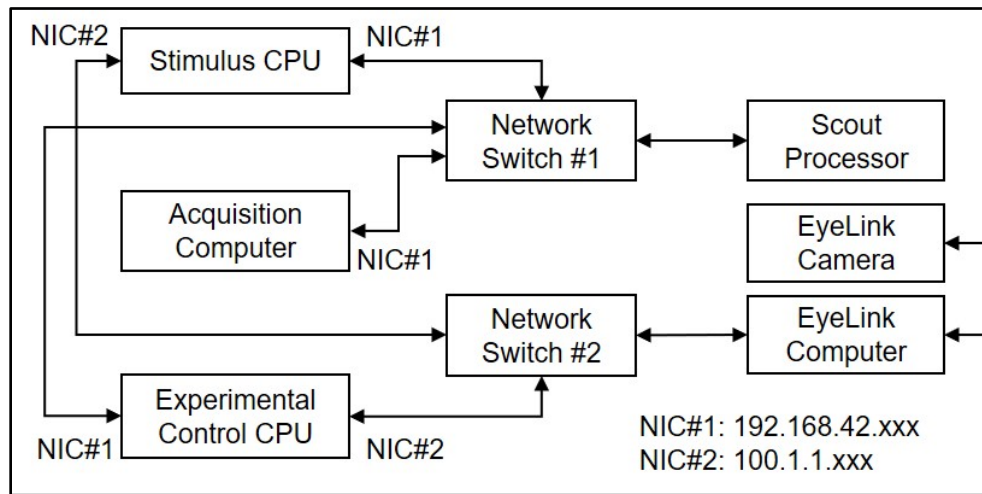


Figure 8.2: Network Communication Schematic

8.3. UDP Communication

The REC-GUI framework relies on a bidirectional communication stream to receive and send UDP packets of 1,024 byte character arrays. Each packet may have multiple commands unless it exceeds 1,024 bytes. Each command starts with an 'identifier' and is followed by the parameter 'value' and '/' (end of command). If the cumulative command strings are smaller than 1,024 bytes, fillers ('q') should be added to make a UDP packet of 1,024 bytes (see Section 8.3.1; **Table 8.1**).

8.3.1. Sending UDP Packets from MATLAB to the GUI

The MATLAB computer can send specific identifiers, specified using a leading integer (e.g., 4, 5, 50, 51, 53) to provide or request information from the GUI. For example, you can send a probe packet with the identifier '4' to request an evaluation of version or '5' for vergence. Whenever the GUI receives this type of packet, it evaluates the eye signals and reports the result by sending UDP packets back, which is described in the outgoing UDP packet (see Section 8.3.2; Table 8.2). Below are example predefined identifiers in the REC-GUI framework download.

Table 8.1: Example UDP Packet Information – MATLAB to GUI

Name	Identifier	Value Name	UDP Packet	UDP Size
Test UDP Connection	-1	N/A	'-1 8256 qqg...q'	
Request Eval. of Version Violation	4	N/A	'4 qqg...q'	1,024 Bytes
Request Eval. of Vergence Violation	5	1. Horizontal position 2. Vertical position 3. Depth of target 4. Vergence limits 5. Vergence option	'5 10 20 0 3 2 qqg...q'	1,024 Bytes
Event Code*	6	Event code	'6 111 qqg...q'	1,024 Bytes
Screen Width	7	Screen width (pixels)	'7 1280 qqg...q'	1,024 Bytes
Screen Height	8	Screen height (pixels)	'8 720 qqg...q'	1,024 Bytes
Display Graphics Objects	9	1. Object number 2. Horizontal position 3. Vertical position	'9 1 10 20 2 11 21 qqg...q'	1,024 Bytes
Version Window Parameters	50	1. Window number 2. Horizontal position 3. Vertical position 4. Depth of target 5. Vergence limits 6. Vergence option	'50 1 10 20 0 3 2 qqg...q'	1,024 Bytes
Enable Version Window	51	N/A	'51 qqg...q'	1,024 Bytes
Enable Vergence Window	53	N/A	'53 qqg...q'	1,024 Bytes

* Event codes are defined in the MATLAB code. For example, in the example above and in the vision-related example codes provided with the REC-GUI download, event code '111' corresponds to trial start. You can define your own event codes for your own experiments.

Below is an example function for sending UDP packets from MATLAB to the GUI.

```
function SendUDPGui(Myudp, tempStr)
    packetSize = 1024;
    SendStr(1:packetSize) = 'q';
    SendStr(1:length(tempStr)+1) = [tempStr '/'];
    fwrite(Dest, SendStr);    %% Send UDP packet
end
```

8.3.2. Sending UDP Packets from the GUI to MATLAB

As described in the Sending Panel section, the GUI can send custom parameters to the counterpart (e.g., MATLAB) computer. Each parameter should be prepared in the counterpart. Check the example codes provided for examples.

8.3.2.1. Predefined Outgoing UDP Packets

There are several predefined UDP packets that signal the status of the GUI, or the current experiment (e.g., Start, Stop, Pause and Exit). See **Table 8.2** for details.

Table 8.2: Example UDP Packet Information – GUI to MATLAB

Name	Identifier	Value	UDP Packet	UDP Size
Test UDP Connection	-1	8257	'-1 8257 qqg...q/'	1,024 Bytes
Start Button	-2	100	'-2 100 qqg...q/'	1,024 Bytes
Stop Button	-2	101	'-2 101 qqg...q/'	1,024 Bytes
Pause Button	-2	102	'-2 102 qqg...q/'	1,024 Bytes
Exit Button	-2	103	'-2 103 qqg...q/'	1,024 Bytes

9. Saved Data File Structure - GUI

The GUI can also work as a data acquisition system to store behavioral data, including eye movements from an eye tracking system, as well as event codes from the MATLAB/counterpart computer. Below is the file structure for the GUI data file. See Section 10 for example code showing how to read the GUI data file.

The following information can also be found in file “data_collection.py”. Use the data structure to save the GUI data as a binary file.

9.1. Header Structure

```
HEADER
-----
* DATA_HEADER_SIZE = 500B
* Data is stored in plain text
* Data in the data file:

    Subject ID: <>
    Date: <>
    Time: <>
    Configuration File: <>
```

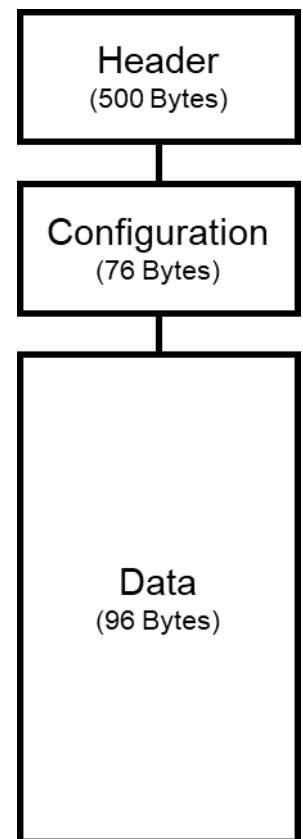


Figure 9.1: GUI Data File Structure

9.2. Configuration Structure

CONFIGURATION

```
-----
* DATA_CONFIGURATION_SIZE = 76
* Config Data is stored in a specified format and that is
* Data in the data file:

Number of records - (int 4B)
Fixation window acquire time - (float- 4B)
Fixation window hold time - (float - 4B)
Target window hold time - (float - 4b)
Inter trial stimulus interval - (float - 4B)
Inter stimulus interval - (float - 4B)
Missed trial interval - (float - 4B)
Response hold time - (int - 4B)
Stimulus duration time - (int - 4B)
Fixation window size - (float - 4B)
Target window size - (float - 4B)
Vergence size - (float - 4B)
X Left eye offset - (short - 2B)
X Left eye gain - (short - 2B)
Y Left eye offset - (short - 2B)
Y Left eye gain - (short - 2B)
X Right eye offset - (short - 2B)
X Right eye gain - (short - 2B)
Y Right eye offset - (short - 2B)
Y Right eye gain - (short - 2B)
Left pupil size - (int - 4B)
Right pupil size - (int - 4B)
Eye Coil channel selection - (int 4B)
```

9.3. Data Structure

DATA

```
-----
* DATA_SIZE = 96
* DATA_MATLAB_SIZE = 64
* What Data:
  Timestamp generated by UI - (double 8B)
  Right eye X - (int - 4B)
  Right eye Y - (int - 4B)
  Right eye Z - (int - 4B)
  Left eye X - (int - 4B)
  Left eye Y - (int - 4B)
  Left eye Z - (int - 4B)
  MATLAB data - (str - 64B)
```


10. Reading Saved REC-GUI Data: Example Code for Reading Data Files

The MATLAB file 'ReadingGUIData.m' contains example code showing how to read the data saved by the GUI. This code can be used as a template for your own experiments.

11. Basic Coding Structure in MATLAB

The MATLAB files 'BasicCodingStructure.m' and 'start_coding_closedloop.m' contain example code which show how to read UDP packets and implement experimental flow. 'BasicCodingStructure.m' implements a visual experiment using the Psychophysics Toolbox, and 'start_coding_closedloop.m' is a simple code that requires no additional hardware or software. These codes can be used as templates for implementing your own experiments.

12. Phototransistor Circuit

The phototransistor circuit shown below can be used to track visual stimuli on a screen.

12.1. Parts

Photo-transistor: PT334-6C

Schmitt trigger inverter: SN74HC14

12.2. Circuit Diagram

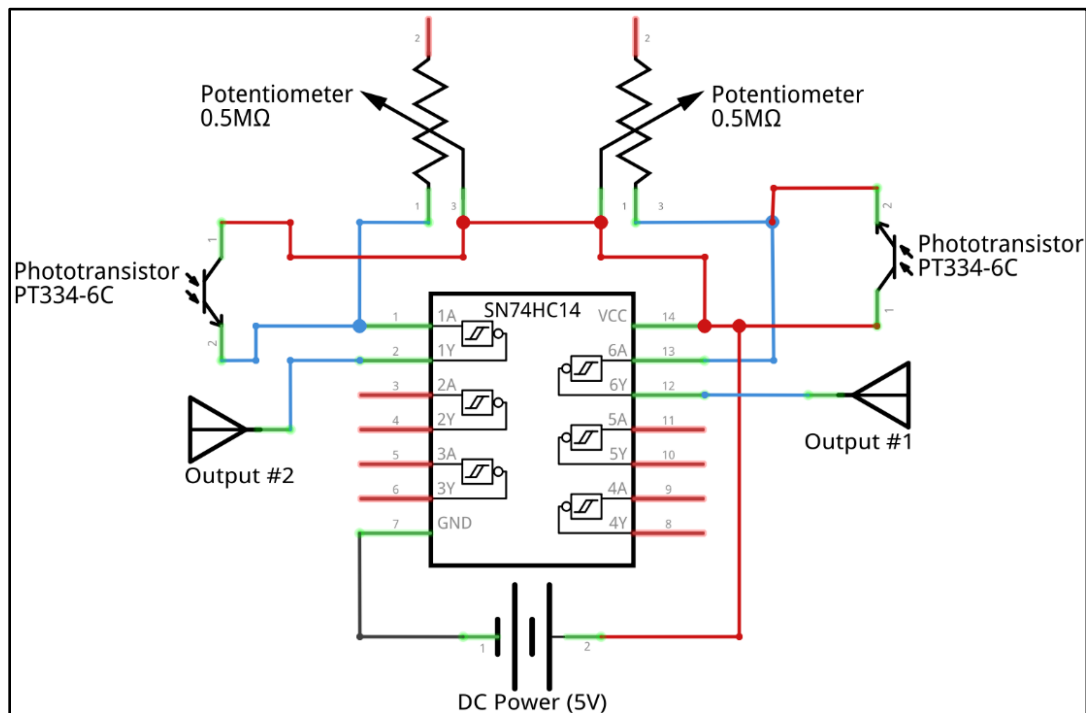


Figure 12.1: Phototransistor Circuit Diagram