

REAL-TIME EXPERIMENTAL CONTROL
WITH GRAPHICAL USER INTERFACE
(REC-GUI)

TECHNICAL USER MANUAL

VERSION 3.0.0 (06/15/2019)

TABLE OF CONTENTS

1	Introduction.....	5
2	Contact	5
3	Citation	5
4	The REC-GUI Framework.....	6
5	Installation & 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	9
5.2.3	Example MATLAB Scripts (Useful Learning Tools).....	9
6	Opening the GUI	10
6.1	Parameters in <i>Default.conf</i>	10
7	Using the GUI: Functions & General Instructions	11
7.1	Non-Configurable Widgets.....	12
7.1.1	Subject Name Panel: Creating & Loading Subject Configuration	12
7.1.2	Subject Configuration Panel.....	13
7.1.3	Communication Panels between the GUI & its Counterparts.....	13
7.1.3.1	Sending Panel	13
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.....	14
7.1.3.2	Receiving Panel	14
7.1.3.2.1	Variable Name	14
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	15
7.1.3.3.2	Start, Stop, Pause, & Exit.....	15
7.1.3.4	Monitoring Panel	16
7.1.3.4.1	Feedback Checkbox.....	16
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.....	17
7.2.1.2	Eye Checkboxes.....	18
7.2.2	Manual Eye Calibration (Example Code)	18
7.2.2.1	Performing Manual Eye Calibration.....	18
7.2.2.2	Enable/Disable Manual Calibration.....	18
7.2.3	Receptive Field Mapping Panel (Example Code)	19
7.2.3.1	Using the Receptive Field Mapping Panel.....	19
7.2.3.1.1	Grating Specific Properties.....	19
7.2.3.1.1.1	Spatial Frequency.....	20
7.2.3.1.1.2	Temporal Frequency.....	20
7.2.3.1.2	Bar Specific Properties	20
7.2.3.1.3	Dot Specific Properties	20
7.2.3.1.3.1	Speed	20
7.2.3.1.3.2	Density	20
7.2.3.1.3.3	Dot Size.....	20
7.2.3.1.4	General Stimulus Properties.....	20
7.2.3.1.4.1	Orientation	20
7.2.3.1.4.2	Diameter	20
7.2.3.1.4.3	Contrast.....	20
7.2.3.1.4.4	Depth.....	20
7.2.3.1.4.5	Pulse Duration	20
7.2.3.1.4.6	Inter-Pulse Interval.....	21
7.2.3.1.5	Fixation Point	21
7.2.3.2	Enable/Disable Receptive Field Mapping.....	21
7.2.4	Display Control Configuration	21
7.2.5	User Programmable Buttons.....	22
7.2.5.1	Changing Button Labels	22
7.2.5.2	Adding Callback Functions to Programmable Buttons.....	23
7.2.5.3	Enable/Disable the User Programmable Buttons Panel.....	23
8	Writing Code for the GUI Counterpart.....	24
8.1	Basic Coding Structure	24
8.1.1	Main while-loop in MATLAB Script.....	24
8.2	Network Configuration	26
8.3	UDP Communication.....	27
8.3.1	Sending UDP Packets from MATLAB to the GUI.....	27
8.3.2	Sending UDP Packets from the GUI to MATLAB.....	28

8.3.2.1	Predefined Outgoing UDP Packets.....	28
9	Saved Data File Structure – GUI.....	28
9.1	Header Structure.....	29
9.2	Configuration Structure.....	29
9.3	Data Structure	30
10	Reading Saved REC-GUI Data: Example Code for Reading Data Files	30
11	Basic Coding Structure in MATLAB.....	30

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.

IMPORTANT:

Depending on your experimental needs, additional resources may be required. This document describes example experimental procedures and 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 for your application. The REC-GUI online forum is a valuable resource for customization: <https://recgui2018.wixsite.com/rec-gui/forum>

2 CONTACT

The REC-GUI framework is an open-source project, and we encourage you to contribute to its continued development. To contribute, please contact us via the forum or at any of the following websites/emails.

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>

REC-GUI YouTube: <https://www.youtube.com/channel/UCWqf4wJ5WSkHNhsvKUcmt0A>

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, as well as video tutorials for running the REC-GUI framework and example codes on your computer.

You can directly reach Byoungsoon Kim (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 (2019). Real-Time Experimental Control using Network-Based Parallel Processing. eLIFE, 8:e40231.

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 demand 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) 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 languages and operating systems to maximize experimental flexibility and minimize researcher effort, (ii) simple interfacing to connect multiple measurement and recording devices, (iii) high temporal fidelity by dividing tasks across CPUs, and (iv) real-time control using a customizable and intuitive GUI.

Experimental systems often require multiple pieces of specialized hardware produced by different companies (e.g., commercial devices 1 and 2 in **Figure 5.1.1**). Network-based communications with such hardware often must occur over non-configurable, predefined subgroups of IP addresses. In **Figure 5.1.1**, commercial devices 1 and 2 have such 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 CPU can both directly communicate with commercial devices 1 and 2.

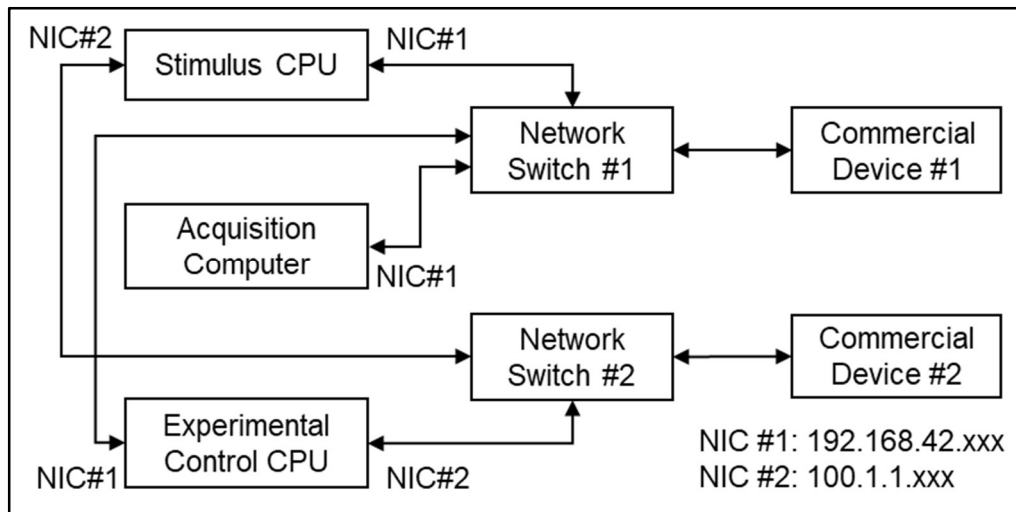


Figure 5.1.1 Network Communication Schematic

5.1 REQUIRED PACKAGES

Main code for the REC-GUI framework:

- Download/clone 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 (<https://www.numpy.org>)
- Tkinter: for building the GUI (<https://tkdocs.com/tutorial/install.html>)
 - Tkinter programming reference: <https://tkdocs.com/tutorial/firstexample.html>
- MATLAB (2017a or greater is recommended)
 - MATLAB's Computer Vision System Toolbox
 - MATLAB's Data Acquisition Toolbox (if using USB-1680G or I/O)
 - Psychtoolbox 3 for visual stimulus generation (<https://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 for updates and usage. Skipping required packages can cause serious system errors.

Selected packages for specific system configurations:

- To use EyeLink (SR Research, Inc.)
 - EyeLink is an eye tracking software that can be used for vision research.
 - 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>

- To use USB-1680G (Measurement Computing, Inc.)
 - USB-1680G is a multifunctional DAQ used to handle analog or digital I/O.
 - **Linux**
 - Ready to use with REC-GUI: <https://pypi.org/project/pydaqflex/>
 - <https://github.com/mccdaq/uldaq>
 - **Windows**
 - <https://github.com/mccdaq/mcculw>
- To use a PC Parallel Port
 - This is a simple solution for digital I/O
 - **Linux**
 - Ready to 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 codes 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 about the framework, but also help you start with your experiments.

5.2.1 CONFIGURATION FILES

The following list of configuration files may be used with the GUI. Files that are required to properly run the system are marked in **bold**, whereas the remaining are example files that may be run for specific experiments.

- ***default.conf***: required default system configuration file (see [Section 6](#))
- ***<subject name>.conf***: required subject configuration files (see [Section 6](#))
- ***stim_default.m***: required MATLAB configuration file that contains default stimulus parameters. It is tailored for stimuli rendered in MATLAB
- *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 7.2.2](#))
- *receptive_field_mapping.conf*: an example system configuration file for implementing a receptive field mapping task
 - To be using in conjunction with *Receptive_Field_Mapping.m* (see [Section 7.2.3](#))

5.2.2 PYTHON FILES

The following files are required for the GUI. Essential classes that need to be modified 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
- *buttons_funs.py*: a class that defines call-back functions for the user programmable buttons (see [Section 7.2.3](#))
- *calibration.py*: a class for handling eye calibration procedures
- ***constants.py***: a class that defines all system constants
- *data_collection.py*: a class for handling eye calibration procedures
- *digitalIO.py*: a class for input/output digital bits to control TTL pulses using USB-1680G
- ***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 output data file
- ***global_parameters.py***: a class that defines all system and configuration variables
- ***gui.py***: a class for building the GUI (in Tkinter).
- *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 pop-up window for selecting the eye tracking method
- *staircase.py*: a class for implementing a staircase procedure in a psychophysical task
- *test.py*: a class containing example subject file parameters to test the GUI setup
- *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 (e.g., 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.

- *Fixation.m*: Example script that implements gaze fixation training using standard operant conditioning procedures. This code presents visual targets 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 a manual eye movement calibration procedure using the GUI. The user manually selects the location of a fixation target and indicated 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 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., for electrophysiological recordings).
 - Use *Receptive_Field_Mapping.conf* to configure the GUI to run this script.
- *start_coding_closedloop.m*: Script that shows how to establish interactive control between the GUI and MATLAB. The MATLAB CPU 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. See the REC-GUI Tutorial Guide for more information on how to get started with your experiment.
 - Use *start_coding_closedloop.conf* to configure the GUI to run this script
- *start_coding.m*: Script that provides a template for preparing a new task, containing code for initializing a UDP connection and a while-loop for experiment specific code.
 - 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 into the command terminal:

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

From the Task Control Panel (see [Section 7.1.3.3](#)), load *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 the included files, be sure to preserve the structure of the code, including parentheses. Altering the structure of a file may result in serious system failures. We encourage you to edit the files but do so with caution.

6.1 PARAMETERS IN DEFAULT.CONF

Parameters for specialized tools that are automatically loaded are defined here. Such parameters can be added or removed as necessary.

- *arbitrator_service*: defines IP addresses and UDP port numbers for CPUs (e.g., stimulus CPU) that communicate with the experimental control CPU. UDP packets are broadcast to a specific IP address with a specific port number.
 - Default: 5001 through 5004 are default port numbers
 - These may need to be adjusted for your CPU and should be open in the firewall.
 - See the REC-GUI Tutorial Guide, as well as the sample code tutorial video *REC-GUI Tutorial #1 – Setting up the network switch* on the REC-GUI YouTube Channel.

- `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., MATLAB).
- `server_port`: port number of the counterpart (e.g., MATLAB). This number should match the local port number in the GUI code.
- `local_port_eye`: port number of the experimental control CPU. This number should match the server port number in the EyeLink system.
- `server_port_eye`: port number in the EyeLink system
- `server_ip`: IP address of the counterpart CPU (e.g., MATLAB)
- `eyelink_service`: IP address of the EyeLink CPU
- `constant`: default system parameters
- `number_of_task_config_entries`: defines how many lines are needed in the Sending Panel
 - Default: 30
- `moving_eye_position_mean`: defines the number of eye movement samples averaged per eye position estimate
 - Default: 25
- `data_size_length`: UDP packet size
 - Default: 1,024 bytes
- `data_padding_character`: filler character for UDP packets
 - Default: q
- `message_delimiter`: delimiter for UDP packets
 - Default: /

7 USING THE GUI: FUNCTIONS & GENERAL INSTRUCTIONS

The GUI is at the core of the REC-GUI framework. 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.

The default configuration of the GUI has two operation 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 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. User can reconfigure these buttons and add callback functions as needed. New user-defined configurations can 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 placed below (the “if” case is the non-vision mode; 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*:

```
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_viewring_params()
223     # Draw eye offset, gain, 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 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 are widget that support key system functions. Editing these codes can result in serious system failures, unless otherwise noted.

7.1.1 SUBJECT NAME PANEL: CREATING & 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 7.1.2](#) “Subject Configuration File” for details). This will generate a subject configuration file that can be loaded. The subject name is used as part of the filename 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). The subject configuration file can contain any information that needs to be broadcast to the counterpart CPU (e.g., MATLAB) with the appropriate configurations.

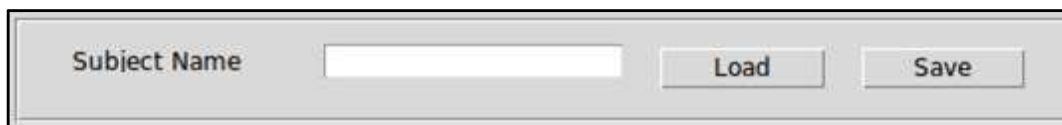
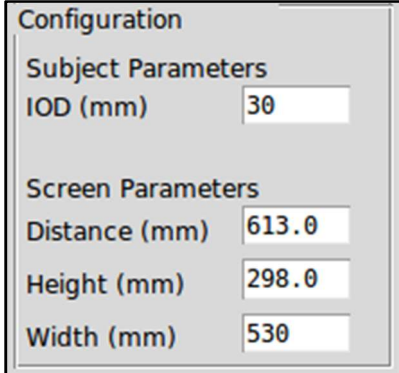


Figure 7.1.1 Subject Name Control Panel

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.

To disable the Eye Control Configuration Panel, find this section in *gui.py* and comment out the highlighted line:



The screenshot shows a window titled "Configuration". Inside, there are two sections: "Subject Parameters" and "Screen Parameters". Under "Subject Parameters", there is a label "IOD (mm)" and a text input field containing the value "30". Under "Screen Parameters", there are three labels: "Distance (mm)" with a value of "613.0", "Height (mm)" with a value of "298.0", and "Width (mm)" with a value of "530". Each label is followed by a text input field.

Figure 7.1.2 Subject Configuration Panel

Code snippet from *gui.py*:

```
220 else:
221     self.draw_subject_viewing_params()
222     # Draw eye offset, gain, 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']
```

IMPORTANT:

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 (e.g., MATLAB). Note that clicking “Save” will alter the loaded subject configuration file. If you do not wish to overwrite this file, enter a new name and then save the file.

7.1.3 COMMUNICATION PANELS BETWEEN THE GUI & 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 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 CPU (e.g., MATLAB) 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 (e.g., MATLAB). The sending panel has three columns of user input text boxes and a “Submit” button, as described in the following sections.

Variable Name	Identifier	Value
StimulusDuration	-106	1
InterTrialInterval	-101	1
FixationWindowHoldTi	-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 7.1.3 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 (e.g., MATLAB) as the identity of a variable. See the example MATLAB scripts on the provided GitHub page, and [Table 8.3.1](#) in [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 (e.g., MATLAB). See the example MATLAB scripts on GitHub and [Table 8.3.1](#) 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.

7.1.3.2 RECEIVING PANEL

The GUI provides a receiving panel to monitor information from the counterpart (e.g., MATLAB) in real-time. Whenever counterparts (e.g., MATLAB) 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 name assigned to the variable in the Sending Panel.

7.1.3.2.2 IDENTIFIER

An integer value uniquely associated with the corresponding variable. This identifier is used by the counterpart (e.g., MATLAB) as signal to send a variable. See the example MATLAB scripts and [Table 8.3.1](#) 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 (e.g., MATLAB). See the example MATLAB scripts on GitHub and [Table 8.3.1](#) for examples of how to send the GUI these values from MATLAB using the Identifiers.

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. As an example, input *Fixation.conf* for the gaze fixation task included with the REC-GUI download.

7.1.3.3.1 CREATING A TASK CONFIGURATION FILE

In the Task Control Panel, 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. This file can now be loaded.

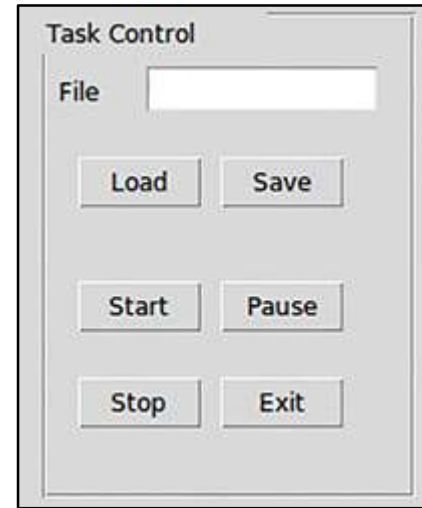


Figure 7.1.4 Task Control Panel

IMPORTANT:

To begin a task, you must load a subject configuration file before starting the experiment script on the counterpart (e.g., MATLAB). After loading the subject configuration, you can launch the experiment on the counterpart and hit the “Start” button in the Task Control Panel.

7.1.3.3.2 START, STOP, PAUSE, & EXIT

Once the Task Configuration is loaded, you can send start, stop, pause, or exit commands to the counterpart CPU(s) (e.g., MATLAB) by pressing the appropriate button(s). See [Section 8](#) for examples of how to handle these events in the counterpart code. The default packet structure of the information sent when these buttons are pressed is defined in *constants.py*, as follows, but may be adjusted as desired:

- Start: begins the experiment after launching it on the counterpart (e.g., MATLAB)
 - Identifier: -2; Value: 100
 - GUI sends the UDP Packet: “-2 100 qq q/” (total 1,024 bytes)
- Pause: pauses the experiment until Start is selected again
 - Identifier: -2; Value: 101
 - GUI sends the UDP Packet: “-2 101 qq q/” (total 1,024 bytes)

- Stop: ends the experiment and clears the Subject and Task Control Panels but leaves the GUI open
 - Identifier: -2; Value: 102
 - GUI sends the UDP Packet: “-2 102 qq q...” (total 1,024 bytes)
- Exit: closes the GUI
 - Identifier: -2; Value: 104
 - GUI sends the UDP Packet: “-2 104 qq q...” (total 1,024 bytes)

7.1.3.4 MONITORING PANEL

The real-time display of continuous signals occurs in the Monitoring Panel. In vision-research mode, the Monitoring Panel shows eye movement signals provided by EyeLink or a search coil system. In non-vision research mode, the Monitoring Panel displays signals received with a default Identifier of “-6” (see [Section 8.3](#)).

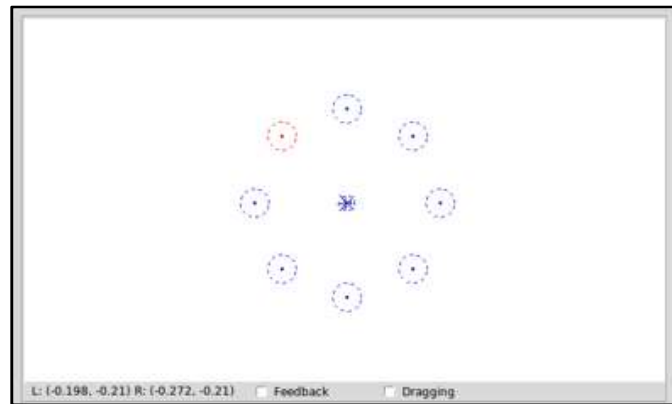


Figure 7.1.5 Monitoring Panel

The Monitoring Panel can also display behavioral enforcement “windows.” For example, you can monitor if a saccade was made to a specific target, or if a fixation window was violated in vision research. An example (application 1 in the REC-GUI manuscript) is shown in **Figure 7.1.5**, with 7 distractor windows (outer blue circles), a correct target window (red), version fixation window (center, blue), left and right eye positions (crosses in 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 (application 3 in the manuscript).

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

7.1.3.4.1 FEEDBACK CHECKBOX

If the Feedback checkbox is selected, when a position in the Monitoring Panel is clicked using the GUI mouse, the position is sent to an established counterpart CPU (e.g., MATLAB) at the time 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.

If the Dragging checkbox is selected, the position of the mouse cursor within the Monitoring Panel is continuously sent to an established counterpart CPU (e.g., MATLAB). 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

For each of the two operating modes, the GUI presents configurable widgets. For vision research, it provides default configurations for the three panels under the Monitoring Panel: Eye Control Configuration, Manual Eye Calibration, and Receptive Field Mapping. For non-vision research, the GUI presents two panels: Display Control Configuration and User Programmable Buttons.

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, Offset should be initially set to 0, Gain to 1, and Pupil Size to 1.

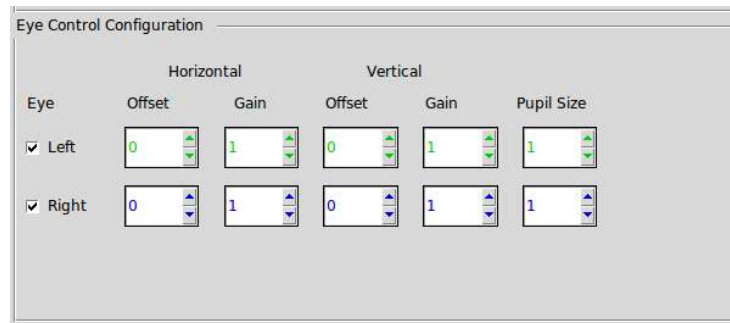


Figure 7.2.1 Eye Control Configuration Panel

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 line below.

Code snippet from *gui.py*:

```

220 else:
221     self.draw_subject_viewimg_params()
222     # Draw eye offset, gain, 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 MANUAL EYE CALIBRATION (EXAMPLE CODE)

The vision research mode of the GUI provides an Eye Calibration Panel for calibrating eye movements. This panel is paired with the MATLAB code *Calibration.m* which is found on GitHub. After entering the *Calibration.conf* task configuration file, and the subject configuration file, the Eye Calibration Panel is ready. Select the “Eye Calibration” checkbox to control the calibration routine.

7.2.2.1 PERFORMING MANUAL EYE CALIBRATION

Follow these steps to perform manual eye calibration:

1. Launch *Calibration.m* on the MATLAB CPU by typing the following into the command window:

```
Calibration(1)
```

2. Select “Manual” to calibrate using the compass
3. Click “Start” to begin
4. Select a compass position to display a fixation point at that location. Hit “Accept” after fixation.
 - a. Dots are plotted on the Monitoring Panel to show the eye positions when “Accept” is pressed
 - b. Select “Clear” to delete the last accepted value
5. Repeat this at a minimum of five compass positions and then select “Calibrate”
 - a. To clear all accepted values, click “Cancel”
6. Click “Save” in the Subject Name Panel to save the calibration
7. Click “Stop” in the Task Control Panel to end the calibration

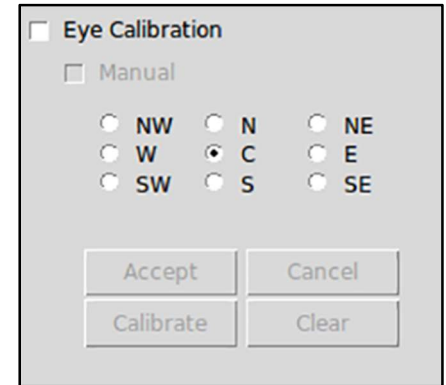


Figure 7.2.2 Eye Calibration Panel

7.2.2.2 ENABLE/DISABLE MANUAL CALIBRATION

To disable the Eye Calibration Panel, comment out the highlighted lines below in *gui.py*:

Code snippet from *gui.py*:

```
220 else:
221     self.draw_subject_viewing_params()
222     # Draw eye offset, gain, 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 RECEPTIVE FIELD MAPPING PANEL (EXAMPLE CODE)

The Receptive Field Mapping Panel helps experimenters find the receptive field location of a neuron during electrophysiological recordings by providing common receptive field mapping stimuli and allowing the user to alter a variety of stimulus properties. It is paired with 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. See [Sections 7.1.3.4.1](#) and [7.1.3.4.2](#) for details on these mouse control options.

7.2.3.1 USING THE RECEPTIVE FIELD MAPPING PANEL

Follow these steps to run receptive field mapping:

1. Load the *Receptive_Field_Mapping.conf* task configuration file in the Task Control Panel, and the subject configuration file in the Subject Name Control Panel
2. Run the *Receptive_Field_Mapping.m* code.
 - a. **Note:** *handmapping_default.m* also must be in the search path
 - b. If connecting to a Scout Processor, use the following line in the MATLAB command window:

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

- c. 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 (see [Section 7.2.3.1](#))
6. Click “Submit” in the Receptive Field Mapping Panel and the Sending Panel
7. Click “Start” to begin showing the stimulus.
 - a. You may alter the stimulus parameters at any point but must click “Submit” in the Receptive Field Mapping Panel for the changed 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. Bar Height (deg) Speed (deg/sec)
Temporal Freq. Bar Width (deg) Density (dots/deg²)
Bar Color (RGB) Dot Size (deg)

General Stimulus Properties Stimulus Position Fixation Point

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

Figure 7.2.3 Receptive Field Mapping Control Panel

7.2.3.1.1 GRATING SPECIFIC PROPERTIES

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

7.2.3.1.1.1 SPATIAL FREQUENCY

Frequency of grating (in cycles per degree).

7.2.3.1.1.2 TEMPORAL FREQUENCY

Cycles of grating per second.

7.2.3.1.2 BAR SPECIFIC PROPERTIES

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

7.2.3.1.3 DOT SPECIFIC PROPERTIES

Displays a random dot pattern.

7.2.3.1.3.1 SPEED

Dot speed (in degrees per second).

7.2.3.1.3.2 DENSITY

Number of dots per square inch.

7.2.3.1.3.3 DOT SIZE

Dot diameter (in degrees).

7.2.3.1.4 GENERAL STIMULUS PROPERTIES

7.2.3.1.4.1 ORIENTATION

Change the orientation of the stimulus (e.g., the bar, grating, or direction of the drifting dots). Orientation value is in degrees, with 0° corresponding to a rightward drift/orientation.

7.2.3.1.4.2 DIAMETER

Diameter of a circular aperture grating or random dot stimulus (in degrees).
Not enabled for the bar stimulus.

7.2.3.1.4.3 CONTRAST

Stimulus contrast (0 – 100%).

7.2.3.1.4.4 DEPTH

The depth of the stimulus relative to the screen (in 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 stereomode enabled in *Receptive_Field_Mapping.m*.

7.2.3.1.4.5 PULSE DURATION

“On” pulse duration (in seconds) when the inter-pulse interval is greater than zero.

7.2.3.1.4.6 INTER-PULSE INTERVAL

Length of time (in seconds) between stimulus presentations when the entered value is greater than zero.

7.2.3.1.5 FIXATION POINT

The X (horizontal), Y (vertical), and Z (depth) locations of the fixation point. Altering the Z location requires 3D presentation capabilities and an enabled stereomode in

Receptive_Field_Mapping.m.

7.2.3.2 ENABLE/DISABLE RECEPTIVE FIELD MAPPING

To disable the Receptive Field Mapping Panel, find this section in *gui.py* and comment out the highlighted line below.

Code snippet from *gui.py*:

```
220 else:
221     self.draw_subject_viewing_params()
222     # Draw eye offset, gain, 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     # receptve field mapping
229     self.draw_receptive_field_mapping_window()
230     tracking_config_dict = GuiWidget['eye_window_config']['eye_window_label']
```

7.2.4 DISPLAY CONTROL CONFIGURATION

In non-vision research mode, the GUI provides a Display Control Configuration Panel rather than an Eye Control Configuration Panel. Any incoming UDP packet with a default Identifier of “-6” will be presented as a graphics object in the Monitoring Panel (see [Section 8.3.1](#)).

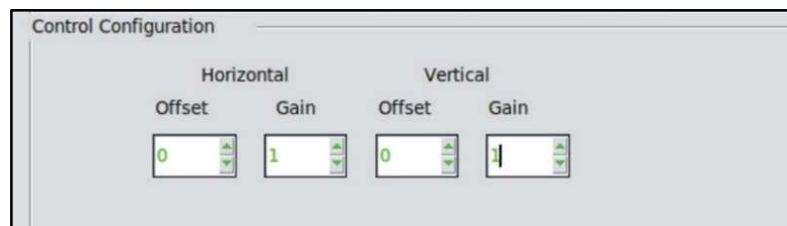


Figure 7.2.4 Display Control Configuration Panel

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

Code snippet from *gui.py*:

```
220 else:
221     self.draw_subject_viewing_params()
222     # Draw eye offset, gain, 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 USER PROGRAMMABLE BUTTONS

When in “non-vision research” mode, the GUI 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.

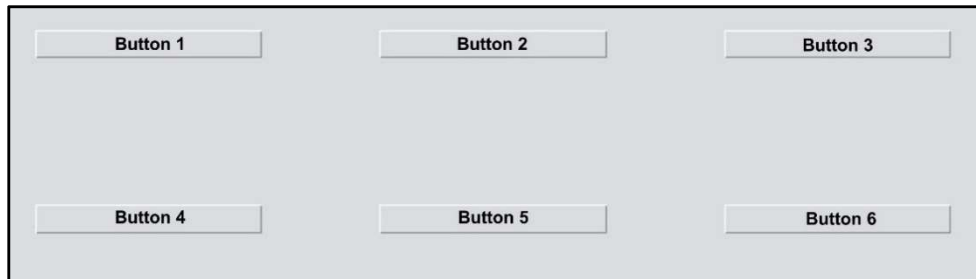


Figure 7.2.5 User Programmable Buttons Panel

7.2.5.1 CHANGING BUTTON LABELS

Labels for these buttons can be changed by modifying the highlighted text in the section of *gui.py* below.

Code snippet from *gui.py*:

```
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     },
```

7.2.5.2 ADDING CALLBACK FUNCTIONS TO PROGRAMMABLE BUTTONS

Callback functions for the User Programmable Buttons can be added in this section of *button_funs.py*.

Code snippet from *button_funs.py*:

```
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.'''
```

7.2.5.3 ENABLE/DISABLE THE USER PROGRAMMABLE BUTTONS PANEL

To disable the User Programmable Buttons Panel, find this section in *gui.py* and comment out the highlighted line 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()
```

This section explains the basic coding structure for creating a UDP connection to send information and communicate with the GUI. This section also provides examples that highlight coding schemes and structures that can help with implementing the REC-GUI framework.

The coding scheme for a counterpart CPU (e.g., MATLAB) 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 another for sending/receiving commands to/from MATLAB. The MATLAB script thus establishes two UDP connections (MyUDP and MyUDP_eye).

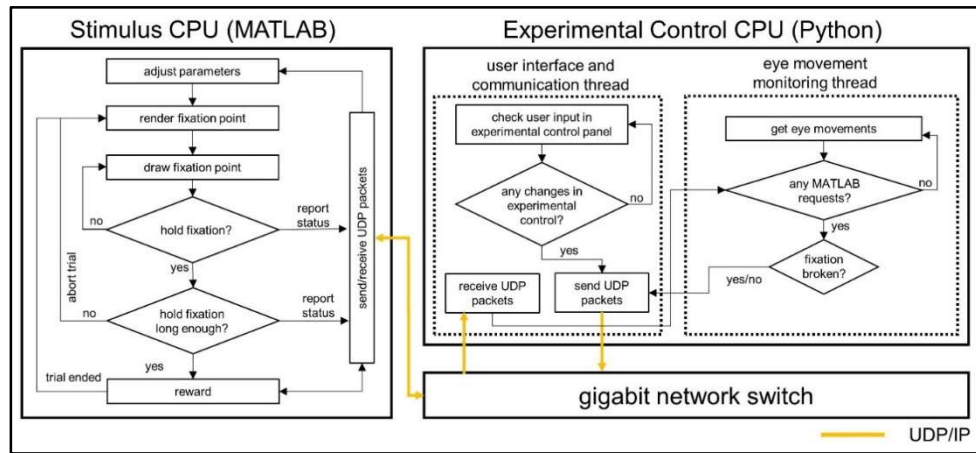


Figure 8.1.1 Experimental Coding Schematic

8.1.1 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. For more examples on how to use the REC-GUI framework, see the REC-GUI Tutorial Guide, as well as the sample code tutorial videos on the REC-GUI website.

Code snippet from :

```
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
                            :
                        :
                    end
                end
            end
        end
    end
end
```

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 NICE assigned to a different subgroup of IP addresses: 100.1.1.* for the EyeLink and 192.168.42.* for the Scout Processor (**Figure 8.2.1**). With this configuration, the stimulus (MATLAB) and experimental control (GUI) CPUs can both communicate with the EyeLink and Scout Processor directly.

The *Receptive_Field_Mappings.m* and *Calibration.m* example codes on GitHub assume the following IP configurations:

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

Stimulus CPU IP (MATLAB): 192.168.42.130

Acquisition CPU 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 CPU 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.

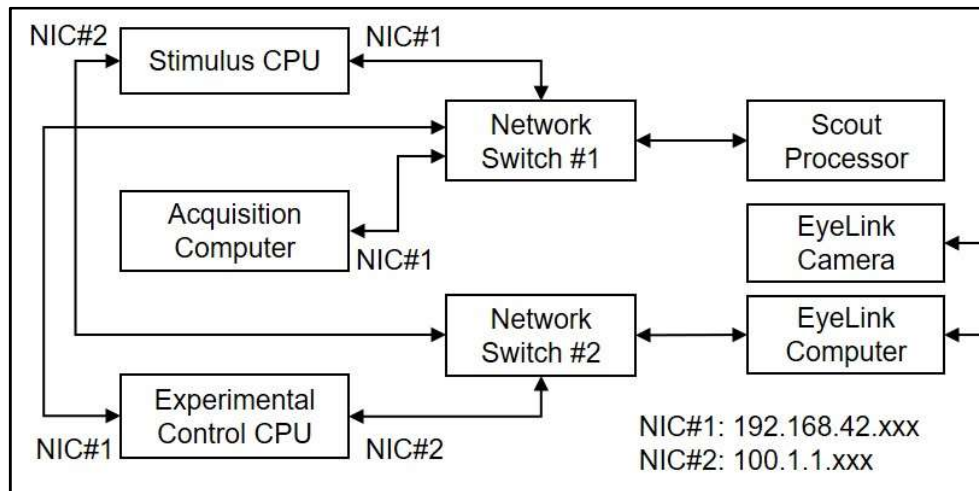


Figure 8.2.1 Network Communication Schematic

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 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 examples below in [Section 8.3.1](#)).

8.3.1 SENDING UDP PACKETS FROM MATLAB TO THE GUI

The MATLAB CPU can send specific Identifiers, specified using a leading integer, to provide or request information from the GUI. Whenever the GUI receives this type of packet, it evaluates and reports the result by sending UDP packets back, which is described in the outgoing UDP packet ([Section 8.3.2](#)). Below are example predefined Identifiers in the REC-GUI framework:

Table 8.3.1 Example UDP Packet Information - MATLAB to GUI

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

* Event codes are defined in the MATLAB code. For example, 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 experiments.

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

Code snippet from *start_coding_closedloop.m*

```
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 [Section 7.1.3.1](#), the GUI can send custom parameters to the counterpart (e.g., MATLAB) CPU. Each parameter should be prepared in the counterpart. Check the codes on GitHub 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 8.3.2 for details.

Table 8.3.2 Example UDP Packet Information - GUI to MATLAB

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

9 SAVED DATA FILE STRUCTURE – GUI

The GUI can also work as a data acquisition system to store behavioral data, as well as event codes from the counterpart CPU (e.g., MATLAB). 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 *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: <>
```

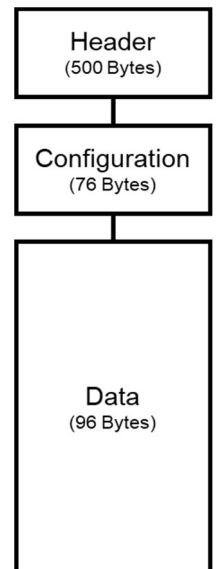


Figure 8.3.1 GUI Data File Structure

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)
```

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 file *BasicCodingStructure.m* and *start_coding_closedloop.m* contains example code showing how to read UDP packets and implement experimental flow. For more examples on how to use the REC-GUI framework, see the REC-GUI Tutorial Guide, as well as the sample code tutorial videos on the REC-GUI website.