TAREA EXPERIMENTAL PARA LA IDENTIFICACIÓN DE MOVIMIENTOS OCULARES ASOCIADOS AL REFRESCAMIENTO ATENCIONAL EN LA MEMORIA DE TRABAJO VISUAL

Manual de uso

1 DE DICIEMBRE DE 2021

UNIVERSIDAD DE COSTA RICA
Centro de Investigación en Neurociencias

1) Introducción

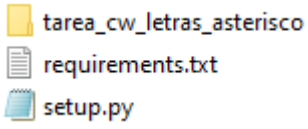
Este documento se divide en cuatro grandes apartados. En el primero, se incluye una breve introducción sobre la organización de este documento. En el segundo, se describen en términos generales los archivos y las subcarpetas contenidas en la carpeta de la tarea experimental. En el tercero, se proporcionan dos diagramas sobre el flujo de ejecución de la tarea experimental. En el cuarto, se presenta el código fuente con comentarios explicativos sobre el propósito de las funciones definidas en los diferentes archivos de Python.

2) Subcarpetas y archivos

Esta tarea experimental se programó con *python 3.9* y con las siguientes librerías: *pylink*, *colormath*, *pandas* y *pygame*. Para reducir la probabilidad de que se presenten errores de ejecución durante la sesión experimental, es importante que no se modifique la posición, el nombre ni el contenido de las carpetas y de los archivos.

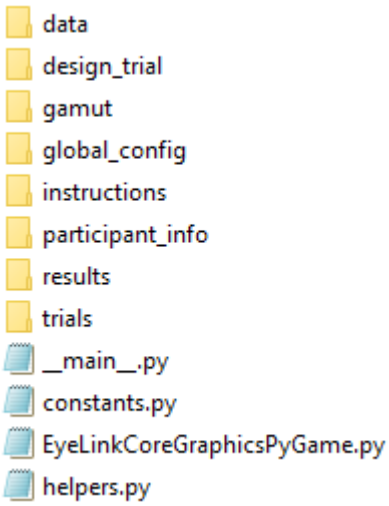
La carpeta que contiene la tarea experimental se compone de una subcarpeta y dos archivos, tal y como puede apreciarse en la Imagen 1. En la carpeta **tarea_cw_letras_asterisco** se almacenan los archivos principales para ejecutar la tarea. El archivo *requirements.txt* incluye las librerías que deben instalarse adicionalmente a las que vienen por defecto con python 3.9. Por su parte, el archivo *setup.py* se puede ejecutar para instalar las dependencias incluidas en *requirements.txt*, pero estas también pueden instalarse individualmente con el módulo *pip* de Python. Además de estas dependencias, es necesario instalar manualmente la librería *pylink*, la cual viene incluida con el *EyeLink Developers Kit / API v2.1.1*, el cual debe descargarse desde el foro de la empresa SR Research (https://www.sr-support.com). La librería *pylink* debe copiarse y pegarse en la carpeta en la que Python almacena las librerías instaladas mediante el módulo *pip*; para el caso de Python 3.9, esta carpeta se denomina *site-packages*.

Imagen 1



A continuación se describen las subcarpetas y archivos contenidos en la carpeta **tarea_cw_letras_asterisco**.

Imagen 2



2.1) Carpeta *data*

2.1.1) Carpeta *fonts*

Contiene los archivos *Vollkorn-Regular.ttf*, *arial.ttf* y *cour.ttf*, los cuales son algunos tipos de letra que pueden emplearse para presentar las instrucciones u otras partes del experimento en donde se requiera mostrar texto. En el caso de este experimento, se emplea el archivo *Vollkorn-Regular.ttf*.

2.1.2) Carpeta *images*

Contiene los archivos que se emplean como fondo de pantalla para visualizar los resultados en el programa *DataViewer* y (2) un archivo de PowerPoint con las flechas que se emplean durante la tarea.

2.1.3) Carpeta *instructions*

Contiene las instrucciones para cada una de las etapas del experimento: ensayos de práctica, ensayos de refrescamiento dirigido, ensayos de refrescamiento paralelo y ensayos de bloqueo subarticulatorio.

2.1.4) Carpeta *sounds*

Contiene tres archivos: *error.wav*, *qbeep.wav* y *type.wav*. Estos son archivos de audio que utiliza el sistema operativo del ET para señalar problemas de calibración o de validación (*error.wav*), el reconcimiento correcto de las fijaciones (*type.wav*) y el fin de la la calibración o de la validación (*qbeep.wav*).

2.2) Carpeta *design_trial*

Contiene una subcarpeta con cuatro archivos en los que cada fila corresponde a lo que debe mostrarse a los participantes en cada ensayo.

2.3) Carpeta *gamut*

Contiene el código fuente en Python con el que se programó la rueda de colores que los participantes utilizan para seleccionar el color de alguno de los discos presentados.

2.4) Carpeta *global_config*

Contiene las constantes para configurar la tarea experimental, tales como el tamaño de la pantalla, el tipo de letras, el tamaño de la rueda de colores, entre otros.

2.5) Carpeta *instructions*

Contiene el código fuente para presentar las instrucciones que el participante lee en las diferentes etapas del experimento para comprender lo que debe hacer durante la ejecución de la tarea.

2.6) Carpeta *participant_info*

Este archivo contiene varias funciones que se encargan de recopilar la siguiente información del participante: nombre, código de identificación, sexo, edad, número de sesión y tipo de experimento. Asimismo, esta función genera un archivo con extensión .csv en la carpeta *results* para guardar la información solicitada, así como las respuestas de los participantes ante la presentación de los estímulos.

2.7) Carpeta *results*

En esta carpeta se localizan los archivos con los datos generados por cada participante durante la ejecución de la tarea. Para cada participante, se generarán dos archivos: uno con la información del registro oculomotor, cuya extensión es .EDF (*EyeLink Data File*), y otro con las respuestas generadas con el *mouse*, cuya extensión es .csv (*Comma Separated Values*). Los nombres de ambos archivos se estructuran de acuerdo con el siguiente formato: código de identificación del participante, fecha (dd/mm/aa) y hora de la sesión (hh/mm/ss).

2.8) Carpeta *trials*

Contiene las funciones que se emplean en cada ensayo para presentar los estímulos a los participantes, así como también de registrar la respuesta dada mediante el mouse. Algunas las funciones incluidas en este archivo se encargan de activar y desactivar el registro de los movimientos oculares, así como también de enviar mensajes al archivo .EDF que se emplean para definir ciertas áreas y periodos de interés. En el apartado 3 del manual se explicará con más detalles la estructura y contenidos de este archivo.

2.9) Archivo *__main__.py*

En este archivo se incluyen las funciones que se encargan de ejecutar todo el experimento en sus grandes etapas: (1) recolección de información básica de los participantes, (2) presentación de las instrucciones y (3) presentación de los ensayos experimentales. En el apartado 3 del manual se explicará con más detalles la estructura y contenidos de este archivo.

2.10) Archivo *constants.py*

Este archivo complementa al incluido en la carpeta ***global_config***, ya que incluye diversas constantes asociadas al diseño experimental. Algunas de las constantes incluidas son el ojo (izquierdo o derecho) que se rastreará, las posiciones en grados en las que aparecerán los discos, el número de ensayo en el que inicia cada etapa de la tarea, los milisegundos durante los cuales se muestra en pantalla cada estímulo experimental, etc.

2.11) Archivo *EyeLinkCoreGraphicsPyGame.py*

Este archivo se puede descargar del foro oficial de la compañía SR Research (https://www.sr-support.com/), cuyo propósito es configurar la ventana de

calibración y validación que se presenta antes del inicio del experimento y de algunos ensayos.

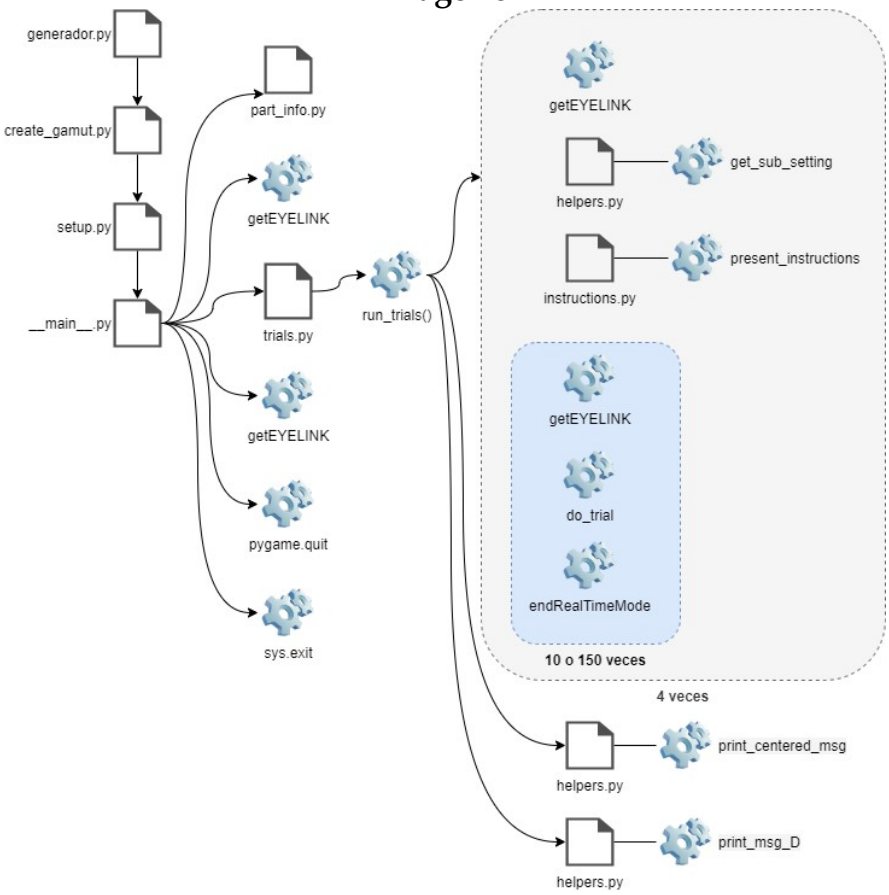
2.12) Archivo *helpers.py*

Contiene las funciones auxiliares que se encargan de mostrar en pantalla los diferentes estímulos experimentales, calificar como correcta o incorrecta las respuestas de los participantes, transformar el código de los colores de un espacio a otro, rotar la rueda de colores, guardar los datos generados por el participante al finalizar cada ensayo, entre otras.

3) Ejecución de la tarea experimental

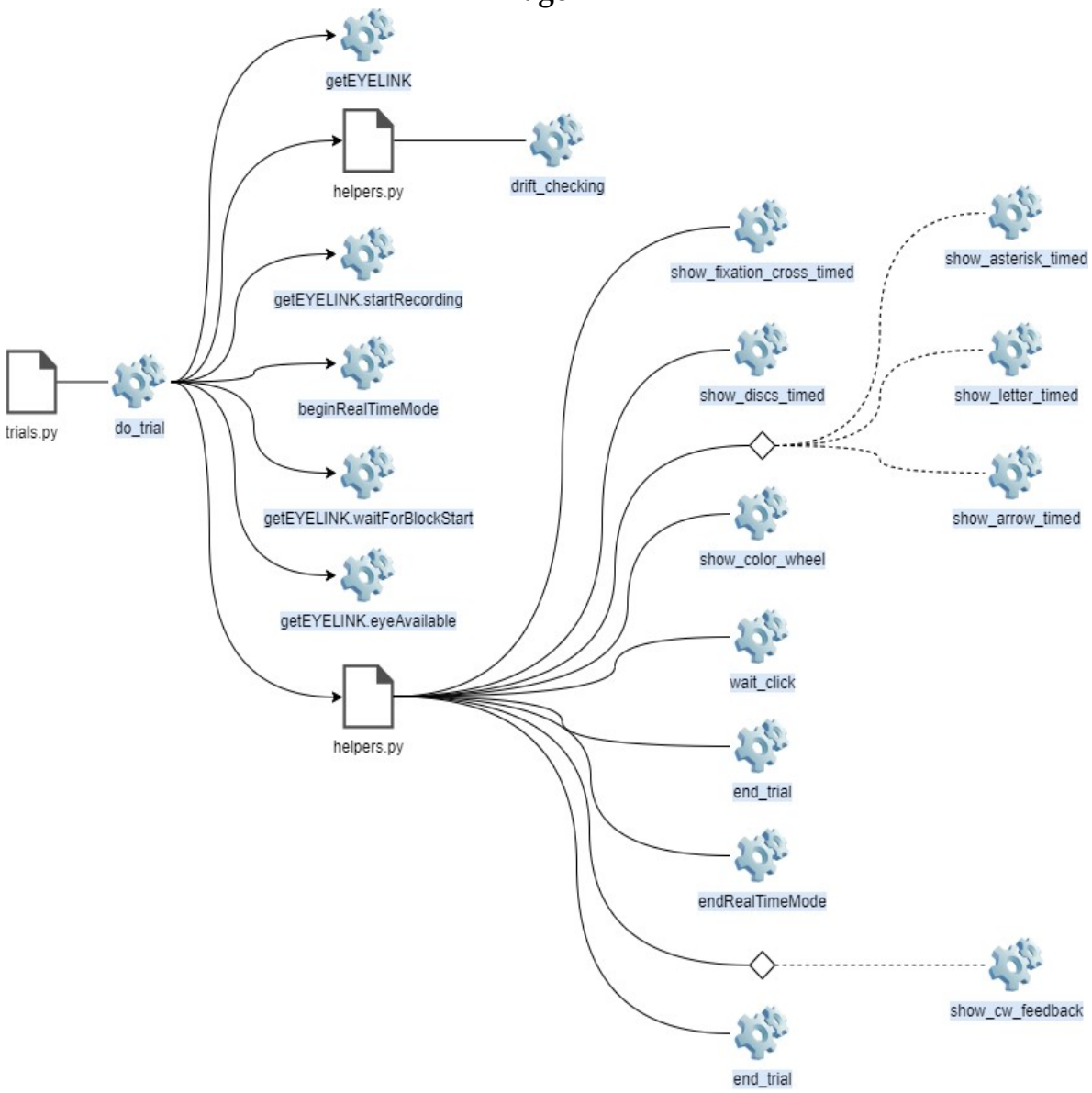
En la Imagen 3 se presenta la estructura general del experimento, la cual está programada fundamentalmente en los archivos *__main__.py* y *trials.py*. Los componentes rodeados por un perímetro de línea discontinua corresponden a los ensayos experimentales, cada uno de los cuales inicia con una cruz de fijación y finaliza cuando el participante da clic en una rueda cromática. Es importante señalar que la Imagen 3 no representa todas las funciones del código fuente, sino solo aquellas que son más relevantes para comprender la dinámica general del experimento.

Imagen 3



Por otra parte, en la imagen 4 se visualiza la función más importante de las presentadas en la Imagen 3, a saber, *do_trial()*. En esta función se presentan los estímulos experimentales al participante con base en las funciones auxiliares almacenadas en el archivo *helpers.py*.

Imagen 4



A continuación se presentarán las funciones más importantes incluidas en los diferentes archivos y subcarpetas mencionados anteriormente.

4) Código fuente

setup.py

Instala los paquetes de Python de los que depende la tarea, los cuales están en el archivo requirements.txt

```
import os

thelibFolder = os.path.dirname(os.path.realpath(__file__))
requirementPath = thelibFolder + '/requirements.txt'
install_requires = []

if os.path.isfile(requirementPath):
    with open(requirementPath) as f:
        install_requires = f.read().splitlines()

for require in install_requires:
    os.system("python3 -m pip install -U " + require + " --user")
```

__main__.py

Importa módulos de Python, pylink y específicos de la tarea

```
import sys, gc, time
import pygame
import participant_info
import instructions as instr
import trials
from global_config import SCREEN
from pylink import *
```

Inicia la recolección de los datos iniciales del participante

```
part_info, part_results_file = participant_info.collect()
```

Crea el archivo .EDF del participante

```
edfFileName = ''.join([part_info['par_id'], part_info['age'], '.EDF'])
getEYELINK().openDataFile(edfFileName)
```

Configuración inicial del eyetracker

```
flushGetkeyQueue()
getEYELINK().setOfflineMode()
getEYELINK().sendCommand("screen_pixel_coords =  0 0 %d %d" % (SCREEN.get_rect().w - 1, SCREEN.get_rect().h - 1))
getEYELINK().sendMessage("DISPLAY_COORDS  0 0 %d %d" % (SCREEN.get_rect().w - 1, SCREEN.get_rect().h - 1))
tracker_software_ver = 0
eyelink_ver = getEYELINK().getTrackerVersion()
if eyelink_ver == 3:
    tvstr = getEYELINK().getTrackerVersionString()
    vindex = tvstr.find("EYELINK CL")
    tracker_software_ver = int(float(tvstr[(vindex + len("EYELINK CL")):].strip()))
if eyelink_ver >= 2:
    getEYELINK().sendCommand("select_parser_configuration 0")
    if eyelink_ver == 2:
        getEYELINK().sendCommand("scene_camera_gazemap = NO")
else:
    getEYELINK().sendCommand("saccade_velocity_threshold = 35")
    getEYELINK().sendCommand("saccade_acceleration_threshold = 9500")
```

Establece los tipos de movimientos oculares que se guardarán en el archivo .EDF

```
getEYELINK().sendCommand("file_event_filter = LEFT,RIGHT,FIXATION,SACCADE,BLINK,MESSAGE,BUTTON,INPUT")
if tracker_software_ver >= 4:
    getEYELINK().sendCommand("file_sample_data  = LEFT,RIGHT,GAZE,AREA,GAZERES,STATUS,HTARGET,INPUT")
else:
    getEYELINK().sendCommand("file_sample_data  = LEFT,RIGHT,GAZE,AREA,GAZERES,STATUS,INPUT")
```

Establece los tipos de movimientos oculares que se enviarán mediante al host

```
getEYELINK().sendCommand("link_event_filter = LEFT,RIGHT,FIXATION,SACCADE,BLINK,BUTTON,INPUT")
```

Establece los tipos de movimientos oculares de los que se extraerán muestras

```
if tracker_software_ver >= 4:
    getEYELINK().sendCommand("link_sample_data  = LEFT,RIGHT,GAZE,GAZERES,AREA,STATUS,HTARGET,INPUT")
else:
    getEYELINK().sendCommand("link_sample_data  = LEFT,RIGHT,GAZE,GAZERES,AREA,STATUS,INPUT")
```

Ejecuta los ensayos de la tarea

```
if getEYELINK().isConnected() and not getEYELINK().breakPressed():
    trials.run_trials(part_info, part_results_file)
```

Cierra el archivo .EDF del participante y lo copia a la computadora que ejecutó la tarea

```
if getEYELINK() != None:
    getEYELINK().setOfflineMode()
    msecDelay(500)
    getEYELINK().closeDataFile()
    getEYELINK().receiveDataFile(edfFileName, ''.join([part_results_file[:-4], '.EDF']))
    getEYELINK().close()
```

Cierra pygame y finaliza el intérprete de Python

```
pygame.quit()
sys.exit()
```

part_info.py

Importa módulos de Python y especificos de la tarea

```
import sys
import time
import pygame
from os import listdir
from os.path import isfile, join
import constants as const
import global_config as glc
import helpers as hlp
```

Registra información inicial del participante

```
def collect():
        part_info = {}
        user_input = ""
        counter = 0
        running = True
        while running:
                for evt in pygame.event.get():
                        if evt.type == pygame.KEYDOWN and evt.key == pygame.K_ESCAPE:
                                running = False
                                pygame.quit()
                                sys.exit()
                        if evt.type == pygame.KEYDOWN:
                                if evt.unicode.isalnum():
                                        user_input += evt.unicode
                                elif evt.key == pygame.K_BACKSPACE:
                                        user_input = user_input[:-1]
                                elif evt.key == pygame.K_SPACE:
                                        user_input += " "
                                elif evt.key == pygame.K_RETURN:
                                        running, counter = validate_req_data(
                                                part_info, user_input, counter)
                                        user_input = ""
                                        counter += 1
                        elif evt.type == pygame.QUIT:
                                pygame.quit()
                                sys.exit()
                display_input_data(user_input)
                display_req_data(counter)
        part_results_file = write_csv_header(part_info)
        hlp.print_centered_msg(const.CLICK_MSG)
        return part_info, part_results_file
```

Valida datos del participante

```
def validate_req_data(part_info, user_input, counter):
        running = True
        if counter == 0:
                part_info['name'] = user_input
                if part_info['name'] == "":
                        counter -= 1
        if counter == 1:
                part_info['par_id'] = str(user_input)
                try: int(part_info['par_id'])
                except ValueError:
                        counter -= 1
                else:
                        if len(part_info['par_id']) != 3:
                                counter -= 1
        if counter == 2:
                part_info['sex'] = user_input
                if part_info['sex'] not in ["masculino", "femenino"]:
                        counter -= 1
        if counter == 3:
                part_info['age'] = user_input
                try: int(part_info['age'])
                except ValueError:
                        counter -= 1
                else:
                        if int(part_info['age']) not in list(range(18, 99)):
                                counter -= 1
        if counter == 4:
                part_info['session'] = user_input
                if part_info['session'] not in ["1"]:
                        counter -= 1
        if counter == 5:
                part_info['setting'] = user_input
                if part_info['setting'] not in [
                        "bloqueodirigido1", "dirigidobloqueo2",
                        "bloqueoparalelo3","paralelobloqueo4"]: counter -= 1
                else: running = False
        return running, counter
```

Muestra en la pantalla los datos del participante que han sido digitados

```
def display_input_data(user_input):
        user_input_x = glc.SCREEN.get_rect().w//2
        user_input_y = glc.SCREEN.get_rect().h//1.5
        glc.SCREEN.fill(const.BLACK)
        user_input_rendered = glc.FONT.render(user_input, True, const.WHITE)
        rect = user_input_rendered.get_rect()
```

```
        rect.center = (user_input_x, user_input_y)
        glc.SCREEN.blit(user_input_rendered, rect)
```

Solicita los datos iniciales del participantes

```
def display_req_data(counter):
        if counter == 0:
                print_req_input(const.NAME_INPUT)
        if counter == 1:
                print_req_input(const.ID_INPUT)
        if counter == 2:
                print_req_input(const.SEX_INPUT)
        if counter == 3:
                print_req_input(const.AGE_INPUT)
        if counter == 4:
                print_req_input(const.SESSION_INPUT)
        if counter == 5:
                print_req_input(const.SETTING_INPUT)
```

Muestra en pantalla los datos del participante que se requieren

```
def print_req_input(msg):
        next_stage = glc.FONT.render(msg, True, const.WHITE)
        next_stage_rect = next_stage.get_rect()
        next_stage_rect.center = glc.SCREEN.get_rect().center
        glc.SCREEN.blit(next_stage, next_stage_rect)
        pygame.display.flip()
```

Guarda en el archivo del participante sus datos iniciales

```
def write_csv_header(part_info):
        part_results_file = ''.join(
                ['./results/', part_info['par_id'], "_" , time.strftime("%d%m%Y_%H%M%S"), '.csv'])
        with open(part_results_file, 'a') as f:
                f.write('{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};\n'.format(
                        'trial', 'practice_trial','name', 'par_id', 'sex', 'age', 'session', 'setting',
                        'hsl_color', 'target_color', 'non_transformed_error',
                        'transformed_error','colorwheel_response_time','letter_presented',
                        'letter_type_presented', 'letter_response_time', 'letter_type_selected',
                        'letter_accuracy'))
        return part_results_file
```

trials.py

Importa módulos de Python, pylink y módulos específicos de la tarea

```
import gc
import pandas
import pygame
import helpers as hlp
import constants as const
import global_config as glc
import instructions as instr
from pylink import *
```

Crea cada ensayo a partir del diseño del experimento

```
def do_trial(trial, trials_design, sub_setting, part_info, part_results_file, cycle):
    letter_result = []
```

Número de ensayo en la pantalla del equipo host

```
    message = "record_status_message 'Ensayo %d '" % (trial)
    getEYELINK().sendCommand(message)
```

Número de ensayo para identificarlo en el archivo .EDF

```
    msg = "TRIALID %s" % str(trial)
    getEYELINK().sendMessage(msg)
```

Identificar imágenes de fondo para visualizar archivo .EDF en DataViewer

```
    trial_discs = ''.join(
        ["./data/images/", "discs_", part_info['setting'], "_trial_", str(trial), ".png"])
    getEYELINK().sendMessage("!V TRIAL_VAR image  %s" % trial_discs)
```

Chequeo de drift en algunos ensayos

```
    if trial in const.DRIFT_CHECKINGS:
        hlp.drift_checking(glc.SCREEN)
```

Cambia eyetracker al modo offLine

```
    getEYELINK().setOfflineMode()
    msecDelay(50)
    error = getEYELINK().startRecording(1, 1, 1, 1)
    if error:
        return error
    gc.disable()
```

Iniciar modo realtime para registrar los movimientos oculares

```
    beginRealTimeMode(100)
    try:
        getEYELINK().waitForBlockStart(100, 1, 0)
    except RuntimeError:
```

Genera un error si no se reciben datos antes de 100 milisegundos

```
        if getLastError()[0] == 0:
            hlp.end_trial()
            print("ERROR: No link samples received!")
            return TRIAL_ERROR
        else:
            raise
```

Establece cuál(es) ojo(s) está(n) siendo registrados(s)

```
    eye_used = getEYELINK().eyeAvailable()
    if eye_used == const.RIGHT_EYE:
        getEYELINK().sendMessage("EYE_USED 1 RIGHT")
    elif eye_used == const.LEFT_EYE or eye_used == const.BINOCULAR:
        getEYELINK().sendMessage("EYE_USED 0 LEFT")
        eye_used = const.LEFT_EYE
    else:
        print("Error in getting the eye information!")
        return TRIAL_ERROR
    getEYELINK().flushKeybuttons(0)
    buttons = (0, 0)
```

Muestra la cruz de fijación

```
    hlp.show_fixation_cross_timed(glc.SCREEN)
```

Marca el momento en el que se muestran los discos de colores

```
    startTime = currentTime()
    getEYELINK().sendMessage("SYNCTIME %d" % (currentTime() - startTime))
```

Muestra los discos de colores

```
    hlp.show_discs_timed(glc.SCREEN, trial, trials_design, part_info)
```

Muestra la manipulación experimental en función del escenario que le haya correspondido al partipante

```
    if sub_setting == 'bloqueo':
        letter_result = hlp.show_letter_timed(glc.SCREEN, trial, trials_design, part_info)
    if sub_setting == 'dirigido':
        hlp.show_arrow_timed(glc.SCREEN, trial, trials_design, part_info)
    if sub_setting == 'paralelo':
        hlp.show_asterisk_timed(glc.SCREEN)
```

Muestra la rueda de colores

```
    ticks = hlp.show_color_wheel(glc.SCREEN, trial, trials_design)
```

Espera a que el participante dé clic en la rueda de colores

```
    clicked = False
    while not clicked:
        clicked, cw_error = hlp.wait_click(
            glc.SCREEN, trial, trials_design, part_info,
            part_results_file, letter_result, ticks)
        if clicked != None:
```

Finaliza ensayo y registro de movimientos oculares

```
            hlp.end_trial()
            getEYELINK().sendMessage("TRIAL_RESULT %d" % (buttons[0]))
            ret_value = getEYELINK().getRecordingStatus()
            endRealTimeMode()
            gc.enable()
            hlp.show_cw_feedback(trial, cw_error)
            ret_value = 0
            return ret_value
```

Verifica si registro fue finalizado

```
        error = getEYELINK().isRecording()
        if error != 0:
            hlp.end_trial()
            return error
```

Verifica si el programa finalizó mediante ALT-F4 o CTRL-C

```
        if (getEYELINK().breakPressed()):
            hlp.end_trial()
            return ABORT_EXPT
```

Verifica si se presionó ESC

```
        elif (getEYELINK().escapePressed()):
            hlp.end_trial()
            return SKIP_TRIAL
```

Verifica si se presionó algún botón en el host

```
        buttons = getEYELINK().getLastButtonPress()
        if (buttons[0] != 0):
            getEYELINK().sendMessage("ENDBUTTON %d" % (buttons[0]))
            hlp.end_trial()
            break
```

Ejecuta los ensayos que fueron creados con do_trials(). Los ensayos se agrupan en 4 ciclos: práctica, primer escenario, práctica y segundo escenario. En cada escenario se emplean dos de las siguientes manipulaciones experimentales: bloqueo, refrescamiento dirigido y refrescamiento paralelo.

```
def run_trials(part_info, part_results_file):
```

Se selecciona el escenario para el participante

```
    setting = part_info['setting']
```

Se importa el diseño de los ensayos con base en el escenario del participante

```
    trials_design = pandas.read_csv(
        ''.join([const.TRIAL_PATH, setting, '.csv']), sep=';').set_index('trial_ID').T.to_dict('list')
```

Calibra el eyetracker getEYELINK().doTrackerSetup()

Para cada uno de los 4 ciclos,

```
    for cycle in range(1, 5):
```

- se selecciona la manipulación experimental

```
        sub_setting = hlp.get_sub_setting(cycle, part_info['setting'])
```

- se presentan las instrucciones y

```
        instr.present_instructions(cycle, setting, part_info)
```

- se ejecutan los ensayos

```
        for trial in range(const.trials_per_cycle[cycle][0], const.trials_per_cycle[cycle][1]):
            pygame.mouse.set_visible(0)
            pygame.mouse.set_pos(glc.SCREEN.get_rect().center)
            if getEYELINK().isConnected() == 0 or getEYELINK().breakPressed():
                break
            while 1:
```

Recibe valor de control del ensayo

```
                ret_value = do_trial(
                    trial, trials_design, sub_setting, part_info, part_results_file, cycle)
                if ret_value == 0:
                    break
```

Finaliza registro de movimientos oculares

```
                endRealTimeMode()
```

Verifica el resultado de la ejecución del ensayo

```
                if ret_value == TRIAL_OK:
                    getEYELINK().sendMessage("TRIAL OK")
                    break
                elif ret_value == SKIP_TRIAL:
                    getEYELINK().sendMessage("TRIAL ABORTED")
                    break
                elif ret_value == ABORT_EXPT:
                    getEYELINK().sendMessage("EXPERIMENT ABORTED")
                    return ABORT_EXPT
                elif ret_value == REPEAT_TRIAL:
```

```
                getEYELINK().sendMessage("TRIAL REPEATED")
            else:
                getEYELINK().sendMessage("TRIAL ERROR")
                break
```

Cuando finalizan los 4 ciclos, se agradece al participante

```
hlp.print_centered_msg(const.THANKS_MSG)
hlp.print_msg_D(glc.FONT.render(const.CLICK_END_MSG, True, const.WHITE))
pygame.display.flip()
```

Cuando el participante da clic, finaliza la tarea

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
            running = False
```

helpers.py

Importa módulos de Python, pylink y específicos de la tarea

```
import random
import pygame
import math
import constants as const
import global_config as glc
from pygame.math import Vector2
from colormath.color_conversions import convert_color
from colormath.color_objects import HSLColor, AdobeRGBColor
from pylink import *
```

Verifica, antes de iniciar ciertos ensayos, si ha ocurrido drift

```
def drift_checking(screen):
    pygame.mouse.set_visible(0)
    while 1:
        if not getEYELINK().isConnected():
            return ABORT_EXPT
        try:
            error = getEYELINK().doDriftCorrect(screen.get_rect().w // 2, screen.get_rect().h // 2, 1, 1)
            if error != 27:
                break
            else:
                getEYELINK().doTrackerSetup()
        except:
            getEYELINK().doTrackerSetup()
    getEYELINK().setOfflineMode()
    msecDelay(50)
```

Finaliza el registro de los movimientos oculares

```
def end_trial():
    endRealTimeMode()
    pumpDelay(100)
    getEYELINK().stopRecording()
    while getEYELINK().getkey(): pass
```

Transforma coordenadas de espacio de colores RGB a las de un espacio de colores HSL

```
def rgb2hsl(r, g, b):
    rgb = AdobeRGBColor(r, g, b)
    hsl_angle_color = convert_color(rgb, HSLColor).get_value_tuple()
    return hsl_angle_color[0]
```

Transforma coordenadas de espacio de colores HSL a las de un espacio de colores RGB

```
def hsl2rgb(h, s=1, l=0.5):
    hsl = HSLColor(h, s, l)
    color = convert_color(hsl, AdobeRGBColor)
    rgb_color = AdobeRGBColor.get_upscaled_value_tuple(color)
    return rgb_color
```

Selecciona manipulación experimental

```
def get_sub_setting(cycle, setting):
    if cycle in [1, 2]:
        sub_setting = const.setting_dic[setting][0]
    else:
        sub_setting = const.setting_dic[setting][1]
    return sub_setting
```

Rota rueda de colores

```
def rotate_color_wheel(screen, wheel):
    rotated_wheel = pygame.transform.rotate(wheel, random.randint(0, 360))
    rotated_wheel_rect = rotated_wheel.get_rect()
    rotated_wheel_rect.center = screen.get_rect().center
    return rotated_wheel, rotated_wheel_rect
```

Muestra rueda de colores

```
def show_color_wheel(screen, trial, trials_design):
    position = Vector2()
    rotated_wheel, rotated_wheel_rect = rotate_color_wheel(screen, glc.WHEEL)
    screen.blit(rotated_wheel, (
        screen.get_rect().size[0] / 2 - rotated_wheel_rect.size[0] / 2,
        screen.get_rect().size[1] / 2 - rotated_wheel_rect.size[1] / 2))
    screen.blit(glc.QUESTION, glc.QUESTION_RECT)
    position.from_polar((const.DISTANCE, int(trials_design[trial][11][-3:])))
    pos = (int(screen.get_rect().center[0] + position[0]),
           int(screen.get_rect().center[1] - position[1]))
    pygame.gfxdraw.filled_circle(screen, pos[0], pos[1], const.RADIUS, const.BLACK)
    pygame.gfxdraw.aacircle(screen, pos[0], pos[1], const.RADIUS, const.WHITE)
    pygame.display.flip()
    getEYELINK().sendMessage("colwheel_start")
    col_wheel = ''.join(['../data/images/', 'col_wheel.png'])
    getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % col_wheel)
    pygame.mouse.set_visible(1)
    ticks = pygame.time.get_ticks()
    return ticks
```

Muestra rueda de colores cuando se están mostrando las instrucciones al participante

```
def show_color_wheel_fixed_colors(counter, event):
    instr_position = Vector2()
    pygame.mouse.set_visible(1)
    ins_wheel = pygame.transform.rotate(glc.WHEEL, 180)
    ins_wheel_rect = ins_wheel.get_rect()
    ins_wheel_rect.center = glc.SCREEN.get_rect().center
    glc.SCREEN.blit(ins_wheel, (
        glc.SCREEN.get_rect().size[0] / 2 - ins_wheel_rect.size[0] / 2, int(
            glc.SCREEN.get_rect().size[1] / 2 - ins_wheel_rect.size[1] / 2)))
    glc.SCREEN.blit(glc.QUESTION, glc.QUESTION_RECT)
    instr_position.from_polar((const.DISTANCE, 30))
    pos = (int(glc.SCREEN.get_rect().center[0] + instr_position[0]),
           int(glc.SCREEN.get_rect().center[1] + -instr_position[1]))
    pygame.gfxdraw.filled_circle(
```

```
        glc.SCREEN, pos[0], pos[1], const.RADIUS, const.BLACK)
    pygame.gfxdraw.aacircle(
        glc.SCREEN, pos[0], pos[1], const.RADIUS, const.WHITE)
    x, y = event.pos
    dist = abs(math.sqrt(
        (x - glc.SCREEN.get_rect().center[0]) ** 2 +
        (y - glc.SCREEN.get_rect().center[1]) ** 2))
    if glc.INNER_CIRCLE_RADIUS < dist < glc.OUTER_CIRCLE_RADIUS:
        pygame.mouse.set_visible(0)
        glc.SCREEN.fill(const.BLACK)
        counter += 1
    return counter
```

Muestra el feedback durante los ensayos de práctica

```
def show_cw_feedback(trial, cw_error):
    if trial in const.FEEDBACK_TRIALS:
        if cw_error > const.MAX_ANGLE:
            print_centered_msg(const.NOT_ACCUCURATE_ANSW_MSG)
            pygame.time.delay(const.FEEDBACKTIME)
```

Detecta clic del partipante en la rueda de colores. Si esto ocurre, calcula tiempo de respuesta, color seleccionado y grados de error

```
def wait_click(screen, trial, trials_design, part_info, part_results_file, letter_result, ticks):
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONUP:
                x, y = pygame.mouse.get_pos()
                dist = abs(math.sqrt(
                    (x - screen.get_rect().center[0]) ** 2 +
                    (y - screen.get_rect().center[1]) ** 2))
                if glc.INNER_CIRCLE_RADIUS < dist < glc.OUTER_CIRCLE_RADIUS:
                    cw_response_time = pygame.time.get_ticks() - ticks
                    pygame.mouse.set_pos(screen.get_rect().center)
                    getEYELINK().sendMessage("colwheel_end")
                    pygame.mouse.set_visible(0)
                    hsl_color, target_color = compute_color(
                        screen, trial, trials_design, x, y)
                    transformed_error, non_transformed_error = compute_transformed_error(
                        hsl_color, target_color)
                    write_csv_part_results(
                        part_results_file, part_info, trial, trials_design,
                        hsl_color, target_color, non_transformed_error,
                        transformed_error, letter_result, cw_response_time)
                    return 0, transformed_error
```

Selecciona el color del disco diana con base en el diseño del ensayo

```
def get_target_color(trial, trials_design):
    target_disc = trials_design[trial][11]
    degrees = target_disc.split('_')[1]
    target_color = trials_design[trial][const.disc_pos[degrees]]
    return target_color
```

Crea color con base en el diseño del ensayo

```
def compute_color(screen, trial, trials_design, x, y):
    clicked_color = screen.get_at((x, y))
    hsl_color = rgb2hsl(clicked_color[0], clicked_color[1], clicked_color[2])
    target_color = get_target_color(trial, trials_design)
    return hsl_color, target_color
```

Calcula grados de error (originales y transformados)

```
def compute_transformed_error(hsl_color, target_color):
    diference = round(abs(target_color - hsl_color), 3)
    non_transformed_error = diference
    if diference < 180:
        transformed_error = diference
    else:
        transformed_error = 180 - (diference % 180)
    return transformed_error, non_transformed_error
```

Crea cruz de fijación

```
def show_fixation_cross(screen):
    pygame.draw.lines(screen, const.WHITE, False, [
        (screen.get_rect().center[0], screen.get_rect().center[1] - 30),
        (screen.get_rect().center[0], screen.get_rect().center[1] + 30)], 5)
    pygame.draw.lines(screen, const.WHITE, False, [
        (screen.get_rect().center[0] - 30, screen.get_rect().center[1]),
        (screen.get_rect().center[0] + 30, screen.get_rect().center[1])], 5)
    pygame.display.flip()
```

Muestra cruz de fijación

```
def show_fixation_cross_timed(screen):
    screen.fill(const.BLACK)
    show_fixation_cross(screen)
    getEYELINK().sendMessage("cross_start")
    fixcross = ''.join(["../data/images/", "fixation_cross.png"])
    getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % fixcross)
    pygame.image.save(screen, ''.join(["./data/images/", "fixation_cross", ".png"]))
    pygame.time.delay(const.CROSSTIME)
    getEYELINK().sendMessage("cross_end")
    screen.fill(const.BLACK)
```

Crea discos de colores con base en el diseño del ensayo

```
def show_discs(screen, trial, trials_design, part_info):
    position = Vector2()
    pygame.display.flip()
    for disc_degress, color_degrees in zip(
            const.DISCS_DEGREES, trials_design[trial][1:7]):
        position.from_polar((
            const.DISTANCE, int(disc_degress)))
        pos = (
            int(screen.get_rect().center[0] + position[0]),
            int(screen.get_rect().center[1] - position[1]))
        rgb = hsl2rgb(h=color_degrees)
        pygame.draw.circle(screen, rgb, pos, const.RADIUS)
        left = pos[0] - const.RADIUS
```

```
            top = pos[1] - const.RADIUS
            right = pos[0] + const.RADIUS
            bottom = pos[1] + const.RADIUS
            getEYELINK().sendMessage("!V IAREA ELLIPSE %s %s %s %s %s %s" % (
                disc_degress, left, top, right, bottom, color_degrees))
        pygame.display.flip()
        getEYELINK().sendMessage("discs_start")
        trial_discs = ''.join(["../data/images/", "discs_", part_info['setting'], "_trial_", str(trial), ".png"])
        getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % trial_discs)
        pygame.image.save(screen, ''.join(["./data/images/", "discs_", "bloqueodirigido1", "_trial_", str(trial), ".png"]))
        pygame.image.save(screen, ''.join(["./data/images/", "discs_", "dirigidobloqueo2", "_trial_", str(trial), ".png"]))
        pygame.image.save(screen, ''.join(["./data/images/", "discs_", "bloqueoparalelo3", "_trial_", str(trial), ".png"]))
        pygame.image.save(screen, ''.join(["./data/images/", "discs_", "paralelobloqueo4", "_trial_", str(trial), ".png"]))
```

Muestra discos de colores

```
def show_discs_timed(screen, trial, trials_design, part_info):
    screen.fill(const.BLACK)
    show_discs(screen, trial, trials_design, part_info)
    pygame.time.delay(const.DISCSTIME)
    screen.fill(const.BLACK)
    getEYELINK().sendMessage("discs_end")
```

Muestra discos de colores cuando se están presentando las instrucciones a los participantes

```
def show_discs_fixed_colors(screen):
    instr_position = Vector2()
    for disc_degress, color_degrees in zip(const.DISCS_DEGREES, [0, 90, 180, 270, 300, 330]):
        instr_position.from_polar((const.DISTANCE, int(disc_degress)))
        pos = (int(screen.get_rect().center[0] + instr_position[0]),
               int(screen.get_rect().center[1] + -instr_position[1]))
        rgb = hsl2rgb(h=color_degrees)
        pygame.draw.circle(screen, rgb, pos, const.RADIUS)
```

Crea letras

```
def show_letter(screen, trial, letter, position, part_info):
    FONT = pygame.font.Font(const.FONT_PATH, 96)
    top = 500
    bottom = 700
    if position == 'left':
        left = 400
        right = 600
        screen.blit(FONT.render(letter, True, const.WHITE), (glc.WIDTH / 4, glc.HEIGHT / 2))
        getEYELINK().sendMessage("!V IAREA RECTANGLE %s %s %s %s %s %s" % (trial, left, top, right, bottom, 'left'))
    else:
        left = 1400
        right = 1600
        screen.blit(FONT.render(letter, True, const.WHITE), (3 * glc.WIDTH / 4, glc.HEIGHT / 2))
        getEYELINK().sendMessage("!V IAREA RECTANGLE %s %s %s %s %s %s" % (trial, left, top, right, bottom, 'right'))

    pygame.display.flip()
    getEYELINK().sendMessage("letter_start")
    getEYELINK().sendMessage("manipulation_start")
    trial_letter = ''.join(["../data/images/", "letter_", part_info['setting'], "_trial_", str(trial), ".png"])
    getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % trial_letter)
```

Muestra letras

```
def show_letter_timed(screen, trial, trials_design, part_info):
    letter = trials_design[trial][7]
    position = trials_design[trial][9]
    screen.fill(const.BLACK)
    show_letter(screen, trial, letter, position, part_info)
    pygame.image.save(screen, ''.join(
        ["./data/images/", "letter_", "bloqueodirigido1", "_trial_", str(trial), ".png"]))
    pygame.image.save(screen, ''.join(
        ["./data/images/", "letter_", "dirigidobloqueo2", "_trial_", str(trial), ".png"]))
    pygame.image.save(screen, ''.join(
        ["./data/images/", "letter_", "bloqueoparalelo3", "_trial_", str(trial), ".png"]))
    pygame.image.save(screen, ''.join(
        ["./data/images/", "letter_", "paralelobloqueo4", "_trial_", str(trial), ".png"]))
    letter_results = check_letter_response(trial, trials_design)
    show_letter_feedback(trial, letter_results[0], letter_results[2])
    screen.fill(const.BLACK)
    getEYELINK().sendMessage("letter_end")
    getEYELINK().sendMessage("manipulation_end")
    return letter_results
```

Verifica si participante seleccionó correctamente el tipo de letra

```
def check_letter_response(trial, trials_design):
    correct_letter_type = trials_design[trial][8]
    ticks = pygame.time.get_ticks()
    letter_response = 0
    elapsed_time = 0
    pressed_click = 0
    while elapsed_time < const.LETTERTIME:
        elapsed_time = pygame.time.get_ticks() - ticks
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONDOWN and event.button in [const.LEFT, const.RIGHT] and not pressed_click:
                letter_response = pygame.time.get_ticks() - ticks
                if event.button == const.LEFT:
                    letter_type = const.VOCAL
                if event.button == const.RIGHT:
                    letter_type = const.CONSONANT
                if letter_type == correct_letter_type:
                    letter_accuracy = 1
                else:
                    letter_accuracy = 0
            pressed_click = 1
    if letter_response == 0:
        letter_response = const.NO_RESPONSE_LETTER
        letter_accuracy = 0
        letter_type = const.NO_RESPONSE_LETTER
        return [letter_response, letter_type, letter_accuracy]
```

Muestra feedback en un ensayo de práctica.

```
def show_letter_feedback(trial, letter_response, letter_accuracy):
    if trial in const.FEEDBACK_TRIALS:
```

```
        if letter_response == const.NO_RESPONSE_LETTER:
            print_centered_msg(const.SLOW_ANSW_MSG)
            pygame.time.delay(const.FEEDBACKTIME)

        if letter_accuracy == 0 and letter_response != const.NO_RESPONSE_LETTER:
            print_centered_msg(const.WRONG_ANSW_MSG)
            pygame.time.delay(const.FEEDBACKTIME)
```

Califica respuesta con base en el tipo de letra seleccionado por el participante

```
def get_letter_result(trial, trials_design, letter_result):
    if trials_design[trial][7] == 'no_letter':
        result = ['no_letter', 'no_letter', 'no_letter']
    else:
        result = letter_result
    return result
```

Crea asterisco

```
def show_asterisk(screen):
    FONT = pygame.font.Font(const.FONT_PATH, 96)
    screen.blit(FONT.render('*', True, const.WHITE), (glc.WIDTH / 2, glc.HEIGHT / 2))
    pygame.display.flip()
    getEYELINK().sendMessage("asterisk_start")
    getEYELINK().sendMessage("manipulation_start")
    trial_asterisk = ''.join(["../data/images/", "asterisk", ".png"])
    getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % trial_asterisk)
    pygame.image.save(screen, ''.join(["./data/images/", "asterisk", ".png"]))
```

Muestra asterisco

```
def show_asterisk_timed(screen):
    screen.fill(const.BLACK)
    show_asterisk(screen)
    pygame.time.delay(const.ASTERISKTIME)
    screen.fill(const.BLACK)
    getEYELINK().sendMessage("asterisk_end")
    getEYELINK().sendMessage("manipulation_end")
```

Importa flecha

```
def show_arrow(screen, trial, degrees, part_info):
    ARROW_PATH = ''.join([const.ARROW_PATH, degrees, '.png'])
    arrow_image = pygame.image.load(ARROW_PATH).convert_alpha()
    arrow_coor_X = int(screen.get_rect().center[0] - arrow_image.get_rect().size[0] / 2)
    arrow_coor_Y = int(screen.get_rect().center[1] - arrow_image.get_rect().size[1] / 2)
    screen.blit(arrow_image, [arrow_coor_X, arrow_coor_Y])
    pygame.display.flip()
    getEYELINK().sendMessage("arrow_start")
    getEYELINK().sendMessage("manipulation_start")
    trial_arrow = ''.join(["../data/images/", "arrow_", part_info['setting'], "_trial_", str(trial), ".png"])
    getEYELINK().sendMessage("!V IMGLOAD FILL  %s" % trial_arrow)
```

Muestra flecha

```
def show_arrow_timed(screen, trial, trials_design, part_info):
    degrees = trials_design[trial][10].split("_")[1]
    screen.fill(const.BLACK)
    show_arrow(screen, trial, degrees, part_info)
    pygame.image.save(screen, ''.join(
        ["./data/images/", "arrow_", "bloqueodirigido1", "_trial_", str(trial), ".png"]))
    pygame.image.save(screen, ''.join(
        ["./data/images/", "arrow_", "dirigidobloqueo2", "_trial_", str(trial), ".png"]))
    pygame.time.delay(const.ARROWTIME)
    screen.fill(const.BLACK)
    getEYELINK().sendMessage("arrow_end")
    getEYELINK().sendMessage("manipulation_end")
```

Muestra mensajes de instrucciones y feedback en diferentes posiciones de la pantalla

```
def print_centered_msg(msg):
    glc.SCREEN.fill(const.BLACK)
    next_stage = glc.FONT.render(msg, True, const.WHITE)
    next_stage_rect = next_stage.get_rect()
    next_stage_rect.center = glc.SCREEN.get_rect().center
    glc.SCREEN.blit(next_stage, next_stage_rect)
    pygame.display.flip()


def print_msg_L1(msg):
    glc.SCREEN.blit(msg, (50, 40))


def print_msg_L2(msg):
    glc.SCREEN.blit(msg, (50, 80))


def print_msg_L3(msg):
    glc.SCREEN.blit(msg, (50, 120))


def print_msg_L4(msg):
    glc.SCREEN.blit(msg, (50, 160))


def print_msg_D(msg):
    glc.SCREEN.blit(msg, (50, glc.HEIGHT - 60))
```

Guarda datos generados por participante en su correspondiente archivo .csv

```
def write_csv_part_results(file, part_info, trial, trials_design, hsl_color, target_color,
                           non_transformed_error, transformed_error, letter_result, cw_response_time):
    with open(file, 'a') as f:
```

Las columnas del archivo .csv tienen los siguientes encabezados:

```
        f.write('{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};{};\n'.format(
```

trial (trial #, 1-320)

```
            str(trial),
```

practice_trial (yes_practice, no_practice)

```
            trials_design[trial][0],
```

name (participant name)	`part_info['name'],`
par_id (3 digits ID, 001-999)	`part_info['par_id'],`
sex (masculino, femenino)	`part_info['sex'],`
age (18-99)	`part_info['age'],`
session (1)	`part_info['session'],`
setting (1-4)	`const.setting_id[part_info['setting']],`
hsl_color (0-360)	`hsl_color,`
target_color (0-360)	`target_color,`
non_transformed_error (0-360)	`non_transformed_error,`
transformed_error (0-180)	`transformed_error,`
colorwheel_response_time (milisegundos)	`str(cw_response_time),`
letter_presented (aA-zZ, excepto ñ)	`trials_design[trial][7],`
letter_type_presented (no_letter, consonant, vocal)	`trials_design[trial][8],`
letter_response_time (milisegundos)	`str(get_letter_result(trial, trials_design, letter_result)[0]),`
letter_type_selected (-999, consonant, vocal)	`str(get_letter_result(trial, trials_design, letter_result)[1]),`
letter_accuracy (0-1)	`str(get_letter_result(trial, trials_design, letter_result)[2])))`

instructions.py

Importa módulos de Python y específicos de la tarea

```
import sys
import pygame
import pygame.gfxdraw
import helpers as hlp
import constants as const
import global_config as glc
from pygame.math import Vector2
```

Presenta las instrucciones

```
def present_instructions(cycle, setting, part_info):
        sub_setting = hlp.get_sub_setting(cycle, setting)
        present_pre_cycle_instructions(cycle)
        if cycle in [1,3]:
                present_sub_setting_instructions(sub_setting, part_info)
```

Importa el archivo de instrucciones

```
def import_instructions_file(INSTR_PATH):
        with open(INSTR_PATH, encoding = 'latin-1') as f:
                list_instr = f.read().splitlines()
        instructions = {}
        for b in list_instr:
                i = b.split(';')
                instructions[i[0]] = glc.FONT.render(i[1], True, const.WHITE)
        return  instructions
```

Presenta las instrucciones con base en el ciclo que se esté ejecutando

```
def present_pre_cycle_instructions(cycle):
        instructions = import_instructions_file(const.PRE_CYCLE_INSTR_PATH[cycle])
        running = True
        counter = 0
        unders_instr = False
        pygame.mouse.set_visible(0)
        if cycle != 1:
                hlp.print_centered_msg(const.CLICK_MSG)

        while running:
                for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
                                running = False
                                pygame.quit()
                                sys.exit()
                        if event.type == pygame.MOUSEBUTTONDOWN and not unders_instr and counter < 4:
                                glc.SCREEN.fill(const.BLACK)
                                if counter == 0:
                                        hlp.print_msg_L1(instructions['1.0'])
                                        hlp.print_msg_D(instructions['0.0'])
                                if counter == 1:
                                        hlp.print_msg_L1(instructions['2.0'])
                                        hlp.print_msg_D(instructions['0.0'])
                                if counter == 2:
                                        hlp.print_msg_L1(instructions['3.0'])
                                        hlp.print_msg_D(instructions['0.0'])
                                if counter == 3:
                                        unders_instr = True
                                        running = False
                                counter += 1
                                if cycle in [2,4] and counter == 2:
                                                counter = 3
                        pygame.display.flip()
        if cycle in [2,4]: msg_next_stage = const.CLICK_BEGIN_MSG
        else: msg_next_stage = const.CLICK_MSG
        next_stage = glc.FONT.render(msg_next_stage, True, const.WHITE)
        next_stage_rect = next_stage.get_rect()
        next_stage_rect.center = glc.SCREEN.get_rect().center
        glc.SCREEN.blit(next_stage, next_stage_rect)
        pygame.display.flip()
        if cycle in [2,4]:
                running = True
                while running:
                        for event in pygame.event.get():
                                if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
                                        running = False
                                        pygame.quit()
                                        sys.exit()
                                if event.type == pygame.MOUSEBUTTONDOWN:
                                        running = False
```

Presenta instrucciones con base en la manipulación experimental que se va a realizar

```
def present_sub_setting_instructions(sub_setting, part_info):
        instructions = import_instructions_file(const.INSTR_PATH[sub_setting])
        running = True
        counter = 0
        unders_instr = False
        instr_presented = False
        pygame.mouse.set_visible(0)
        while running:
                for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
                                running = False
                                pygame.quit()
```

```
                sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN and not unders_instr and counter < 8:
                glc.SCREEN.fill(const.BLACK)
                if counter == 0:
                        hlp.print_msg_L1(instructions['0.1'])
                        hlp.print_msg_D(instructions['0.0'])
                if counter == 1:
                        hlp.show_fixation_cross(glc.SCREEN)
                        hlp.print_msg_L1(instructions['1.0'])
                        hlp.print_msg_D(instructions['0.0'])
                if counter == 2:
                        hlp.show_discs_fixed_colors(glc.SCREEN)
                        hlp.print_msg_L1(instructions['2.0'])
                        hlp.print_msg_D(instructions['0.0'])
                if counter == 3:
                        hlp.print_msg_L1(instructions['3.0'])
                        hlp.print_msg_L2(instructions['3.1'])
                        hlp.print_msg_D(instructions['0.0'])
                        if sub_setting == 'bloqueo':
                                hlp.show_letter(glc.SCREEN, 0, 'A','left', part_info)
                                hlp.print_msg_L3(instructions['3.2'])
                        if sub_setting == 'paralelo':
                                hlp.show_asterisk(glc.SCREEN)
                        if sub_setting == 'dirigido':
                                hlp.show_arrow(glc.SCREEN, 0, '330', part_info)
                                hlp.print_msg_L3(instructions['3.2'])
                                hlp.print_msg_L4(instructions['3.3'])
                if counter == 4:
                        hlp.print_msg_L1(instructions['4.0'])
                        hlp.print_msg_D(instructions['0.0'])
                if counter == 5:
                        hlp.print_msg_L1(instructions['5.0'])
                        hlp.print_msg_D(instructions['0.0'])
                if counter == 6:
                        hlp.show_color_wheel_fixed_colors(counter, event)
                if counter == 7:
                        hlp.print_msg_L1(instructions['6.0'])
                        instr_presented = True
                counter += 1
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_n and
        instr_presented):
                pygame.mouse.set_visible(0)
                glc.SCREEN.fill(const.BLACK)
                hlp.print_msg_L1(instructions['7.0'])
                pygame.display.flip()
                counter = 0
                unders_instr = False
                instr_presented = False
        if (event.type == pygame.KEYDOWN and event.key == pygame.K_s and instr_presented):
                unders_instr = True
                running = False
pygame.display.flip()
```

global_config.py

Importa módulos de Python, pylink y específicos de la tarea

```
import pygame
import constants as const
import gamut
import helpers as hlp
from pylink import *
from EyeLinkCoreGraphicsPyGame import EyeLinkCoreGraphicsPyGame
```

Inicia pygame

```
pygame.init()
```

Se inicializan las variables globales

```
WIDTH, HEIGHT = pygame.display.Info().current_w, pygame.display.Info().current_h
```

SCREEN = pygame.display.set_mode((WIDTH, HEIGHT), pygame.FULLSCREEN)

```
SCREEN = pygame.display.set_mode((WIDTH, HEIGHT))
FONT = pygame.font.Font(const.FONT_PATH, 28)
INNER_CIRCLE_RADIUS, OUTER_CIRCLE_RADIUS = gamut.create_gamut(HEIGHT)
WHEEL = pygame.image.load(const.GAMUT_PATH).convert_alpha()
QUESTION = FONT.render(const.COLOR_MSG, True, const.WHITE)
QUESTION_RECT = QUESTION.get_rect()
QUESTION_RECT.center = SCREEN.get_rect().center
```

Se vincularn el eyetrackler y el entorno gráfico en el que se ejecutará la tarea

```
eyelinktracker = EyeLink(None)
genv = EyeLinkCoreGraphicsPyGame(SCREEN, eyelinktracker)
openGraphicsEx(genv)
```

constants.py

Constantes del eyetracker

```
DRIFT_CHECKINGS = []
RIGHT_EYE = 1
LEFT_EYE = 0
BINOCULAR = 2
```

Constantes de estímulos experimentales

```
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
DISCS_DEGREES = ["030", "090", "150", "210", "270", "330"]
DISTANCE = 300
RADIUS = 30
LEFT = 1
RIGHT = 3
```

Cantidad de ensayos de práctica para cada manipulación experimental

```
N_PRACTICE_TRIALS = 10
```

Cantidad de ensayos experimentales para cada manipulación experimental

```
N_EXP_TRIALS = 150
```

Número (trial_ID) del ensayo de práctica en el que sa da feedback al participante

```
FEEDBACK_TRIALS = [1, 2, 3, 4, 161, 162, 163, 164]
```

Número (trial_ID) del ensayo en el que inicia cada ciclo

```
INIT_CYCLE1 = 1
INIT_CYCLE2 = 11
INIT_CYCLE3 = 161
INIT_CYCLE4 = 171
```

Tiempos en milisegundos de estímulos experimentales

```
CROSSTIME = 500
DISCSTIME = 500
LETTERTIME = 500
ARROWTIME = 500
ASTERISKTIME = 500
FEEDBACKTIME = 1000
```

Constantes para letras

```
NO_RESPONSE_LETTER = -999
VOCAL = 'vocal'
CONSONANT = 'consonant'
```

Grados en el espacio de colores a partir de los que una respuesta es muy inexacta

```
MAX_ANGLE = 45
```

Ubicación de las instrucciones para cada ciclo

```
PRECYCLE1_PATH = 'data/instructions/pre_cycle1.txt'
PRECYCLE2_PATH = 'data/instructions/pre_cycle2.txt'
PRECYCLE3_PATH = 'data/instructions/pre_cycle3.txt'
PRECYCLE4_PATH = 'data/instructions/pre_cycle4.txt'
```

Ubicación de las instrucciones para cada manipulación experimental

```
INSTR_BLOQUEO_PATH = 'data/instructions/bloqueo.txt'
INSTR_DIRIGIDO_PATH = 'data/instructions/dirigido.txt'
INSTR_PARALELO_PATH = 'data/instructions/paralelo.txt'
```

Ubicación de fuentes, flechas, diseños y rueda de colores

```
FONT_PATH = "./data/fonts/Vollkorn-Regular.ttf"
ARROW_PATH = "./data/images/arrow"
TRIAL_PATH = "./design_trial/generated_trial_designs/"
GAMUT_PATH = "./data/images/gamut.png"
```

Mensajes iniciales y finales de la tarea

```
CLICK_MSG = "Presione clic en el mouse para continuar."
CLICK_BEGIN_MSG = "Presione clic en el mouse para iniciar."
CLICK_END_MSG = "Presione clic en el mouse para salir."
THANKS_MSG = "Muchas gracias por su participación."
NOT_ACCUCURATE_ANSW_MSG = "Respuesta imprecisa."
SLOW_ANSW_MSG = "Muy lento."
WRONG_ANSW_MSG = "Respuesta incorrecta."
COLOR_MSG = "¿Color?"
NAME_INPUT = "Nombre (Nombre y apellidos)"
ID_INPUT = "ID (Tres dígitos)"
SEX_INPUT = "Sexo (masculino / femenino)"
AGE_INPUT = "Edad (Dos dígitos, años cumplidos)"
SESSION_INPUT = "Sesión (1)"
SETTING_INPUT = "Escenario (bloqueodirigido1 / dirigidobloqueo2 / bloqueoparalelo3 / paralelobloqueo4)"
```

Identificación de escenarios

```
setting_id = {
    'bloqueodirigido1': 1,
    'dirigidobloqueo2': 2,
    'bloqueoparalelo3': 3,
    'paralelobloqueo4': 4}

setting_dic = {
    'bloqueodirigido1': ['bloqueo','dirigido'],
```

```
    'dirigidobloqueo2': ['dirigido','bloqueo'],
    'bloqueoparalelo3': ['bloqueo','paralelo'],
    'paralelobloqueo4': ['paralelo','bloqueo']}
```

Identificación de posiciones (en grados) de discos

```
disc_pos = {
    '030': 1,
    '090': 2,
    '150': 3,
    '210': 4,
    '270': 5,
    '330': 6}
```

Ensayos por ciclo

```
trials_per_cycle = {
    1: [INIT_CYCLE1, INIT_CYCLE1 + N_PRACTICE_TRIALS],
    2: [INIT_CYCLE2, INIT_CYCLE2 + N_EXP_TRIALS],
    3: [INIT_CYCLE3, INIT_CYCLE3 + N_PRACTICE_TRIALS],
    4: [INIT_CYCLE4, INIT_CYCLE4 + N_EXP_TRIALS]}
```

Instrucciones de cada ciclo

```
PRE_CYCLE_INSTR_PATH = {
    1: PRECYCLE1_PATH,
    2: PRECYCLE2_PATH,
    3: PRECYCLE3_PATH,
    4: PRECYCLE4_PATH}
```

Instrucciones de cada manipulación experimental

```
INSTR_PATH = {
    'bloqueo': INSTR_BLOQUEO_PATH,
    'dirigido': INSTR_DIRIGIDO_PATH,
    'paralelo': INSTR_PARALELO_PATH}
```

create_gamut.py

Importa módulos de Python

```
import math
import constants as const
from PIL import Image
from colormath.color_conversions import convert_color
from colormath.color_objects import HSLColor, AdobeRGBColor
```

Crea la rueda de colores que se muestra al final de cada ensayo.

```
def create_gamut(HEIGHT):
        s = 1
        l = 0.5
        r = 60
        img_size = int(HEIGHT*0.88)
        img_half = img_size / 2
        inner_radius = int(HEIGHT*0.39)
        outer_radius = int(HEIGHT*0.44)


        def ang_to_coord_colors(s, l, r, angle):
            angle = math.radians(angle)
            x = s + math.cos(angle)*r
            y = l + math.sin(angle)*r
            return int(x), int(y)


        def hsl2rgb(h,s,l):
            hsl = HSLColor(h, s, l)
            colour = convert_color(hsl, AdobeRGBColor)
            rgb_color =  AdobeRGBColor.get_upscaled_value_tuple(colour)
            return rgb_color
        im = Image.new('RGBA', (img_size, img_size), (0, 0, 0, 0))
        for x in range (0, img_size):
          for y in range (0, img_size):
            dist = abs(math.sqrt((x - img_half) ** 2 + (y - img_half) ** 2))
            if dist < inner_radius or dist > outer_radius:
              continue
            if x - img_half == 0:
              angle = -90
              if y > img_half:
                angle = 90
            else:
              angle = math.atan2((y - img_half), (x - img_half)) * 180 / math.pi
            angle = 360 - (angle) % 360
            A,B = ang_to_coord_colors(s, l, r, angle)
            color = hsl2rgb(angle, s, l)
            im.putpixel((x, y), color)
        im.save(const.GAMUT_PATH)
        return inner_radius, outer_radius
```