UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Repositorio Digital USM **https://repositorio.usm.cl**

Tesis USM TESIS de Pregrado de acceso ABIERTO

2018

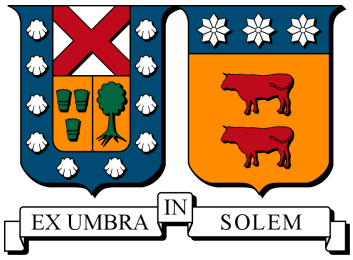
SISTEMA DE ESTIMULACIÓN VISUAL Y REGISTRO DE MOVIMIENTOS OCULARES PARA TAREAS SICOMOTORAS

WICHE LATORRE, CHRISTIAN ANDRES

http://hdl.handle.net/11673/41528

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



”SISTEMA DE ESTIMULACIÓN VISUAL Y REGISTRO DE MOVIMIENTOS OCULARES PARA TAREAS SICOMOTORAS”

CHRISTIAN ANDRÉS WICHE LATORRE

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO MENCIÓN CONTROL E INSTRUMENTACIÓN

PROFESOR GUÍA: **MARÍA JOSÉ ESCOBAR SILVA**
PROFESOR CORREFERENTE: **MATÍAS ZAÑARTU SALAS**

JUNIO - 2018

Para aquellos que siempre me han apoyado e inspirado
y nunca leerán este trabajo.

Agradecimientos

Este trabajo no hubiera sido posible sin el apoyo de un gran equipo.

En estas líneas plasmaré sus nombres y contribuciones, qué forman una parte vital de este trabajo.

En primer lugar quisiera agradecerle a mis profesores por extenderme este tema, instarme a buscar los recursos para armar el sistema y por sus detalladas correcciones que sin duda fueron un escalón importante para subir la calidad de este texto.

A mi amigo y colega Christopher Rozas y al departamento de IDP de la universidad por su generosidad y su apoyo que se hizo físico en el diseño e impresión 3D del apoya barbilla, una pieza fundamental del sistema.

A mi familia por su apoyo incondicional y su infinita buena voluntad para echarme más de una mano incluso en los momentos mas álgidos.

A mis hermanos de distintas madres por hacer épica esta aventura universitaria: Pedro, Rodrigo, Eduardo y Oscar.

Resumen

Este trabajo de título muestra la construcción e implementación de un sistema de estimulación visual y registro de movimiento ocular para uso en experimentos que evalúan tareas sicomotoras.

El estudio de las dinámicas del ojo y la capacidad de registrar sus movimientos ha permitido avances importantes en áreas diversas como entender el desarrollo de la enfermedad de Parkinson, estudiar la usabilidad de productos o la efectividad de la publicidad o entrenamiento en sistemas simulados, entre otros.

Cada experimento tiene requerimientos específicos, relacionados a las características técnicas de las tareas y a la forma en que se registran los parámetros de configuración de las mismas. El sistema desarrollado en este trabajo de título permite al usuario montar un experimento, ingresar los parámetros y definir tareas a través de una interfaz gráfica, sin requerir alterar el código del software.

El sistema propuesto incluye hardware (PC, monitores, *eye tracker* y apoya barbilla) y software para la adquisición de datos y la proyección del experimento.

Para demostrar el funcionamiento del sistema se incluye un ejemplo de aplicación, con instrucciones para realizar un experimento de muestra, ya sea mediante código o usando la interfaz gráfica, además de mostrar los resultados de este experimento.

Abstract

This final project shows the construction and implementation of a visual stimulation system and an ocular movement record to use in experiments that evaluate psychomotor tasks.

The study of eye dynamics and the capacity to record it's movements has allowed important advances in diverse areas such as undestanding the development of the Parkinson's decease, study the usability of products or the effectivity of advertisment or training in simulate systems, among others.

Each experiment have different requirements related to the technical characteristics of the tasks and the way in which the configuration parameters are recorded. The system developed in this final project allows the user to setup an experiment, enter the parameters and define tasks through a graphical user interface without altering the code or the software.

The proposed system includes hardware (PC, monitors, eye trackers and chin rest) and software for the acquisition of data and the proyection of the experiment.

To show how the system works an application example is included with instructions to run a sample experiment either through code or using the graphical user interface as well as showing the results of this experiment.

Glosario

Ejes visuales	Corresponde a la proyección de una línea recta que pasa simultáneamente por el centro de la fóvea, y la pupila.
Esclerótida	Sección blanca del ojo que rodea a la pupila
Eye tracker	Dispositivo utilizado para realizar seguimiento de los movimientos oculares
Fóvea	Área de la retina que permite la visión más nítida y detallada
Log	Se asigna este término a un registro secuencial de todos los acontecimientos que afectan a un proceso particular.
Retina	Capa fotosensible ubicada en la parte posterior del ojo encargada de transducir estímulos lumínicos en impulsos nerviosos
Setup	Se asigna esta denominación a cierta configuración de un espacio de trabajo

Siglas

API	Application Programming Interface (Interfaz de Programación de Aplicaciones)
CRT	Cathode Ray Tube (Tubo de Rayos Catódicos)
EOG	Electro-OculoGraphy (Electro-oclulografía)
FP	Fixation Point (Punto de Fijación)
FPS	Frames Per Second (Cuadros Por Segundo)
GUI	Graphical User Interface (Interfaz gráfica de Usuario)
IR	Infra-Red (Infra-Rojo)
ISI	Inter Saccadic Inteval (Intervalo Inter-Sacádico)
RT	Response Time (Tiempo de Respuesta)
SL	Saccadic Latency (Latencia Sacádica)
SO	Sistema Operativo
SSC	Scleral Search Coil (Bobina Escleral de Búsqueda)
TP	Target Point (Punto Objetivo)
VOG	Video-OculoGraphy (Oculografía basada en Video)

Índice general

Agradecimientos I

Resumen II

Abstract III

Glosario III

1. Introducción **1**
1.1. Motivación 1
1.2. Objetivos 2

2. Estado del arte **3**
2.1. Movimiento ocular 3
2.2. Métodos de captura de movimiento ocular 5
2.2.1. Antecedentes históricos 5
2.2.2. Tecnologías actuales 6
2.2.3. Comparación entre tecnologías 8
2.3. Métodos de estimulación visual 10
2.3.1. Hardware de estimulación 10
2.3.2. Software de estimulación 11
2.3.3. Experimentos de estimulación 12
2.4. Sistemas de estimulación y registro visual 15

3. Sistema propuesto **17**
3.1. *Setup* a utilizar 17
3.2. Diseño del sistema 18
3.2.1. Primera parte: Estructura de datos 18
3.2.2. Segunda parte: Implementación de las funciones principales 21
3.2.3. Tercera parte: Interfaz gráfica 39

4. Resultados **48**
4.1. Modo de uso 48
4.1.1. Usando la *API* 49
4.1.2. Usando la *GUI* 51
4.2. Comprobación de funcionamiento. 58

5. Conclusiones y trabajo futuro **60**
5.1. Conclusiones 60
5.2. Trabajo futuro 61

Referencias **62**

A. Lógica del script de ejecución **64**

B. Configuración del entorno de programación **65**

Índice de figuras

2.1. Estructura del ojo 3
2.2. Ejemplo de uso de SSG 6
2.3. Ejemplo de posicionamiento de electrodos para EOG 7
2.4. Principio de detección VOG en base a reflexión de luz 8
2.5. Muestra de *eye trackers* disponibles en el mercado con tecnología VOG basados en reflexión de luz IR 8
2.6. Tarea pro/anti sacádica. 13
2.7. Paradigmas de experimentación. 14
2.8. Tarea guiada por memoria. 15
2.9. *Setup* experimental típico 16
2.10. Diagrama general de la interfaz hombre-máquina 16

3.1. Diagrama de base de datos a implementar. 19
3.2. Diagrama de funcionamiento general. 22
3.3. Estructura de directorios para resultados. 22
3.4. Diagrama de clases del sistema (no considera *GUI*). 23
3.5. Diagrama de clases de la *GUI*. 39
3.6. Ventana principal. 40
3.7. Ventana para configuración de experimentos. 41
3.8. Ventana de copia para experimentos. 42
3.9. Ventana de configuración para secuencia de ejecución del experimento. . . . 43
3.10. Ventana para configuración de tareas. 43
3.11. Ventana para configuración de cuadros. 44
3.12. Ventana de configuración de componente. 45
3.13. Ventana para copia de elementos. 46
3.14. Ventana utilizada en el proceso de copia. 47
3.15. Ventana de información. 47

4.1. Pestaña de configuración. 51
4.2. Perfil de configuración propuesto. 51
4.3. Pestaña de experimentos. 52
4.4. Experimento propuesto con la primera tarea configurada. 52
4.5. Ventana de configuración de la primera tarea. 53
4.6. Configuración de los cuadros pertenecientes a la primera tarea. 53
4.7. Componentes utilizados en la configuración de la primera tarea. 54

4.8. Agregando la primera tarea a la secuencia de ejecución. 54
4.9. Experimento propuesto con la primera tarea configurada y en la secuencia de ejecución. 55
4.10. Ventana de configuración de la segunda tarea. 55
4.11. Configuraciones de la segunda tarea. 56
4.12. Experimento propuesto con configuración completa. 56
4.13. Resultado obtenido. 57
4.14. Aplicación lista para ejecutar *script*. 57
4.15. Datos registrados. 58

A.1. Funcionamiento general del *script* de ejecución. 64

Índice de cuadros

2.1. Comparativa de los sistemas de adquisición encontrados 9
2.2. Comparativa de software de estimulación 12

3.1. Hardware y software utilizado en el desarrollo. 17
3.2. Métodos implementados en `utils`. 26
3.3. Métodos implementados en la clase `SaccadeDB`. 27
3.4. Métodos implementados en la clase `Configuration`. 28
3.5. Métodos implementados en la clase `ItemList`. 30
3.6. Métodos implementados en la clase `ItemListSequence`. 31
3.7. Métodos implementados en la clase `Component`. 33
3.8. Métodos implementados en la clase `Frame`. 34
3.9. Métodos implementados en la clase `Test`. 36
3.10. Métodos implementados en la clase `Experiment`. 37
3.11. Métodos implementados en la clase `ExperimentHandler`. 38
3.12. Métodos implementados en la clase `ExperimentRuntime`. 38

Capítulo 1

Introducción

1.1. Motivación

Diariamente y sin prestar mayor atención, gran parte de la población utiliza sus ojos para interactuar con su entorno: se detienen a admirar el paisaje, leer un libro, revisar su teléfono, navegar en internet, verificar que sus alimentos se encuentran en buen estado, etc. A pesar de lo simple que puede parecer esta acción, en la realidad corresponde a un proceso sumamente complejo que involucra la participación de un sinnúmero de estructuras sensoriomotrices.

El simple hecho de orientar nuestra vista hacia un nuevo objetivo desencadena una serie de eventos fascinantes: El globo ocular rota hasta lograr posicionarse en una determinada dirección de forma tal que los rayos de luz que son reflejados por el punto de interés se proyectan en la retina, estructura que transduce esta información en impulsos eléctricos que son interpretados de forma posterior por el cerebro y que se traducen en información que percibimos como una imagen.

El estudio de las dinámicas del ojo y la capacidad de registrar sus movimientos ha permitido con el paso de los años avances importantes en áreas sumamente variadas como la detección y seguimiento de enfermedades, estudios de marketing y usabilidad, ayuda a personas con discapacidad, entrenamiento y detección de errores en sistemas simulados, defensa y seguridad, videojuegos y experiencia de usuario, entre otras.

La principal dificultad en los avances de todas las áreas recae no solo en la calidad del registro ocular tanto en precisión temporal como espacial, si no, en la capacidad de correlacionar estímulos visuales y respuestas motrices con el fin de detectar la relación causa/efecto de los eventos. Para lograr esto, es necesario el uso de sistemas que permitan sincronizar las etapas de estimulación y registro de respuesta de forma tal de entregar insumos que permitan

la obtención de información pertinente y útil del procesamiento posterior.

1.2. Objetivos

Así, el objetivo principal de este trabajo de título consiste en el diseño y construcción de un sistema de estimulación visual y registro de movimientos oculares enfocado en la realización de experimentos asociados a tareas sicomotoras. De esta forma, se presentan a continuación los objetivos específicos para el desarrollo:

(I) Definir y programar un mecanismo de estimulación visual acorde con las características técnicas de los protocolos utilizados en la realización de tareas sicomotoras.
(II) Implementar un sistema de sincronización entre el registro y la estimulación visual que permita la integración de un sistema de captura de movimiento ocular y que asegure el correcto registro de los datos.
(III) Integrar todas las etapas en una interfaz gráfica de usuario (*GUI*).

Capítulo 2

Estado del arte

2.1. Movimiento ocular

La acción de dirigir la mirada hacia un objeto (*fixation point*) es parte fundamental del proceso de visión. Este acto involucra el direccionamiento de los ejes visuales (ver figura 2.1) hacia un objetivo determinado, permitiendo la realización de análisis visuales precisos. Dicha orientación muchas veces implica movimientos coordinados de los ojos, cuello y cabeza, no obstante, existen movimientos más pequeños que son realizados únicamente por los ojos, conocidos como movimientos sacádicos [1, 2].

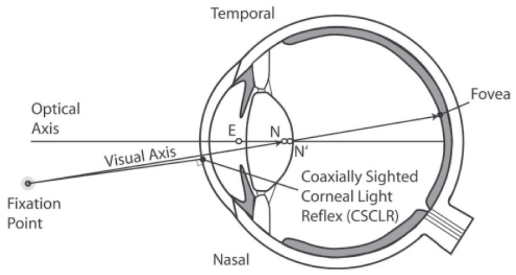


Figura 2.1: Estructura del ojo[1].

En la figura 2.1 se observa un esquema simplificado del ojo observado desde la parte superior de la cabeza. En ella pueden apreciarse las proyecciones del eje óptico (*optical axis*) y visual (*visual axis*) del ojo, que corresponden a líneas imaginarias que van desde el centro de la retina y fóvea respectivamente, pasando por la parte posterior del cristalino y extendiéndose luego a través de la pupila y córnea.

[1]Fuente imagen: J. T. Schwiegerling, "Eye Axes and Their Relevance to Alignment of Corneal Refractive Procedures", Journal of Refractive Surgery, vol 29(8), pp. 515-516, fig. 1-D.

Los movimientos sacádicos corresponden a las rotaciones que realiza el globo ocular entre dos momentos de posicionamiento estacionario, por tanto se traducen en el consecuente desplazamiento de la pupila. Dichos movimientos pueden ocurrir tanto en el eje horizontal como vertical y se encuentra aún en discusión [2, 3] si se entrega información visual en todo momento o solo es relevante al mantener la mirada detenida en alguna posición. Los movimientos sacádicos ocurren en todo momento y pueden estar provocados por procesos cognitivos conscientes o inconscientes. Independientemente de la naturaleza de estos movimientos, ambos generan patrones que permiten la construcción de un mapa mental de la escena observada. La necesidad de redireccionar el eje visual constantemente para analizar una imagen se relaciona con la idea de que, en el ojo humano, solo la parte central de la retina (fóvea) se encuentra encargada de la visión en alta resolución, pues presenta una alta concentración de células fotorreceptoras sensibles al color.

Existen dos tipos de movimientos sacádicos: Los voluntarios, también conocidos como provocados, que implican control consciente sobre los procesos cognitivos y los reflexivos o automáticos, que corresponden a la respuesta natural a la aparición de un nuevo estímulo visual.

Las características principales de dichos movimientos son:

1. Por su naturaleza los movimientos sacádicos se consideran balísticos, lo que quiere decir que la posición de destino no cambia durante el desarrollo del movimiento. Esto puede entenderse también como que el objetivo se encuentra predeterminado en el momento de partida.
2. La velocidad de las sacadas[2] aumenta de forma no lineal en la medida que aumenta la amplitud de movimiento y puede alcanzar magnitudes de hasta $600 - 700[\frac{grados}{s}]$. Además, la duración del movimiento puede fluctuar entre $20 - 120[ms]$, aunque en promedio solo dura de $20 - 40[ms]$.
3. La precisión del movimiento sacádico presenta un error que varía entre el $5 - 10\,\%$ de la amplitud total del movimiento. Las correcciones son realizadas por desplazamientos de calibración denominados micro-sacadas, que permiten además realizar tareas que requieren de gran presición visual [4]. Estos métodos correctivos permiten suponer que existe algún tipo de procesamiento paralelo encargado de la calibración ocular de largo plazo [2].

Las características expuestas entregan, a grandes rasgos, nociones que permiten comprender el por qué su estudio se ha vuelto común en campos científicos como la neurociencia:

[2]Forma de referirse a un movimiento sacádico.

Dado que los movimientos del globo ocular son caracterizables, de patrones definidos y de alta precisión es posible identificar mediante ellos enfermedades cuyos síntomas se traduzcen en alteraciones de las capacidades motrices y cognitivas de zonas que decidan el punto de destino de futuras sacadas. Un ejemplo de esto es la enfermedad de Parkinson, donde una afección crónica a los ganglios basales produce una reducción progresiva de la sustancia negra lo que se traduce en una producción insuficiente de dopamina, neurotransmisor presente en diversas áreas del cerebro y que es especialmente relevante para la función motora. Esta insuficiencia se traduce en aumentos en los tiempos de respuesta y tasas de error en diversas tareas asociadas a movimiento ocular [5, 6, 7, 8] (ver 2.3.3).

2.2. Métodos de captura de movimiento ocular

2.2.1. Antecedentes históricos

Para poder registrar los movimientos oculares es necesario el uso de equipamiento especializado que, por su función y sin importar la tecnología utilizada, se denomina por su nombre en inglés: *eye tracker*. A modo introductorio se presenta a continuación una pincelada de su desarrollo en la historia [9, 10]

La primera aparición de dispositivos de este tipo data de finales del siglo XIX [11, 12] y permitieron objetivar las investigaciones de comportamiento existentes. Sus primeras versiones consistieron en sistemas sumamente invasivos donde alambres finos conectados a una especie de lente de contacto movían una serie de palancas que amplificaban el movimiento y lo registraban en papel. Por su construcción, dichos dispositivos permitían observar el comportamiento espacial, mas no el temporal.

A principios del siglo XX comenzaron a desarrollarse sistemas más parecidos a las tecnologías actuales de la mano de técnicas no invasivas basadas en óptica y reflexión de luz: La proyección de luz sobre la córnea genera reflejos que se mueven de forma similar a la pupila. Si se hace posible su registro, es posible conocer el movimiento ocular tanto horizontal como vertical, más no rotatorio [13]. Este método revolucionario marcaría los desarrollos futuros en esta área.

En la década de los 70', gracias a los avances en sistemas de grabación de vídeo y procesamiento digital, se hizo posible detectar electrónicamente características del movimiento en base al contraste existente entre la esclerótida y los bordes del iris. Debido a los efectos de sombra producidos por los párpados, este método presentaba problemas para detectar movimientos verticales, no obstante, permitía registrar movimiento horizontal con buena calidad.

De forma posterior y en base a estos avances iniciales fueron desarrolladas las tecnologías que, cada vez más, permiten obtener información relevante sin producir daño sobre quienes forman parte de los experimentos, reduciendo así las limitantes en este campo de investigación.

2.2.2. Tecnologías actuales

En la actualidad existen una gran gama de tecnologías para registrar movimiento ocular [9, 10, 14]. A continuación se indican las más relevantes:

1. **Bobina escleral magnética (SSC):** Esta técnica requiere del uso de lentes de contacto de gran tamaño directamente sobre el globo ocular. Dicho lente posee dos pequeñas bobinas de alambre que, al ser alineadas con el eje de visión e inducidas por campos electromagnéticos externos de alta frecuencia, permiten obtener información sobre la dirección en la que se encuentra el ojo en forma de voltaje. A pesar de la gran incomodidad que producen (ver figura 2.2), esta técnica es una de las más precisas y exactas.

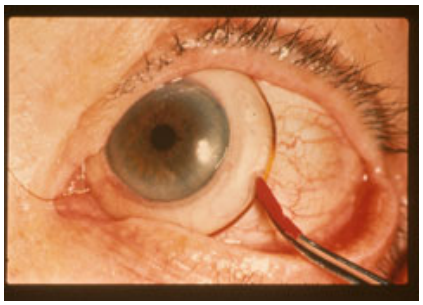


Figura 2.2: Ejemplo de uso de SSG[3].

2. **Electro-OculoGrafía (EOG):** Este método de seguimiento ocular permite determinar la dirección en la que se encuentra direccionado el ojo en base a la medición de diferencias de potencial eléctrico en la piel. En la medida que el ojo rota, el dipolo producido por la córnea y la retina cambia su orientación, lo que se ve reflejado en los potenciales de las zonas aledañas. Sus ventajas principales, además del bajo costo, son su buena precisión temporal y la capacidad de medir el movimiento ocular incluso aunque los ojos se encuentren cerrados (lo que hace de este método una herramienta interesante en casos de estudio de sueño) y que las mediciones son relativas a la posición de la cabeza. No obstante lo anterior, la precisión y exactitud de las mediciones obtenidas es

[3]Fuente imagen: http://www.dizziness-and-balance.com/practice/default.htm

baja ya que se encuentran sujetas a artefactos como el movimiento de los párpados. La figura 2.3 muestra el posicionamiento típico de los electrodos en este tipo de *setup*.

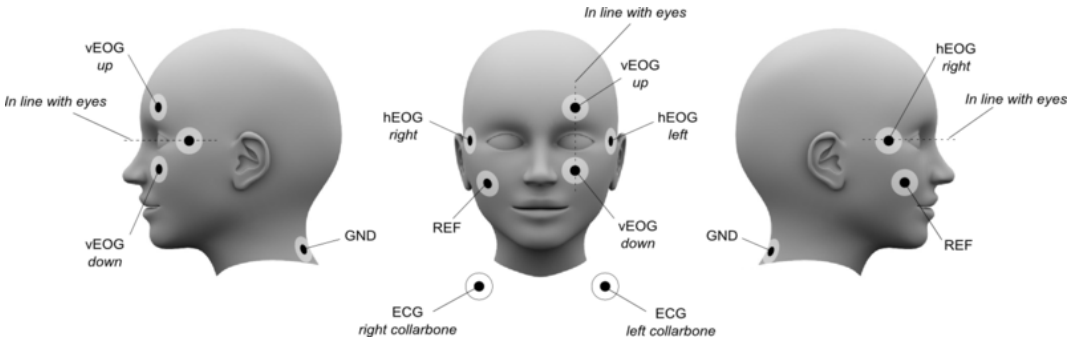


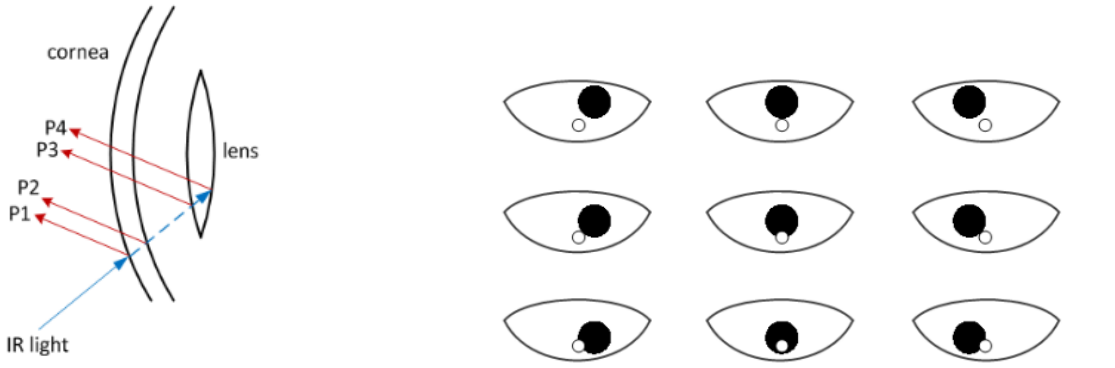
Figura 2.3: Ejemplo de posicionamiento de electrodos para EOG[4].

3. **Seguimiento ocular basado en vídeo (VOG):** Este método de seguimiento ocular es el más popular en la actualidad. Consiste en capturar, con una o más cámaras, la actividad ocular y mediante procesamiento de imágenes determinar en que dirección se encuentra orientada la mirada. El procesamiento puede ser dividido en dos etapas principales: en la primera se detecta y localiza el ojo en la imagen, para lo que típicamente se utiliza la pupila o el iris y la segunda etapa corresponde al proceso por el cual se estima hacia donde se encuentra dirigido.

   Dentro de las técnicas VOG existentes la más utilizada corresponde a la estimación mediante reflexión corneal (con una fuente de luz típicamente infrarroja (IR)) y detección de pupila. Los focos de luz se ubican de forma que siempre se encuentren en la misma posición respecto del ojo, con lo cual la reflexión se mantiene virtualmente siempre en el mismo lugar. De esta forma la estimación se genera en base a la diferencia entre la ubicación de centro de la pupila y la posición de la reflexión.

   El principio de funcionamiento asociado a este método se basa en las imágenes de Purkinje-Sanson que describen la existencia de al menos 4 reflexiones de luz producidas por la córnea y el cristalino (figura 2.4 (a)) que pueden ser utilizadas como referencia para estimar la posición del ojo. En la figura 2.4 (b) es posible observar la primera reflexión de Purkinje en relación a distintas posiciones de la pupila.

[4]Fuente imagen: http://sbiwiki.cns.ki.se/mediawiki/index.php/Natmeg/checklist_MEG_preparation



(a) Reflexiones de Purkinje-Sanson (b) Posición de la reflexión P1 relativa al centro de la pupila.

Figura 2.4: Principio de detección VOG en base a reflexión de luz [14].

La presentación de estos dispositivos es variada y van desde cascos y lentes hasta trípodes y columnas donde se integran un elemento apoya-barbilla y el *eye tracker*, como en la figura 2.9. En la figura 2.5 se muestra un par a modo de referencia.

(a) Tobii Pro Glasses 2 (b) MiraMetrix S2

Figura 2.5: Muestra de *eye trackers* disponibles en el mercado con tecnología VOG basados en reflexión de luz IR[5].

2.2.3. Comparación entre tecnologías

En la literatura consultada [9, 10, 14] se entregan nociones sobre las capacidades operativas que se resumen en la tabla 2.1 para EOG, SSG, VOG (común) y DPI (VOG de gama alta utilizando distintas reflexiones de Purkinje).

En esta, es posible evidenciar a grandes rasgos la cantidad de parámetros a considerar en el proceso de elección de una tecnología particular: Debe existir un compromiso entre la comodidad para el usuario, las prestaciones tanto en precisión temporal como espacial,

[5]Fuente imagen: https://www.tobii.com/ y https://www.microway.com.au/ respectivamente.

dificultad para obtener una buena calibración, etc. De forma tal que permitan la ejecución de cierto tipo de experimentos.

Si lo que interesa medir corresponde al efecto de, por ejemplo, las micro-sacadas en el proceso visual, la principal preocupación se centra en obtener una alta precisión espacial y temporal con el fin de caracterizar correctamente estos movimientos. Por otro lado, si lo que interesa estudiar son los patrones de comportamiento de usuarios para un estudio de usabilidad[6] en un sitio web, estas variables no son tan importantes como la comodidad del usuario, ya que sus acciones deben corresponder lo más posible a una situación común, con el fin de reflejar correctamente sus intereses.

	EOG	SSG	VOG	DPI
Precisión espacial (grados)	$\approx 0{,}5$	$\approx 0{,}01$	$\approx 0{,}05$	$\approx 0{,}017$
Precisión temporal[7](Hz)	40	500	$50 - 400$	$500 - 1000$
Registro de mov. verticales	Es posible pero se encuentran sujetos a error por efecto de artefactos producidos por el párpado	Si	Si	Si
Registro de torsión	No	Si	Si	Si
Tiempo de *setup*	Lento debido a que requiere la utilización de electrodos	Lento	Rápido	Rápido
Requiere calibración por enfoque	Si	No	Si	Si
Complejidad de calibración	Requiere configuración bitemporal	La no-linealidad puede ser compensada con un modelo basado en ajuste de parámetros	Buena linealidad	Buena linealidad
Invasividad	Electrodos cercanos al ojo (sin contacto), no afecta el campo visual	Lentes de contacto, posibles efectos negativos en precisión visual, incomodidad	Aparato montado en la cabeza (sin contacto con los ojos), limitación moderada del campo visual	Cabeza inmovilizada en un soporte de barbilla, limitación moderada del campo visual

Cuadro 2.1: Comparativa de los sistemas de adquisición encontrados en [9, 10, 14].

[6]Estudios enfocados en caracterizar la experiencia de usuario e identificar el comportamiento del mismo en alguna plataforma (pagina web, aplicación, etc.).

2.3. Métodos de estimulación visual

2.3.1. Hardware de estimulación

Debido a que la presentación de resultados de investigación científica requieren de que los datos obtenidos sean comprobables y los experimentos reproducidos, es que se hace necesario declarar de forma precisa las características de los estímulos, métodos utilizados para adquirir datos y técnicas aplicadas en el procesamiento posterior. Por lo cual, la elección del artefacto de estimulación es una tarea particularmente sensible [15].

A lo largo de la historia, los investigadores dedicados al estudio del movimiento ocular han usado diversas tecnologías con el propósito de estimular visualmente a sus pacientes, no obstante, no fue hasta la década de los 70' y de la mano de los computadores, que los monitores CRT revolucionaron este campo de investigación convirtiéndose en el estándar durante décadas. El motivo principal de su popularidad dice relación con la capacidad que brindan, mediante la programación en computador, de diseñar con relativa facilidad una gran gama de pruebas y experimentos distintos, lo que permitió explorar de forma rápida nuevos métodos e hipótesis. Esto, complementado a la integración del control de los parámetros de estimulación y almacenamiento de resultados e historiales en el mismo dispositivo permitió robustecer los procesos de investigación debido a la capacidad de repetir los experimentos sin afectar las características de los estímulos.

Las limitantes de los primeros dispositivos se fueron subsanando con el avance de la tecnología, de esta forma, los monitores análogos avanzaron hasta alcanzar tasas de refresco elevadas ($\geq 200[Hz]$), una gran gama de colores, buena resolución espacial (alcanzando hasta $1600[px]$ de ancho), un rápido decaimiento del fósforo de la pantalla que se traduce en tiempos de repuesta reducidos ($< 1[ms]$) y un buen tamaño (típicamente $20[pulg]$ en la diagonal). En base a esta información pueden definirse dos conceptos relevantes para el *hardware* de estimulación:

1. **Tasa de refresco (FPS):** Dice relación con la cantidad de veces que se actualiza la imagen de la pantalla por cada segundo. Influye en el *timing* de los estímulos.
2. **Tiempo de respuesta (RT):** Es cuanto demora un pixel de la pantalla en cambiar su color de blanco a negro. Influye en la calidad y nitidez de las imágenes.

A pesar de la mejoras considerables en sus características, las nuevas tecnologías han hecho desaparecer a los monitores CRT del mercado, siendo más comunes ahora monitores

[7] Frecuencia de muestreo.

LCD, LED y oLED que tienen pantallas de mayor tamaño, menor consumo de energía, menor radiación electromagnética y una menor huella de carbono. Es importante destacar que, a pesar de que el aspecto de las nuevas tecnologías es similar, sus principios de funcionamiento, capacidades y características difieren.

Existen varias consideraciones que hacer al utilizar estas tecnologías [16, 17]. Transversalmente se tiene un problema de *timing* debido a las bajas tasas de refresco de la mayor parte de los monitores modernos, lo que vuelve una tarea no trivial el cumplir con los requerimientos de los estímulos y evitar que el usuario perciba una falta de fluidez en la presentación, lo que podría producir que distraiga su atención. En este sentido, por ejemplo, aunque muchos monitores modernos indican que su tasa de refresco se encuentra entre $60 - 75[Hz]$, no se aclara en sus hojas de datos cual es el límite efectivo. Este punto se vuelve crítico si se considera que una buena parte de los equipos modernos incorporan sistemas de procesamiento de imágenes para mejorar la calidad, lo que retrasa aún mas estos tiempos.

Otro elemento de cuidado es el tiempo de respuesta. Si este es elevado, producirá un efecto de halos o rastros de luz cuando las imágenes se mueven con rapidez, por esto, es ideal asegurar que este sea reducido para lograr que el cambio entre imágenes no sea notorio y afecte el experimento (esto se hace más presente en casos en los que se muestra secuencias de vídeo).

Finalmente, es importante compaginar las características del experimento con las propiedades lumínicas del equipo ya que es sabido que en tecnologías como la de los monitores LCD tanto el ángulo del observador respecto de la pantalla como las distintas zonas de la misma afectan el color/contraste observado (efecto de retro-iluminación).

2.3.2. Software de estimulación

Tal como se indicó en el apartado anterior la generación de estímulos es una pieza clave en la realización de experimentos ya que permiten preparar un escenario adhoc para la obtención de datos específicos. En el sitio web de Hans Strasburger [18] es posible encontrar una larga lista de software especializado para el desarrollo de experimentos en el área de la sicofísica además de referencias e información sobre sus características. A modo de resumen se presenta en el cuadro 2.2 una comparación entre algunas de las aplicaciones indicadas en la página[8].

[8]Esta selección fue realizada en base al número de citas para cada aplicación en Google Scholar.

	PsychoPy	VissionEgg	PsychoToolbox	Stimulus Presentation
Tipo de software	Open source			Privativo, de pago
Plataforma	python + OpenGL		Matlab/Octave	Software independiente (IDE con editor de python)
Sistema operativo	Linux, MacOS, Windows			Windows
Fecha de última actualización	$Dec/2017$	$Sep/2014$	$Oct/2017$	$Abr/2017$
Citaciones en Google Scholar	2220	427	6310	3520
Programable	Si			
Imágenes y Vídeo	Si			
Sonido	Si			
Soporte para *Eye trackers* incluido	Si	No	Si	Si
Capacidad para registro de data	Si	No	Si	Si
Documentación/Foro disponible	Si	No (Link caído)	Si	Si

Cuadro 2.2: Comparativa de software de estimulación [19, 20, 21, 22].

2.3.3. Experimentos de estimulación

Debido a la versatilidad de los movimientos sacádicos, se ha desarrollado con el tiempo un número importante de tareas sicomotoras para probar distintos mecanísmos cognitivos. Por este motivo y a modo de acotar las actividades asociadas a este trabajo de título, se desarrollará el sistema propuesto de forma tal que permita solo la implementación de experimentos con características similares a los presentados en [5, 6, 7, 8, 23], que corresponden a métodos utilizados en la evaluación y caracterización de desempeño sicomotor en pacientes que padecen la enfermedad de Parkinson. Dichas tareas corresponden a la evaluación de movimientos pro-sacádicos, anti-sacádicos y aquellos guiados por memoria.

En la explicación de estos experimentos se utilizarán dos elementos importantes. El primero corresponde al punto de fijación o FP y el segundo al punto objetivo o TP. FP es un punto ubicado en el centro de la estimulación a modo de referencia y se utiliza para calcular las amplitudes de los movimientos oculares. TP es un punto objetivo que sirve como guía en la realización de los movimientos.

Métricas de interés

Las métricas principales a considerar para este tipo de experimentos son:

1. **Latencia sacádica (SL):** Lapso de tiempo que transcurre entre la aparición de un nuevo estímulo y el comienzo de una sacada $[ms]$.
2. **Intervalo inter-sacádico (ISI):** Tiempo entre el término de un movimiento sacádico y el comienzo de otro en $[ms]$.
3. **Ganancia inicial de movimiento:** Corresponde a una medida de precisión del movimiento ocular. Relaciona la amplitud efectiva de la sacada realizada y la amplitud existente entre el punto de inicio y el punto de destino.
4. **Tasa de error:** Relación entre el número de veces que el movimiento ejecutado fue errado y el total de movimientos realizados.

Movimientos pro/anti sacádicos

Aunque se parecen en estructura, el enfoque de estas tareas es completamente distinto. La medición de la respuesta prosacádica tiene como fin el cuantificar la capacidad del individuo para responder de forma reflexiva frente a un nuevo objetivo y entrega información que suele utilizarse como base para comparar otros métodos. Por otro lado, la actividad antisacádica implica la inhibición del comportamiento reflexivo y la realización de un movimiento voluntario en la dirección opuesta, de esta forma la información obtenida permite conocer el estado de estructuras cerebrales como el lóbulo frontal, en donde se procesan las funciones ejecutivas.

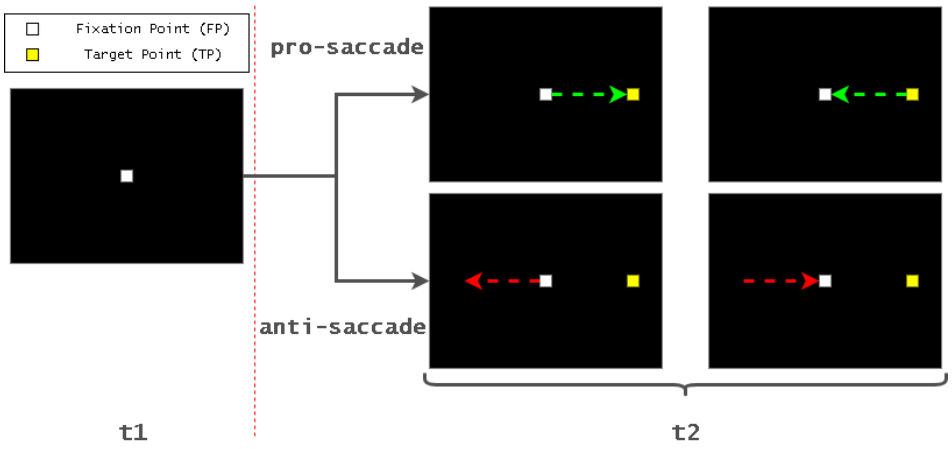


Figura 2.6: Tarea pro/anti sacádica.

Los tipos de movimientos descritos se estudian mediante estímulo similares al propuesto en la figura 2.6. Usualmente se pide al sujeto que mantenga su vista en FP (blanco) durante

algún tiempo t_1 hasta que aparezca TP (amarillo) y luego realice la tarea solicitada, para lo cual se otorga un tiempo t_2. En el caso de los movimientos prosacádicos, como ya fue explicado con anterioridad, se debe rotar los ojos hacia TP y luego volver a FP (flechas verdes). Para movimientos antisacádicos se realiza la acción contraria (flechas rojas). En algunos casos resulta conveniente añadir una imagen de *feedback* al final del experimento para indicar si la tarea fue realizada correctamente.

Estos experimentos pueden ser modificados y complementados con el fin de estudiar actividades cognitivas más complejas. Típicamente estas variaciones aplican uno o más paradigmas como los que se muestran en la figura 2.7, donde puede apreciarse un diagrama de temporización de un estímulo bajo el efecto de los paradigmas de *gap*, *overlap* y *delay*.

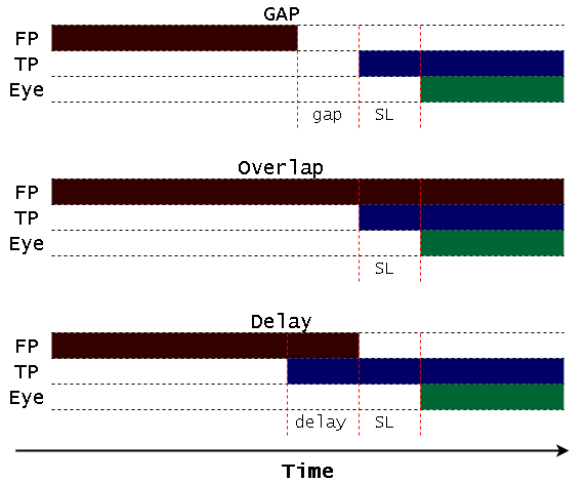


Figura 2.7: Paradigmas de experimentación.

Los paradigmas de estimulación permiten analizar el desempeño en distintos escenarios. En el caso de *gap* y *overlap* se incluyen distractores que corresponden a un espacio de tiempo sin estímulo entre FP y TP y a mantener FP durante la presentación de TP respectivamente. En el caso de *delay* se incluye la dificultad de esperar a que FP desaparezca antes de realizar la tarea asociada a la aparición de TP.

Movimientos guiados por memoria

Los experimentos guiados por memoria corresponden a tareas en las cuales el usuario debe repetir con movimientos oculares algún patrón observado con anterioridad.

En la figura 2.8 es posible observar un diseño simple para este tipo de experimentos. Después de mantener FP (blanco) por un lapso t_1 en pantalla se muestra una serie de objetivos T_n (amarillo) en intervalos de tiempo idénticos, luego se muestra nuevamente TP por un

lapso t_3 para indicar que ha finalizado la serie y dar tiempo al usuario para prepararse. Finalmente se solicita que se muevan los ojos en secuencia por los puntos en que se vio aparecer los objetivos (flechas rojas).

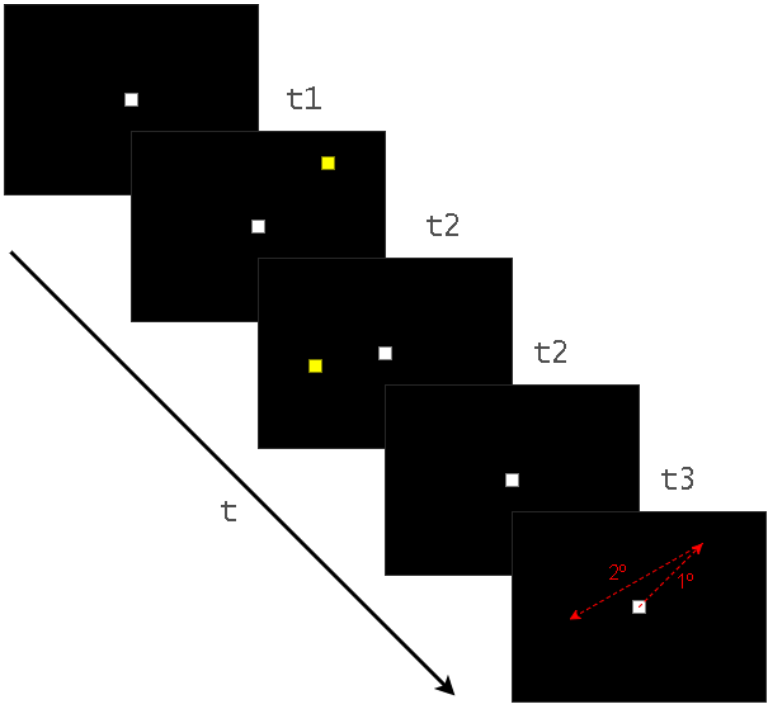


Figura 2.8: Tarea guiada por memoria.

2.4. Sistemas de estimulación y registro visual

A continuación se muestran los elementos principales requeridos para la configuración y puesta en marcha de un sistema de estimulación visual y registro de movimiento ocular según lo propuesto por Scott MacKenzie [24].

En la figura 2.9 se presenta a modo de ejemplo un *setup* típico en estudios de movimiento ocular. Su configuración consiste en un monitor LCD, un *eye tracker* de trípode que en este caso se encuentra montado en un apoya-barbilla y un teclado en el caso de requerir interacción con el usuario. El apoya-barbilla cumple dos funciones sumamente importantes: Mantiene al usuario a una distancia determinada de la pantalla, lo que permite generar estímulos en un rango visual específico y ayuda al paciente a no mover su cabeza durante el experimento, con lo cual se evitan distorsiones en las mediciones realizadas por el *eye tracker*.

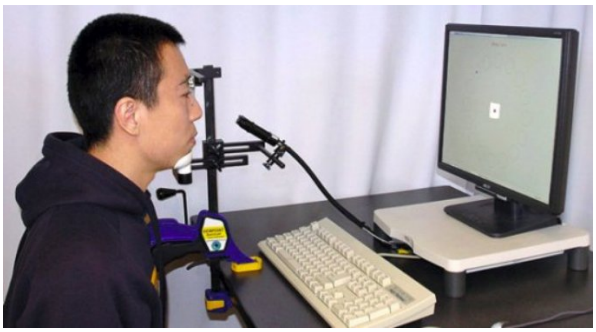


Figura 2.9: *Setup* experimental típico [24].

En la figura 2.10 se pretende explicitar la función del sistema de estimulación - adquisición - registro y los requerimientos del mismo. El software asociado al sistema debe ser capaz de manejar la interfaz con el usuario para poder entregar estímulos (visuales principalmente) y recibir las respuestas asociadas (en forma de datos de posicionamiento ocular o respuestas solicitadas por pantalla) de forma de agrupar las acciones temporalmente, permitiendo de esta forma dar sentido a los datos obtenidos en el experimento.

Figura 2.10: Diagrama general de la interfaz hombre-máquina [24].

Capítulo 3

Sistema propuesto

3.1. *Setup* a utilizar

Debido a que este trabajo de memoria corresponde fundamentalmente a una prueba de concepto se decidió junto al profesor guía evitar en lo posible la compra de nuevo equipamiento y por tanto no se entrega particular importancia a cumplir de forma estricta con las características de *hardware* utilizado en instancias formales de investigación. Con esto en consideración se presenta a continuación una descripción del entorno de trabajo utilizado:

Hardware	PC	Procesador:	Intel Core i7-3770
		Memoria RAM:	16,0$[GB]$, DDR3, 650$[MHz]$
		Disco duro:	Samsung SSD, 840 Series, 120$[GB]$
		Tarjeta gráfica:	NVIDIA GeForce GTX 660 Ti, 2$[GB]$
	Monitores	Principal:	Samsung SyncMaster, $1680 \times 1050[px]$, 60$[Hz]$
		Secundario:	Samsung SyncMaster, $1680 \times 1050[px]$, 60$[Hz]$
	Adquisición	*Eye tracker*:	EyeTribe
	Otro	Apoya-barbilla:	Hecho a medida[1].
Software	Entorno	SO:	Windows 10 Pro x64
		Base:	Python 2.7 x86, Anaconda
	Módulos	Principal:	PsychoPy 1.84.2 [25]
		Requeridos:	Ver anexo B

Cuadro 3.1: Hardware y software utilizado en el desarrollo.

[1] Diseñado y construido por Christopher Rozas con colaboración del departamento de IDP de la Universidad Técnica Federico Santa María.

La elección de software se encuentra influenciada fuertemente por los siguientes puntos:

1. El *eye tracker* a utilizar solo tiene disponible sus *drivers* para Mac y Windows. Debido a la facilidad de encontrar computadoras con Windows se decanta por este SO.
2. Los desafíos del sistema a implementar requieren del uso de un lenguaje orientado a objetos. A pesar de las bondades de Octave o Matlab en procesamiento y acceso a módulos complementarios estos entornos carecen de buen soporte para este tipo de programación, por este motivo se utiliza Python como lenguaje de desarrollo.
3. Dentro de las opciones disponibles PsychoPy resulta sumamente interesante. Cumple con ser *open-source*, es ampliamente utilizado por investigadores, presenta documentación detallada y clara para todas sus funciones y los foros se encuentran activos. Otro punto interesante es que entrega soporte para varias marcas de *eye tracker*, entre los cuales se encentra el seleccionado.

3.2. Diseño del sistema

El sistema a ser desarrollado en este trabajo de título tiene como función principal facilitar el proceso de configuración y puesta en marcha de experimentos asociados a movimiento ocular. Para esto, se considera oportuno subdividir el desarrollo en tres partes: La primera consiste en determinar la estructura de datos requerida para almacenar tanto las configuraciones del sistema como de las tareas a implementar. La segunda implica la implementación de los métodos de configuración y ejecución del experimento, asegurando su correcto funcionamiento. La tercera etapa corresponde a montar estas funciones en una *GUI* para facilitar el proceso a personas que no tengan conocimiento del lenguaje. Así, se presenta a continuación el trabajo realizado.

3.2.1. Primera parte: Estructura de datos

Al diseñar la estructura de datos se debió considerar el problema desde tres aristas. La primera correspondía a la necesidad de almacenar la información del conjunto compuesto por experimento-tareas-cuadros-componentes donde, complementando lo expuesto en el apartado 2.3.3, las tareas corresponden a las actividades específicas que el paciente debe realizar. Los cuadros corresponden a los elementos que conforman dichas tareas y los componentes serán considerados como las figuras e imágenes contenidas en cada cuadro. La segunda dice relación con la información complementaria que se desea incluir para posterior análisis, tal como comentarios del investigador encargado del experimento para una sesión específica o

algunas características de quien realizará el experimento, tales como su edad, sexo, color de ojos o si se utilizan lentes durante la ejecución. Finalmente, la tercera arista implica el conservar las configuraciones del equipo y *hardware* a utilizar para montar el servicio de ejecución asociado al módulo *ioHub*, que es el componente de *PsychoPy* que permite sincronizar todos los dispositivos asociados a la ejecución del experimento: teclado, pantalla, *eye tracker*, etc. y almacenar los datos en un archivo.

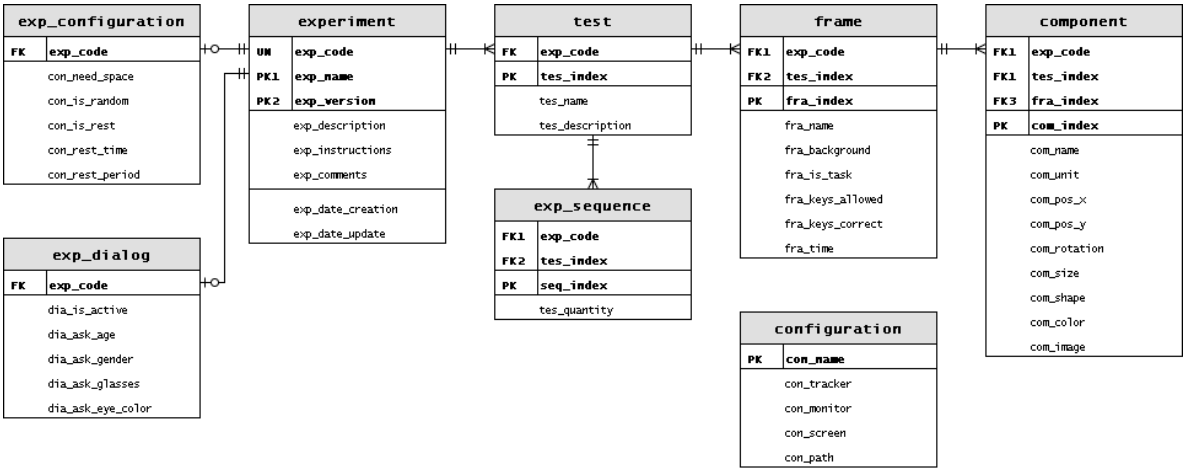


Figura 3.1: Diagrama de base de datos a implementar.

En base a lo anterior, se propone como solución el uso de una base de datos local de tipo *SQLite* con la estructura presentada en la figura 3.1. A continuación se comenta el contenido de cada tabla propuesta y su función:

1. **`configuration`:** Almacena la información asociada a un perfil de configuración. Cada perfil se identifica con un nombre (`con_name`) e indica qué monitor será utilizado (`con_screen`), cuál es su configuración[2] (`con_monitor`), el *eye tracker* a utilizar (`con_tracker`) y el directorio en que serán almacenados los resultados (`con_path`).
2. **`experiment`:** Almacena la configuración de un experimento. Cada experimento se identifica por un código único (`exp_code`) y un par nombre / versión (`exp_name` / `exp_version`), que tiene como objetivo identificar a una aplicación específica de una familia del mismo tipo, por ejemplo: (”antisaccade” / ”v1.0”). Además, se incluye un campo para indicar una descripción (`exp_description`), instrucciones para el usuario (`exp_instructions`) y comentarios para quien tome el experimento (`exp_comments`). Se incluyen la fecha de creación y la última modificación para dejar de forma explícita

[2]Los parámetros del monitor se definen en un aplicativo especializado de *psychoPy*.

una señal de cuidado: Las condiciones del experimento pueden no ser las mismas de la última ejecución.

3. **`exp_configuration`:** Esta tabla es complementaria al experimento y almacena la configuración general de ejecución. ¿Es necesario que el paciente presione la tecla espacio antes de cada tarea? (`exp_need_space`), ¿El orden de las tareas es aleatorio o secuencial? (`exp_is_random`), ¿Se incluyen en el experimento tiempos de descanso? (`exp_is_rest`), ¿Cada cuántas tareas? (`exp_rest_period`), ¿Con qué duración? (`exp_rest_time`).
4. **`exp_dialog`:** Esta tabla es complementaria al experimento y almacena la configuración del diálogo inicial, que permite obtener información adicional sobre el paciente. ¿Es necesario incluir información complementaria? (`dia_is_active`), ¿Qué edad tiene el paciente? (`dia_ask_age`), ¿Cuál es su sexo? (`dia_ask_gender`), ¿Utiliza gafas o lentes de contacto? (`dia_ask_glasses`), ¿Cuál es su color de ojos? (`dia_ask_eye_color`).
5. **`test`:** Almacena, para cada experimento, la identificación de las tareas que lo componen ordenadas por un índice (`tes_index`). Cada tarea se identifica por su nombre (`tes_name`), que es único para cada experimento, y un campo para añadir su descripción (`tes_description`).
6. **`exp_sequence`:** Almacena, para cada experimento, la secuencia de tareas a ejecutar. Cada ítem incluye un identificador del experimento (`exp_code`) y la tarea correspondiente (`tes_index`), además de un índice que permite definir el orden de ejecución (`seq_index`) y el número de veces que debe ejecutarse la tarea (`seq_quantity`).
7. **`frame`:** Esta tabla contiene, para cada tarea, el conjunto de cuadros que la conforman ordenados por aparición mediante un índice (`fra_index`). Cada cuadro se identifica por un nombre (`fra_name`), único para cada tarea, y permite la configuración de su comportamiento y características de forma general: color de fondo (`fra_color`), si el cuadro corresponde a una tarea que requiere respuesta de teclado o es temporizada (`fra_is_task`), las teclas habilitadas (`fra_keys_allowed`) y las teclas que se espera sean presionadas (`fra_keys_correct`) en el primer caso o el tiempo por el cual debe ser presentado (`fra_time`) en el segundo.
8. **`component`:** Esta tabla contiene, para cada cuadro, el conjunto de componentes o elementos que lo conforman ordenados por presentación mediante un índice (`com_index`). Cada componente se identifica por un nombre (`com_name`), único para cada cuadro, y las configuraciones asociadas tanto a las unidades de medida a utilizar (`com_units`), su posición en la pantalla ((`com_pos_x`), (`com_pos_y`)), su tamaño (`com_size`), si se encuentra rotada o no (`com_rotation`), en el cao de ser figura su forma (`com_shape`) y co-

lor (`com_color`) o si es una imagen la información binaria correspondiente (`com_image`).

Este modelo se considera apropiado por los siguientes motivos:

1. Aísla un experimento de otro al tener tareas y configuraciones completamente independientes. Esta configuración permite evitar que cambios en alguna tarea específicas afecte a mas de un experimento.
2. No puede existir una tarea que no se encuentre asociada a un experimento, lo que evita almacenar datos innecesarios (esto aplica también a cuadros y componentes).
3. Tener todas las configuraciones almacenadas en un archivo facilita la portabilidad y migración desde un equipo a otro. Además, dado que es tratable como una base de datos convencional facilita el proceso de revisión de los datos sin necesidad de contar con el programa principal.

Cabe destacar que, para asegurar la limpieza de la base de datos se tomo la decisión de definir las operaciones de *update* como *delete* en cascada a todas las tablas asociadas a un experimento. Esto implica que al eliminar un experimento particular también se eliminan todos los datos dependientes de él, tales como sus tareas, cuadros y componentes, evitando mantener data residual. Cabe destacar que esto se define también en niveles más bajos: Remover una tarea elimina todos sus cuadros, si es un cuadro se eliminan todos sus componentes.

3.2.2. Segunda parte: Implementación de las funciones principales

Para lograr sincronizar la ejecución del experimento con el uso de dispositivos tales como el monitor, teclado y *eye tracker* se hará uso de *ioHub* [26]. Esta herramienta, que forma parte de los módulos de *psychoPy*, tiene por característica principal la capacidad de montar un servicio que permite realizar una revisión constante de los dispositivos de entrada/salida configurados previamente para el experimento, almacenando la información recopilada de forma ordenada en un archivo con formato *hdf5*. Este tipo de archivos es considerado estándar en investigación y, a grandes rasgos, corresponde a un tipo *log* de datos organizado a modo de diccionario. Utilizando esta aplicación se plantea formular un sistema que permita las funcionalidades presentadas en la figura 3.2.

La aplicación propuesta tiene por requerimientos tanto la configuración del experimento (`Experiment`) como la del perfil de configuración (`Configuration Profile`), explicados en la sección 3.2.1.

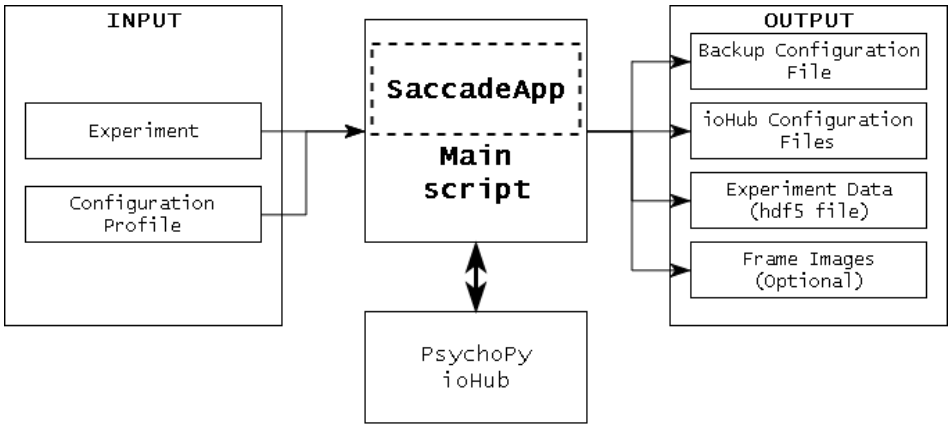


Figura 3.2: Diagrama de funcionamiento general.

Los resultados esperados son:

1. Creación de una estructura de carpetas que permita almacenar la información de las distintas instancias de ejecución de forma organizada y clara.

   Para lograr esto se propone el sistema de la figura 3.3 que consiste en separar los resultados por tipo de experimento y por versión. De esta forma, en el directorio base determinado en el perfil de configuración (`Base_Folder`) se crean carpetas con los nombres de los experimentos ejecutados (`Experiment_Name`) y dentro de estas se organizan los resultados en carpetas que indican la versión del experimento (`Experiment_Version`) y su código (`Experiment_Code`).

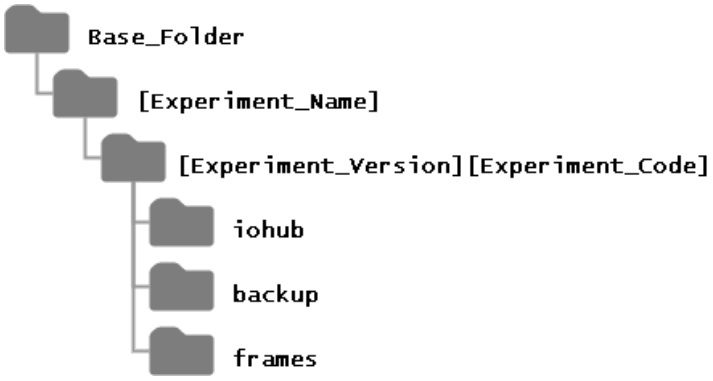


Figura 3.3: Estructura de directorios para resultados.

2. Para facilitar el proceso de revisión y posterior *debug* de los experimentos realizados se decide almacenar en la carpeta de resultados 3 archivos: El primer archivo, a almacenar en la carpeta `backup` consiste en un diccionario que contiene todas las configuraciones asociadas al experimento y perfil utilizados (`Backup Configuration File`). Los

siguientes 2 archivos, a almacenar en la carpeta `ioHub`, corresponden a las configuraciones del experimento y *hardware* en el formato requerido por *ioHub* para su funcionamiento (`ioHub Configuration Files`). Para identificar una ejecución de otra, cada grupo de archivos se identifica por la marca de tiempo del momento de ejecución en formato *Unix*.

3. Los resultados del experimento (`Experiment Data`) se almacenan directamente en la carpeta `[Experiment_Version][Experiment_Code]`. Este archivo se actualiza con cada ejecución y contiene los resultados de todas las sesiones del experimento.
4. De forma opcional, el *script* permite obtener capturas de pantalla de cada uno de los cuadros del experimento (`Frame Images`). Estas imágenes son almacenadas en la carpeta `frames`.

En la figura 3.4 se presenta la estructura de clases implementada en este trabajo de título. Las clases `Configuration`, `Experiment`, `Test`, `Frame` y `Component` representan el conjunto de métodos que permiten la manipulación de los datos almacenados en la base de datos, además de entregar funcionalidades que permiten la ejecución.

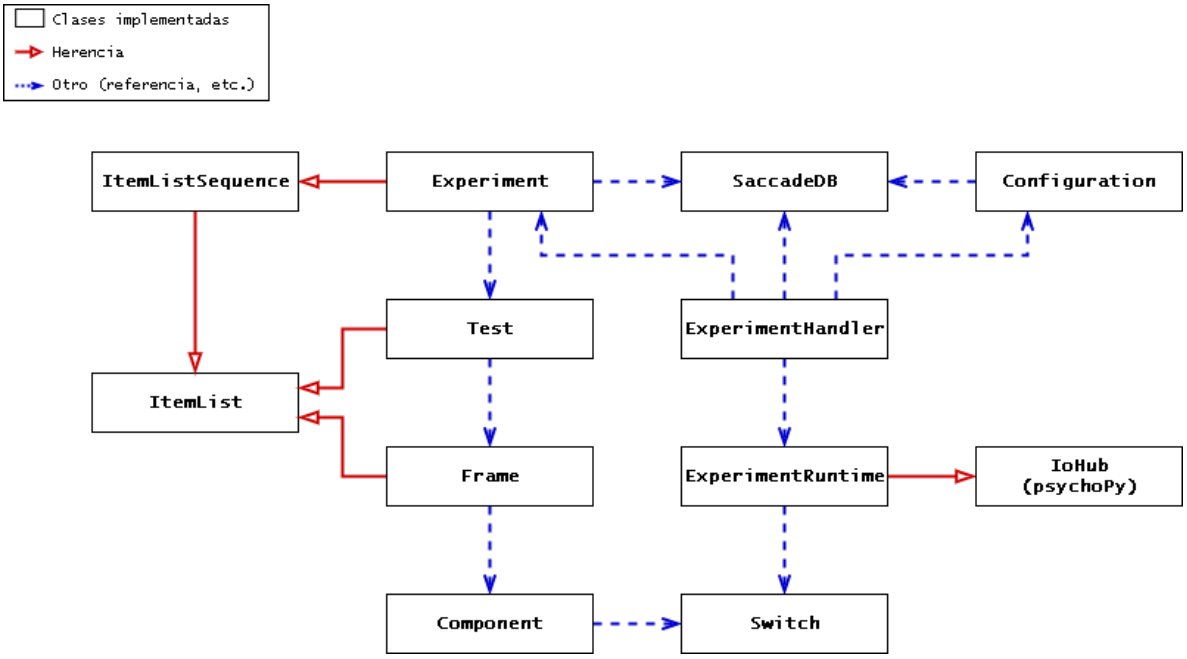


Figura 3.4: Diagrama de clases del sistema (no considera *GUI*).

Debido a que las clases `Experiment`, `Test` y `Frame` implican el manejo de una lista de objetos se crea una clase base llamada `ItemList` que permite integrar funcionalidades tales como agregar, quitar, mover o reemplazar elementos. Además, ya que un experimento implica la ejecución de una secuencia de tareas que pueden repetirse se implementa una variante de

`ItemList` que permite la manipulación de secuencias en base a los elementos disponibles en lista `ItemListSequence`.

Las clases `SaccadeDB` y `Switch` añaden funcionalidades transversales al resto de las clases: `SaccadeDB` permite la comunicación e interacción con el archivo de base de datos y `Switch` permite implementar estructuras *switch-case* para facilitar el uso de máquinas de estado. `ExperimentHandler` y `ExperimentRuntime` (que hereda sus funcionalidades de `ioHub` e inicializa los servicios asociados) son las encargadas de preparar los archivos de configuración y ejecutar el experimento respectivamente.

En las siguientes páginas se procede a describir con un mayor grado de detalle cada clase y módulo utilizado. Por simplicidad se obvian las funciones asociadas al get-set de los atributos de las clases (ya que corresponden a los datos almacenados en la base de datos).

1. **`utils`:** Módulo auxiliar que implementa métodos de uso general.

`format_text`		
Descripción	Convierte el dato de entrada a unicode. Si el dato ingresado no cumple con las condiciones de formato se levanta una excepción indicando el error asociado.	
Inputs	text:	Dato a ser convertido en `unicode`.
	lmin:	Largo mínimo (`int`). Por defecto: `None` (Sin condición).
	lmax:	Largo máximo (`int`). Por defecto: `None` (Sin condición)
	var_name:	Nombre de la variable para la cual se utilizará el dato formateado (`string`). Se agrega en el mensaje de excepción para hacer seguimiento del error. Por defecto: vacío (Sin nombre).
Return	`unicode`.	
`format_int`		
Descripción	Convierte un valor de entrada en entero. Si el valor ingresado no cumple con las condiciones de formato se levanta una excepción indicando el error asociado.	
Inputs	value:	Valor a ser convertido en int.
	vmin:	Valor mínimo permitido (`int`). Por defecto: `None` (Sin condición)
	vmax:	Valor máximo permitido (`int`). Por defecto: `None` (Sin condición)
	var_name:	Nombre de la variable para la cual se utilizará el dato formateado (`string`). Se agrega en el mensaje de excepción para hacer seguimiento del error. Por defecto: vacío (Sin nombre).
Return	`int`.	

format_float		
Descripción	Convierte un valor de entrada en flotante. Si el valor ingresado no cumple con las condiciones de formato se levanta una excepción indicando el error asociado.	
Inputs	value:	Valor a ser convertido en float.
	vmin:	Valor mínimo permitido (`float`). Por defecto: `None` (Sin condición)
	vmax:	Valor máximo permitido (`float`). Por defecto: `None` (Sin condición)
	var_name:	Nombre de la variable para la cual se utilizará el dato formateado (`string`). Se agrega en el mensaje de excepción para hacer seguimiento del error. Por defecto: vacío (Sin nombre).
Return	`float`.	
format_bool		
Descripción	Convierte el valor de entrada entregado en un booleano. Si el dato ingresado no puede ser convertido se levanta una excepción indicando el error asociado.	
Inputs	state:	Variable a convertir en bool.
	var_name:	Nombre de la variable para la cual se utilizará el dato formateado (`string`). Se agrega en el mensaje de excepción para hacer seguimiento del error. Por defecto: vacío (Sin nombre).
Return	`bool`.	
is_in_list		
Descripción	Indica si un elemento se encuentra o no en una lista.	
Inputs	item:	Elemento a encontrar.
	item_list:	Lista a revisar (`list`).
Return	`bool`.	`True` en caso de ser encontrado, `False` en caso contrario.
get_time		
Descripción	Permite convertir una fecha desde el formato de tiempo GTM al horario de Chile continental.	
Inputs	date:	Fecha (`string`) en formato "`Y-m-d H:M:S`".
Return	`unicode`.	
get_main_path		
Descripción	Retorna la ubicación en donde la aplicación se encuentra siendo ejecutada.	
Inputs	`void`.	
Return	`unicode`.	
get_module_path		
Descripción	Retorna la ubicación donde se encuentra almacenado el módulo de la aplicación (`saccadeapp`).	
Inputs	`void`.	
Return	`unicode`.	
format_path		
Descripción	Modifica la ubicación de entrada dependiendo del sistema operativo para asegurar compatibilidad.	
Inputs	path:	Ubicación de un archivo o directorio (`string`).
Return	`bool`.	

get_available_colors	
Descripción	Retorna una lista con los nombres de los colores disponibles en *psychoPy*.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
get_available_trackers	
Descripción	Retorna una lista con los nombres de los *eye trackers* disponibles.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
get_available_screens	
Descripción	Retorna una lista con la información (id y resolución) de los monitores disponibles.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
get_available_monitors	
Descripción	Retorna una lista con los nombres de los perfiles de configuración de monitor disponibles en *psychoPy* mediante el *Monitor Center*.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
get_available_units	
Descripción	Retorna una lista con los nombres de los tipos de unidad disponibles en *psychoPy* para el dibujo de estímulos en pantalla.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
get_available_shapes	
Descripción	Retorna una lista con los nombres de las figuras disponibles para la construcción de estímulos.
Inputs	`void`.
Return	`list`. Si no hay elementos: lista vacía.
open_documentation	
Descripción	Abre una copia del documento de memoria como documentación.
Inputs	`void`.
Return	`void`.
open_psychopy_monitor_center	
Descripción	Abre una instancia de la aplicación de configuración de monitores de *psychoPy*.
Inputs	`void`.
Return	`bool`. `True` si abre una nueva ventana, `False` si ya existe una en ejecución.

Cuadro 3.2: Métodos implementados en `utils`.

2. **`SaccadeDB`:** Clase utilizada para implementar la conexión con la base de datos SQLite.

`SaccadeDB`		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	path:	Directorio en donde se encuentra el archivo de base de datos (`string`). Por defecto: vacío (Se crea en el directorio de ejecución).
Return	`void`.	
`connect`		
Descripción	Inicia una conexión con el archivo de base de datos. Si el archivo de base de datos epecificado en el constructor no existe, crea uno nuevo.	
Inputs	`void`.	
Return	`void`.	
`close`		
Descripción	Finaliza, de ser posible, la conexión con la base de datos.	
Inputs	`void`.	
Return	`bool`.	`True` si la acción se realizó correctamente, `False` en caso contrario.
`push_query`		
Descripción	Permite realizar querys para modificar los contenidos de la base de datos (`insert, update, delete`).	
Inputs	query.	Instrucción SQL (`string`).
Return	`bool`.	`True` si la acción se realizó correctamente, `False` en caso contrario.
`pull_query`		
Descripción	Permite realizar querys para obtener datos desde de la base de datos (`select`).	
Inputs	query.	Instrucción SQL (`string`).
Return	`object`.	Array (`numpy`) en caso de existir datos, `None` en caso contrario.

Cuadro 3.3: Métodos implementados en la clase `SaccadeDB`.

3. **`Configuration`:** Clase que permite manipular las configuración del *hardware* a utilizar en la ejecución de un experimento y la ubicación de los archivos de salida de la misma. Para utilizar las funciones de carga/guardado es necesario asignar un objeto de base de datos (`SaccadeDB`).

`Configuration`		
Descripción	Constructor. Inicializa los atributos del objeto. Si se le entrega una objeto de base de datos y un nombre (asociado a un perfil de configuración) carga inmediatamente los datos.	
Inputs	db:	Instancia de clase (`SaccadeDB`). Por defecto: vacío. (Sin conexión).
	name:	Nombre de perfil de configuración (`string`). Por defecto: vacío (No buscar).
Return	`void`.	

get_list		
Descripción	Método estático. Entrega una lista con los nombres de los perfiles de configuración que se encuentran almacenados en la base de datos.	
Inputs	db:	Instancia de clase (`SaccadeDB`). Por defecto: vacío. (Sin conexión).
Return	`list`.	Si no hay elementos: lista vacía.
load		
Descripción	Permite cargar en el objeto las configuraciones almacenadas en la base de datos.	
Inputs	name:	Nombre de perfil de configuración (`string`).
Return	`bool`.	`True` si la acción se realizó correctamente, `False` en caso contrario.
save		
Descripción	Permite guardar el perfil de configuración en la base de datos.	
Inputs	`void`.	
Return	`bool`.	`True` si la acción se realizó correctamente, `False` en caso contrario.
copy		
Descripción	Permite realizar una copia del objeto siempre que el nombre asignado no exista en la base de datos.	
Inputs	name:	Nombre a ser asignado en la copya del perfil de configuración (`string`).
Return	`object`.	Devuelve una copia si se ingresa un nombre correctamente, `None` en caso contrario.
remove		
Descripción	Permite eliminar el perfil de configuración de la base de datos.	
Inputs	`void`.	
Return	`bool`.	`True` si la acción se realizó correctamente, `False` en caso contrario.
get_iohub		
Descripción	Retorna un diccionario con las configuraciones y formatos requeridos por *ioHub* para definir los dispositivos de *hardware* a ser utilizados en la ejecución de un experimento.	
Inputs	`void`.	
Return	`dict`.	Si no existe en la base de datos: `None`.
get_configuration		
Descripción	Retorna un diccionario que contiene los atributos del objeto.	
Inputs	`void`.	
Return	`dict`.	Si no existe en la base de datos: `None`.

Cuadro 3.4: Métodos implementados en la clase `Configuration`.

4. **`ItemList`:** Clase utilizada para manejar listas de objetos. Su implementación no considera el uso directo por parte de los usuarios del programa.

`ItemList`		
Descripción	Constructor. Inicializa los atributos de la clase. Permite definir el tipo de objetos a manipular.	
Inputs	item_class:	Tipo de objeto (`Test`, `Frame`, `Component`).
Return	`void`.	
`item_add`		
Descripción	Permite agregar un objeto a la lista.	
Inputs	item:	Objeto a añadir (`item_class`).
Return	`bool`.	`True` si el objeto fue añadido, `False` si el objeto es del tipo definido.
`item_copy`		
Descripción	Agrega una copia del objeto seleccionado en el último lugar de la lista.	
Inputs	item_id: name:	Posición en lista del objeto a copiar (`int`). Nombre del nuevo objeto (`string`).
Return	`bool`.	`True` si el índice existe y se realiza la copia, `False` en caso contrario.
`item_remove`		
Descripción	Elimina el objeto seleccionado.	
Inputs	item_id:	Posición en lista del objeto a eliminar (`int`).
Return	`bool`.	`True` si el índice existe y es eliminado, `False` en caso contrario.
`item_replace`		
Descripción	Remplaza el objeto seleccionado por uno nuevo.	
Inputs	item_id: new_item:	Posición en lista del objeto a reemplazar (`int`). Objeto reemplazante (`item_class`).
Return	`bool`.	`True` si el índice existe y el objeto es añadido, `False` en caso contrario.
`item_swap`		
Descripción	Permite hacer un enroque entre dos elementos de la lista.	
Inputs	item1_id: item2_id:	Posición en lista del primer objeto (`int`). Posición en lista del segundo objeto (`int`).
Return	`bool`.	`True` si los índices existen y se realiza el enroque, `False` en caso contrario.
`get_items`		
Descripción	Retorna la lista de objetos.	
Inputs	`void`.	
Return	`list`.	Si no hay elementos: lista vacía.
`get_item`		
Descripción	Retorna un elemento de la lista en base a su índice.	
Inputs	item_id:	Posición en lista del objeto a recuperar (`int`).
Return	`object`.	Si el índice no existe: `None`.

check_item_name		
Descripción	Retorna el índice del objeto que tenga el nombre ingresado.	
Inputs	item_name:	Nombre del objeto a buscar (`string`).
Return	`int`.	Si el nombre no existe: -1.
get_items_str		
Descripción	Retorna una lista con los nombres de los objetos almacenados.	
Inputs	`void`.	
Return	`list`.	Si no hay elementos: lista vacía
get_items_length		
Descripción	Devuelve el número de elementos de la lista.	
Inputs	`void`.	
Return	`int`.	

Cuadro 3.5: Métodos implementados en la clase `ItemList`.

5. **`ItemListSequence`:** Clase utilizada para extender las capacidades de una lista de objetos (hija de `ItemList`) permitiendo la configuración y manejo de secuencias. Para su funcionamiento deben re-implementarse dos métodos: El primero es `item_remove` ya que debe incluirse la eliminación de los elementos de la secuencia asociados al item. El segundo es `item_replace` ya que deben actualizarse las referencias de los elementos de la secuencia al nuevo ítem.

ItemListSequence		
Descripción	Constructor. Inicializa los atributos de la clase. Permite definir el tipo de objetos a manipular.	
Inputs	item_class:	Tipo de objeto (`Test`, `Frame`, `Component`).
Return	`void`.	
sequence_add		
Descripción	Permite agregar un elemento a la secuencia.	
Inputs	item_id:	Posición en lista del objeto a agregar (`int`).
	quantity:	Cantidad de veces que será ejecutado (`int`).
Return	`bool`.	`True` si el índice existe y la cantidad no es `None`, `False` en caso contrario.
sequence_edit		
Descripción	Permite editar un elemento de la secuencia.	
Inputs	index:	Posición del elemento de la secuencia a ser modificado (`int`).
	item_id:	Posición en lista del objeto reemplazante (`int`).
	quantity:	Cantidad de veces que será ejecutado (`int`).
Return	`bool`.	`True` si los índices existen y la cantidad no es `None`, `False` en caso contrario.
sequence_remove		
Descripción	Elimina el elemento seleccionado.	
Inputs	index:	Posición del elemento de la secuencia a ser eliminado (`int`).
Return	`bool`.	`True` si el índice existe y es eliminado, `False` en caso contrario.

sequence_swap		
Descripción	Permite hacer un enroque entre dos elementos de la secuencia.	
Inputs	index1: index2:	Posición del primer elemento de la secuencia (`int`). Posición del segundo elemento de la secuencia (`int`).
Return	`bool`.	`True` si los índices existen y se realiza el enroque, `False` en caso contrario.
get_sequence		
Descripción	Retorna la secuencia de ejecución.	
Inputs	`void`.	
Return	`list`.	Si no hay elementos: lista vacía.
get_sequence_item		
Descripción	Retorna un elemento de la secuencia en base a su índice.	
Inputs	index:	Posición del elemento de la secuencia (`int`).
Return	`list`.	Si el objeto no existe: `None`.
get_sequence_str		
Descripción	Retorna una lista con los nombres de los objetos en la secuencia y el numero de veces a ser ejecutados.	
Inputs	`void`.	
Return	`list`.	Si no hay elementos: lista vacía
get_sequence_length		
Descripción	Devuelve el número de elementos de la secuencia.	
Inputs	`void`.	
Return	`int`.	

Cuadro 3.6: Métodos implementados en la clase `ItemListSequence`.

6. **`Component`:** Clase utilizada para manejar la configuración de los componentes que conforman un cuadro.

Component		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	`void`.	
Return	`void`.	
get_list		
Descripción	Método estático. Retorna una lista con el nombre y tipo de figura de todos los componentes que forman parte de un cuadro específico.	
Inputs	db: exp: tes: fra:	Instancia de clase (`SaccadeDB`). Código del experimento al que pertenecen los componentes de interés (`string`). ID de la tarea a la cual pertenecen los componentes de interés (`int`). ID del cuadro al cual pertenecen los componentes de interés (`int`).
Return	`list`.	Si no hay elementos: lista vacía.

encode_image		
Descripción	Método privado. Si el componente corresponde a una imagen esta función retorna su contenido codificado en "base64".	
Inputs	void.	
Return	unicode.	
decode_image		
Descripción	Método privado. Permite decodificar una imagen almacenada en "base64".	
Inputs	enc_img:	Imagen codificada en "base64" (string).
Return	void.	
load		
Descripción	Permite cargar en el objeto las configuraciones almacenadas en la base de datos.	
Inputs	db:	Instancia de clase (SaccadeDB).
	exp:	Código del experimento al que pertenece el componente (string).
	tes:	ID de la tarea a la cual pertenece el componente (int).
	fra:	ID del cuadro al cual pertenece el componente (int).
	com:	ID asignado al componente (int).
Return	bool.	True si la acción se realizó correctamente, False en caso contrario.
save		
Descripción	Permite guardar la configuración del componente en la base de datos.	
Inputs	db:	Instancia de clase (SaccadeDB).
	exp:	Código del experimento al que pertenecerá el componente (string).
	tes:	ID de la tarea a la cual pertenecerá el componente (int).
	fra:	ID del cuadro al cual pertenecerá el componente (int).
	com:	ID asignado al componente (int).
Return	bool.	True si el proceso se completó satisfactoriamente, False en caso contrario.
copy		
Descripción	Retorna una copia profunda del componete.	
Inputs	void.	
Return	object.	
get_execution		
Descripción	Retorna el componente de foma tal que pueda ser presentado en el monitor (mediante uso de *psychoPy*).	
Inputs	win:	Instancia de clase Window, perteneciente al módulo psychopy.visual.
Return	object.	Objeto de la familia psychopy.visual. Actualmente se encuentran implementadas 5 figuras (cruz, cuadrado, círculo, gaussiana y flechas) además de la posibilidad de incluir imágenes. En caso de existir algún error, devuelve None.

get_configuration	
Descripción	Retorna un diccionario que contiene los atributos del objeto.
Inputs	`void`
Return	`dict`

Cuadro 3.7: Métodos implementados en la clase `Component`.

7. **`Frame`:** Clase utilizada para manejar la configuración de un cuadro. Ya que los cuadros contienen y muestran componentes, esta clase se define como hija de `ItemList` y se inicializa para objetos de tipo `Component`.

Frame		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	`void`.	
Return	`void`.	
get_list		
Descripción	Método estático. Retorna una lista con el nombre de todos los cuadros que forman parte de una tarea específica.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenecen los cuadros de interés (`string`).
	tes:	ID de la tarea a la cual pertenecen los cuadros de interés (`int`).
Return	`list`.	Si no hay elementos: lista vacía.
load		
Descripción	Permite cargar la configuración del cuadro desde la base de datos. Para cargar los componentes se hace uso del método privado `load_components`.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenece el cuadro (`string`).
	tes:	ID de la tarea a la cual pertenece el cuadro (`int`).
	fra:	ID del cuadro (`int`).
Return	`bool`.	`True` si el proceso se completó satisfactoriamente, `False` en caso contrario.
load_components		
Descripción	Método privado. Permite cargar desde la base de datos todos los componentes del cuadro.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenece el cuadro (`string`).
	tes:	ID de la tarea a la cual pertenece el cuadro (`int`).
	fra:	ID del cuadro (`int`).
Return	`void`.	

save	
Descripción	Permite guardar la configuración del cuadro en la base de datos. Para guardar los componentes se hace uso del método privado `save_components`.
Inputs	db: Instancia de clase (`SaccadeDB`). exp: Código del experimento al que pertenecerá el cuadro (`string`). tes: ID de la tarea a la cual pertenecerá el cuadro (`int`). fra: ID asignado al cuadro (`int`).
Return	`bool`. `True` si el proceso se completó satisfactoriamente, `False` en caso contrario.
save_components	
Descripción	Método privado. Permite guardar en la base de datos cada uno de los componentes del cuadro.
Inputs	db: Instancia de clase (`SaccadeDB`). exp: Código del experimento al que pertenece el cuadro (`string`). tes: ID de la tarea a la cual pertenece el cuadro (`int`). fra: ID del cuadro (`int`).
Return	`void`.
copy	
Descripción	Retorna una copia profunda del cuadro.
Inputs	`void`.
Return	`object`.
get_execution	
Descripción	Retorna un diccionario que contiene los elementos necesarios para la presentación del cuadro: Su nombre (para identificación), las características asociadas a su presentación (si es temporizado o involucra respuesta de teclado) y un objeto que permita cambiar el color del fondo según sea requerido. Además, este diccionario contiene una lista de los componentes que conforman el cuadro (que se obtiene al llamar el método `get_execution` para cada uno).
Inputs	win: Instancia de clase `Window`, perteneciente al módulo `psychopy.visual`.
Return	`dict`.
get_configuration	
Descripción	Retorna un diccionario que contiene los atributos del objeto y una lista con la configuración de sus componentes (esta lista se obtiene al llamar el método `get_configuration` para cada uno).
Inputs	`void`.
Return	`dict`.

Cuadro 3.8: Métodos implementados en la clase `Frame`.

8. **Test:** Clase utilizada para manejar la configuración de una tarea. Ya que las tareas se componen de cuadros, esta clase se define como hija de `ItemList` y se inicializa para objetos de tipo `Frame`.

Test		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	`void`.	
Return	`void`.	
get_list		
Descripción	Método estático. Retorna una lista con el nombre de todas las tareas que forman parte de un experimento específico.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al cual pertenecen las tareas de interés (`string`).
Return	`list`.	Si no hay elementos: lista vacía.
load		
Descripción	Permite cargar la configuración de una tarea desde la base de datos. Para cargar los cuadros se hace uso del método privado `load_frames`.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenece la tarea (`string`).
	tes:	ID de la tarea (`int`).
Return	`bool`.	`True` si el proceso se completó satisfactoriamente, `False` en caso contrario.
load_frames		
Descripción	Método privado. Permite cargar desde la base de datos todos los cuadros de la tarea.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenece la tarea (`string`).
	tes:	ID de la tarea (`int`).
Return	`void`.	
save		
Descripción	Permite guardar la configuración de la tarea en la base de datos. Para guardar los cuadros se hace uso del método privado `save_frames`.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenecerá la tarea (`string`).
	tes:	ID asignado a la tarea (`int`).
Return	`bool`.	`True` si el proceso se completó satisfactoriamente, `False` en caso contrario.
save_frames		
Descripción	Método privado. Permite guardar en la base de datos cada uno de los cuadros de la tarea.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
	exp:	Código del experimento al que pertenece la tarea (`string`).
	tes:	ID de la tarea (`int`).
Return	`void`.	

copy		
Descripción	Retorna una copia profunda del objeto.	
Inputs	`void`.	
Return	`object`.	
get_execution		
Descripción	Retorna un diccionario que contiene los elementos necesarios para la ejecución de la tarea: Su nombre (para identificación) y una lista con los cuadros que la conforman (que se obtiene al llamar el método `get_execution` para cada uno).	
Inputs	win:	Instancia de clase `Window`, perteneciente al módulo `psychopy.visual`.
Return	`dict`.	
get_configuration		
Descripción	Retorna un diccionario que contiene los atributos del objeto y una lista con la configuración de los cuadros que lo componen (esta lista se obtiene al llamar el método `get_configuration` para cada uno).	
Inputs	`void`.	
Return	`dict`.	

Cuadro 3.9: Métodos implementados en la clase `Test`.

9. **`Experiment`:** Clase utilizada para manejar la coniguración de un experimento. Ya que los experimentos se componen de tareas y estas pueden formar secuencias de ejecución esta clase se define como hija de `ItemLisSequence` y se inicializa para objetos de tipo `Test`. Para utilizar las funciones de carga/guardado es necesario asignar un objeto de base de datos (`SaccadeDB`).

Experiment		
Descripción	Constructor. Inicializa los atributos del objeto. Si se le entrega un objeto de base de datos y un código (asociado a un experimento) carga inmediatamente los datos.	
Inputs	db:	Instancia de clase (`SaccadeDB`). Por defecto: vacío. (Sin conexión).
	name:	Nombre de perfil de configuración (`string`). Por defecto: vacío (No buscar).
Return	`void`.	
get_list		
Descripción	Método estático. Retorna una lista con el código, nombre, versión y fechas de creación/modificación de todos los experimentos disponibles.	
Inputs	db:	Instancia de clase (`SaccadeDB`).
Return	`list`.	Si no hay elementos: lista vacía.
load		
Descripción	Permite cargar la configuración de un experimento desde la base de datos. Para cargar la configuración de las tareas se hace uso del método privado `load_tests`.	
Inputs	code:	Código del experimento (`string`).
Return	`bool`.	`True` si el proceso se completó satisfactoriamente, `False` en caso contrario.

load_tests		
Descripción	Método privado. Permite cargar desde la base de datos cada una de las tareas asociadas al experimento y su secuencia de ejecución.	
Inputs	`void`.	
Return	`void`.	
save		
Descripción	Permite guardar la configuración del experimento en la base de datos. Para guardar la configuración de las tareas se hace uso del método privado `save_tests`.	
Inputs	`void`.	
Return	`bool`.	`True` si el proceso se completó satisfactoriamente, `False` en caso contrario.
save_tests		
Descripción	Método privado. Permite guardar en la base de datos cada una de las tareas asociadas al experimento y su secuencia de ejecución.	
Inputs	`void`.	
Return	`void`.	
copy		
Descripción	Permite realizar una copia del objeto siempre que el código y versión asignados no existan en la base de datos.	
Inputs	code: version:	Código a ser asignado en la copia del experimento (`string`). Versión a ser asignada en la copia del experimento (`string`).
Return	`object`.	Devuelve una copia si el código y versión se ingresan correctamente, `None` en caso contrario.
get_iohub		
Descripción	Retorna un diccionario con las configuraciones y formatos requeridos por *ioHUb* para la configuración de un experimento a ser ejecutado.	
Inputs	`void`.	
Return	`dict`.	
get_execution		
Descripción	Retorna un diccionario que contiene todos los elementos para la ejecución de un experimento: Instrucciones a presentar por pantalla para el paciente, definición de tiempos de descanso, secuencia de ejecución de tareas y una lista con las tareas que lo conforman (esta lista se obtiene al llamar el método `get_execution` para cada una).	
Inputs	win:	Instancia de clase `Window`, perteneciente al módulo `psychopy.visual`.
Return	`dict`.	
get_configuration		
Descripción	Retorna un diccionario que contiene los atributos del experimento y una lista con la configuración de las tareas que lo conforman (esta lista se obtiene al llamar el método `get_configuration` para cada una).	
Inputs	`void`.	
Return	`dict`.	

Cuadro 3.10: Métodos implementados en la clase `Experiment`.

10. **`ExperimentHandler`:** Clase utilizada para generar los archivos y directorios necesarios para la ejecución además de dar partida a la misma.

`ExperimentHandler`		
Descripción	Constructor. Inicializa los atributos del objeto.	
Inputs	`void`.	
Return	`void`.	
`prepare`		
Descripción	Permite verificar que los datos del perfil de configuración y experimento se encuentren almacenados en la base de datos y generar los archivos de configuración necesarios para realizar el experimento.	
Inputs	db_path:	Ubicación del archivo de base de datos (`string`).
	conf_name:	Nombre del perfil de configuración (`string`).
	exp_code:	Código del experimento a ser ejecutado (`string`).
Return	`string`, `bool`.	Mensaje de estado, estado.
`execute`		
Descripción	Si `prepare` se ejecutó correctamente (archivos de configuración y directorios disponibles) este método ejecuta el experimento.	
Inputs	save_frame:	Permite escoger si se desea guardar los frames como imagen (`bool`).
Return	`string`, `bool`.	Mensaje de estado, estado.

Cuadro 3.11: Métodos implementados en la clase `ExperimentHandler`.

11. **`ExperimentRuntime`:** Esta clase es utilizada para ejecutar el experimento. Para ello, hereda de `ioHubExperimentRuntime`, perteneciente a *ioHub*, la capacidad de iniciar el servicio por el cual se manejan todos los dispositivos de *hardware*.

`ExperimentRuntime`		
Descripción	Constructor. Inicializa los atributos de la clase y permite entregar la configuración del experimento a la rutina.	
Inputs	parameters:	Parámetros del experimento generados por `ExperimentHandler` (`dict`).
Return	`void`.	
`run`		
Descripción	Este método tiene por función definir tanto la inicialización de los dispositivos como el *script* del experimento a ejecutar. Es importante destacar que este método NO se ejecuta directamente, para ello se utiliza la función específica `ExperimentRuntime.start()`.	
Inputs	*args:	No utilizado.
Return	`void`.	

Cuadro 3.12: Métodos implementados en la clase `ExperimentRuntime`.

La lógica general del *script* de ejecución puede ser revisada en el anexo A.

3.2.3. Tercera parte: Interfaz gráfica

Para facilitar el uso de las funcionalidades implementadas y hacer el proceso de creación, configuración y ejecución de experimentos más intuitivo se decide integrar todas las etapas en una interfaz gráfica de usuario o *GUI*. El diseño de esta herramienta se plantea de forma tal que el acceso a las características mencionadas se encuentre centralizada en una ventana principal y las configuraciones particulares de cada una sean accedidas desde ventanas auxiliares.

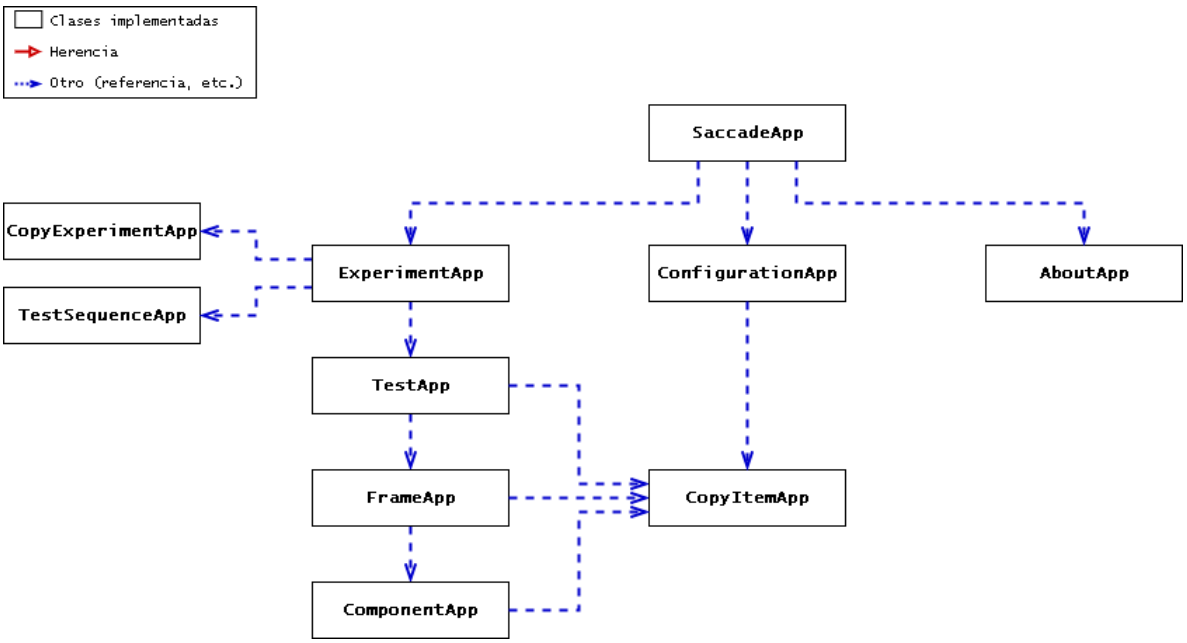
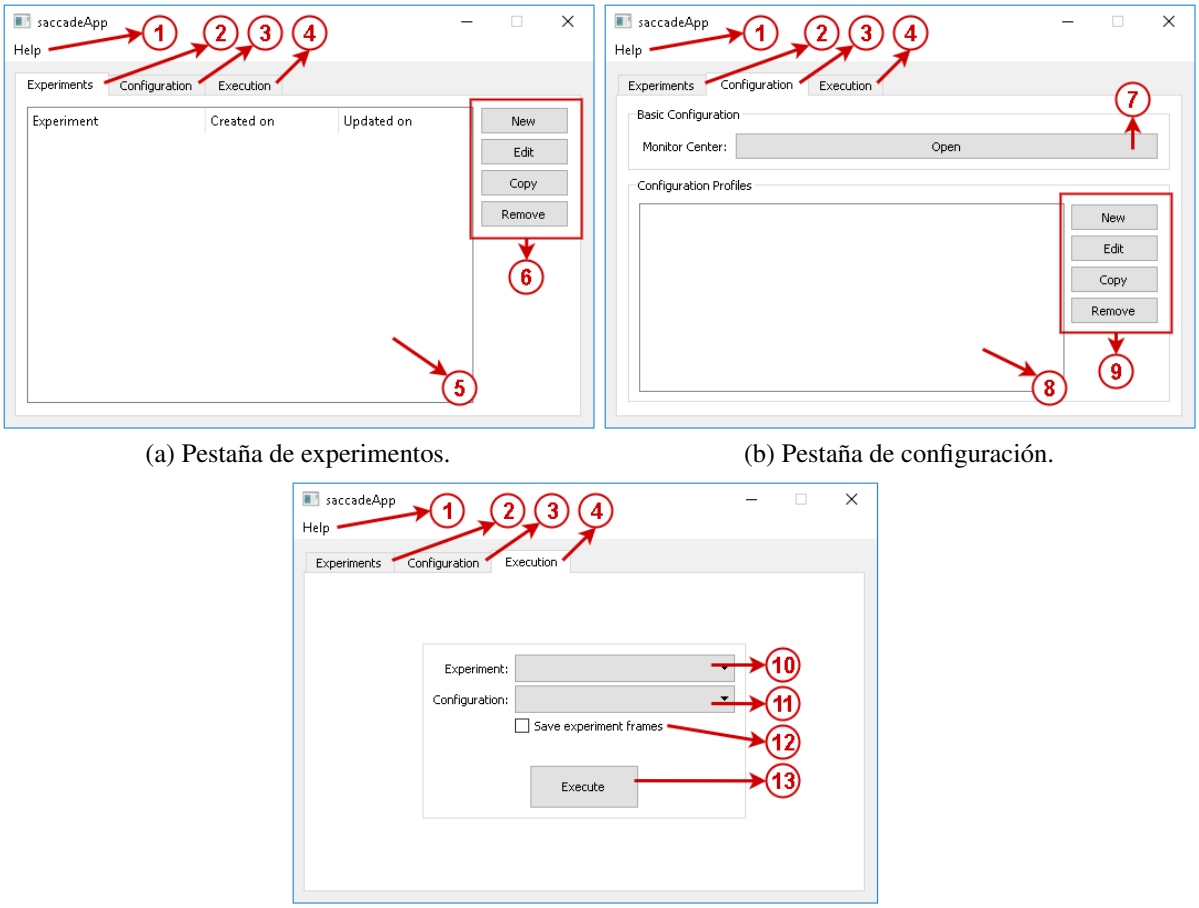


Figura 3.5: Diagrama de clases de la *GUI*.

En la figura 3.5 se aprecia un diagrama simplificado de las clases utilizadas para la construcción de las ventanas que conforman la *GUI*. En las siguientes páginas se presentará, para cada una, una descripción breve de su funcionalidad y se detallarán sus componentes.

1. **`SaccadeApp`:** Ventana principal de la aplicación. Permite revisar los experimentos y perfiles de configuración almacenados en la base de datos, acceder a la documentación y ejecutar experimentos.



(a) Pestaña de experimentos.

(b) Pestaña de configuración.

(c) Pestaña de ejecución.

Figura 3.6: Ventana principal.

En la figura 3.6:

(1) Menú de ayuda. Contiene accesos a la documentación de la aplicación (este trabajo de título) y a información de contacto del autor.
(2) Pestaña que contiene los experimentos disponibles.
(3) Pestaña que contiene los perfiles de configuración.
(4) Pestaña donde se encuentra el dialogo de ejecución de experimentos.
(5) Vista de tipo árbol que muestra los experimentos almacenados en la base de datos ordenados por nombre y versión. Se indican también la fecha de creación y la fecha de la última modificación del mismo.
(6) Permite la manipulación de los experimentos almacenados.
(7) Acceso al centro de monitores de *psychoPy*.

(8) Lista de los perfiles de configuración almacenados en la base de datos ordenados por nombre.
(9) Permite la manipulación de los perfiles de configuración almacenados.
(10) Lista de experimentos disponibles para ejecución.
(11) Lista con los perfiles de configuración disponibles.
(12) Permite escoger si se desea almacenar durante la ejecución capturas de pantalla de los cuadros desplegados en cada experimento.
(13) Permite comenzar la ejecución de un experimento.

2. **`ExperimentApp`:** Permite revisar y/o configurar un experimento, las tareas que lo componen y la secuencia de ejecución de las mismas. Se puede acceder a ella desde la ventana principal mediante los botones `New` y `Edit` ubicados en la pestaña `Experiments`.

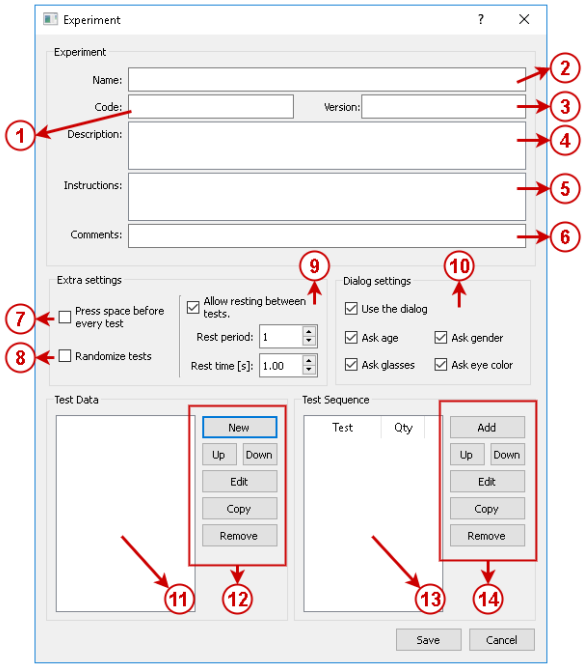


Figura 3.7: Ventana para configuración de experimentos.

En la figura 3.7:

(1) Campo obligatorio, usualmente críptico, que identifica de forma única al experimento. Debe tener entre 3 y 10 caracteres.
(2) Campo obligatorio que identifica de forma breve, más no críptica, un experimento (similar al nombre que se le daría en el título de un *paper*). Debe tener entre 3 y 50 caracteres.
(3) Campo obligatorio que identifica distintas implementaciones o versiones de un tipo de experimento. Debe tener entre 3 y 10 caracteres.

(4) Campo optativo que permite describir de forma más clara la naturaleza y objetivos del experimento.
(5) Campo optativo, a ser presentado al paciente al comienzo de la sesión de experimentación, que describe de forma general el como proceder en las tareas a realizar.
(6) Campo optativo que permite incluir un recordatorio breve para quien realiza la sesión de experimentación. El mensaje puede contener hasta 50 caracteres.
(7) Permite definir si el paciente debe presionar la tecla espacio antes de la realización de cada tarea.
(8) Permite presentar las tareas de la secuencia de ejecución de forma aleatoria.
(9) Permite configurar tanto la frecuencia como duración de descansos durante la sesión de experimentación.
(10) Permite incluir algunas preguntas tipo para adquirir más información del paciente.
(11) Lista con las tareas configuradas para el experimento.
(12) Permite la manipulación de las tareas configuradas.
(13) Tabla que muestra la secuencia de ejecución del experimento indicando el nombre de la tarea a presentar y el número de repeticiones.
(14) Permite la manipulación de los elementos de la secuencia.

3. **`CopyExperimentApp`:** Permite realizar una copia de un experimento existente, para lo cual se requiere obtener un nuevo código y versión. Se puede acceder a ella desde la ventana principal mediante el botón `copy` ubicado en la pestaña `Experiments`.

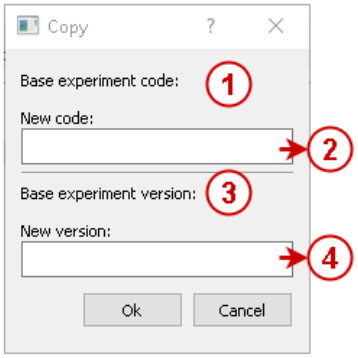


Figura 3.8: Ventana de copia para experimentos.

En la figura 3.8:

(1) Muestra el código utilizado en el experimento original.
(2) Campo obligatorio, usualmente críptico, que identifica de forma única al experimento. Debe tener entre 3 y 10 caracteres.
(3) Muestra la versión utilizada en el experimento original.
(4) Campo obligatorio que identifica distintas implementaciones o versiones de un tipo de experimento. Debe tener entre 3 y 10 caracteres.

4. **`TestSequenceApp`:** Permite la creación de un nuevo elemento para la secuencia de ejecución del experimento. Se puede acceder a ella desde `ExperimentApp` mediante el botón `Add` o `Edit` de la botonera de secuencia.

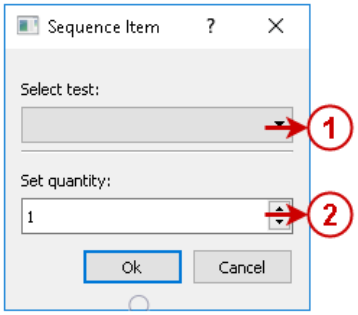


Figura 3.9: Ventana de configuración para secuencia de ejecución del experimento.

En la figura 3.9:

(1) Lista con las tareas disponibles para el experimento.
(2) Campo para ingresar el número de veces que la tarea debe ser ejecutada.

5. **`TestApp`:** Permite revisar y/o configurar una tarea específica y los cuadros que la componen. Se puede acceder a ella desde `ExperimentApp` utilizando los botones `New` y `Edit` de la lista de tareas.

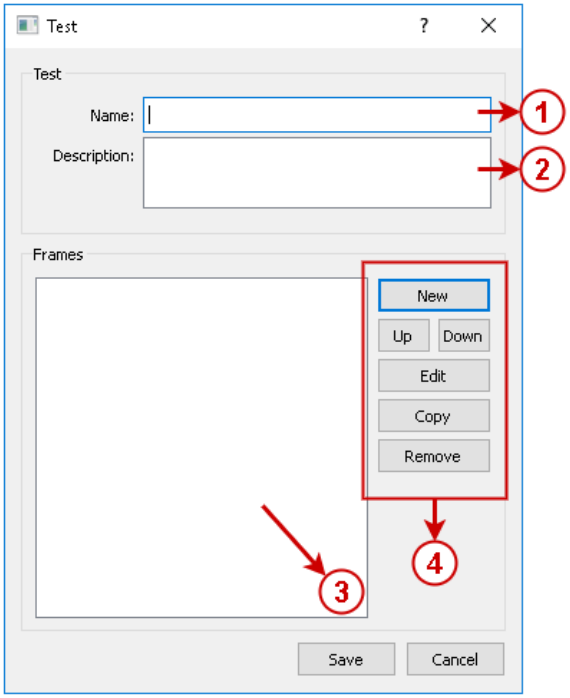


Figura 3.10: Ventana para configuración de tareas.

En la figura 3.10:

(1) Campo obligatorio, identifica de forma única a la tarea dentro de un experimento. Debe tener entre 3 y 50 caracteres.
(2) Campo optativo que permite describir de forma mas clara las características de la tarea.
(3) Lista con los cuadros que componen la tarea.
(4) Permite manipular los cuadros configurados.

6. **`FrameApp`:** Permite revisar y/o configurar un cuadro específico, además de los componentes que lo conforman. Se puede acceder a esta ventana desde `TestApp` utilizando los botones `New` y `Edit` de la lista de cuadros.

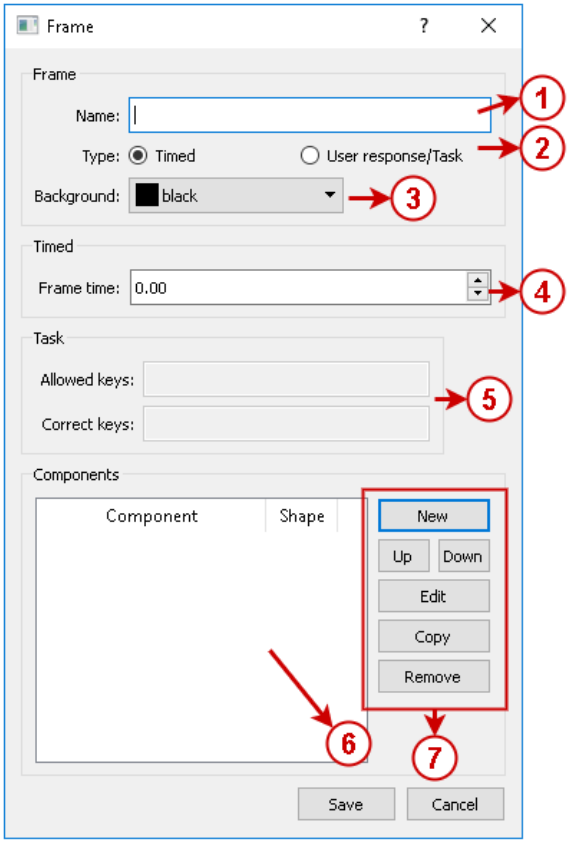


Figura 3.11: Ventana para configuración de cuadros.

En la figura 3.11:

(1) Campo obligatorio, identifica de forma única al cuadro dentro de una tarea. Debe tener entre 3 y 50 caracteres.
(2) Selecciona el tiempo de comportamiento que tendrá el cuadro durante la ejecución.
(3) Permite escoger el color de fondo del cuadro.

(4) Si se selecciona un comportamiento temporizado este campo permite definir por cuanto tiempo, en segundos, se mantendrá el cuadro en pantalla.
(5) En caso de escoger un comportamiento que requiere respuesta de usuario (mediante teclado) estos campos permiten definir tanto las teclas permitidas como, en caso de existir, las que se asocian a respuestas correctas.
(6) Lista que indica tanto los nombres como tipos de forma de los componentes que conforman el cuadro.
(7) Permite manipular los componentes configurados.

7. **`ComponentApp`:** Permite revisar/configurar las características de un componente visual específico. Se puede acceder a esta ventana desde `FrameApp` utilizando los botones `New` o `Edit` de la lista de componentes.

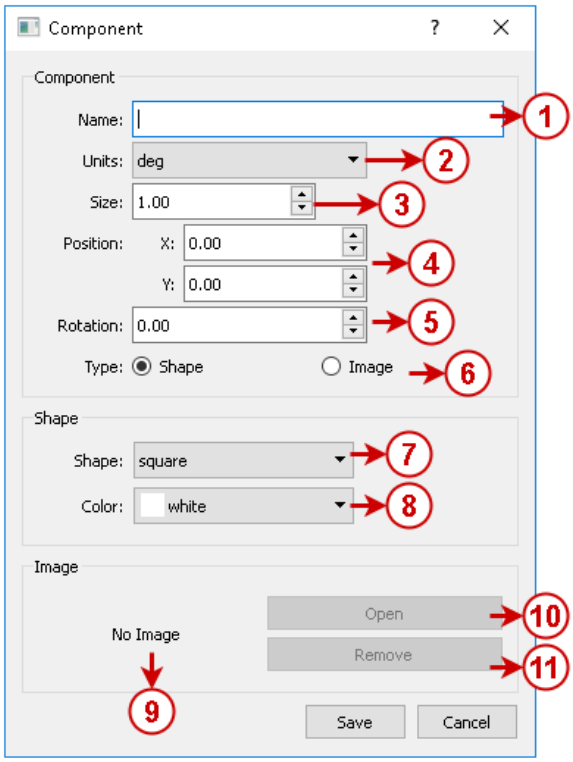


Figura 3.12: Ventana de configuración de componente.

En la figura 3.12:

(1) Campo obligatorio, identifica de forma única al componente dentro de un cuadro. Debe tener entre 3 y 50 caracteres.
(2) Permite definir el tipo de unidades a utilizar para dibujar el componente.
(3) Tamaño de la figura.
(4) Posición de la figura en el cuadro en base a las unidades de medida seleccionadas.

(5) Permite rotar al componente respecto de su centro.
(6) Selecciona si se trabajará con una figura predeterminada o una imagen.
(7) Permite escoger la figura a utilizar.
(8) Color del relleno y bordes de la figura.
(9) En caso de existir una imagen cargada en el componente esta se muestra en este espacio. Para hacer *zoom* solo es necesario pasar el cursor por encima.
(10) Permite buscar una imagen en el equipo y cargarla en el componente.
(11) Permite eliminar la imagen.

8. **`ConfigurationApp`:** Permite revisar/configurar las características de un perfil de configuración. Se puede acceder a ella desde la ventana principal mediante los botones `New` y `Edit` ubicados en la pestaña `Configuration`.

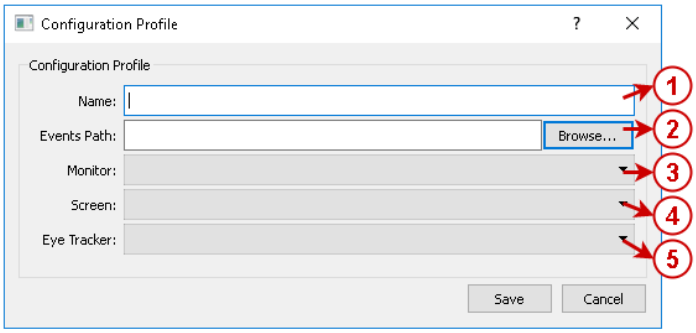


Figura 3.13: Ventana para copia de elementos.

En la figura 3.13:

(1) Campo obligatorio, identifica de forma única al perfil de configuración. Debe tener entre 3 y 50 caracteres.
(2) Permite definir un directorio para almacenar los resultados obtenidos de la ejecución del experimento.
(3) Lista que muestra los perfiles de configuración de monitor disponibles[3].
(4) Lista que indica las pantallas conectadas y sus dimensiones.
(5) Lista que indica las configuraciones disponibles de *eye tracker*.

9. **`CopyItemApp`:** Esta ventana se usa en los procesos de copia de tareas, cuadros, componentes y perfiles de configuración para determinar el nombre del nuevo objeto, que debe cumplir con no estar en la lista contenedora. Esta ventana puede ser accedida al utilizar el botón `Copy` desde `FrameApp`, `TestApp`, `ExperimentApp` y la pestaña `Configuration` en `SaccadeApp`.

[3]En el *Monitor Center* de *psychoPy*. Es posible acceder a esta herramienta desde `SaccadeApp`, pestaña `Configuration`.

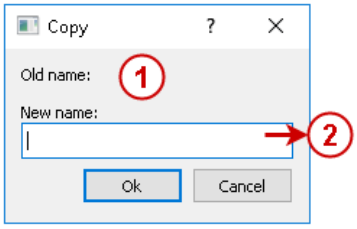


Figura 3.14: Ventana utilizada en el proceso de copia.

En la figura 3.14:

(1) Muestra el nombre del objeto original.
(2) Campo obligatorio que identifica de forma única al objeto a copiar. Debe tener entre 3 y 50 caracteres.

10. **`AboutApp`:** Ventana de información. Indica la versión de la aplicación y datos de contacto del autor.

Figura 3.15: Ventana de información.

Capítulo 4

Resultados

4.1. Modo de uso

Debido a las características de este trabajo de título se opta por presentar los resultados obtenidos a modo de manual de uso. En este sentido, y dado que la aplicación puede ser utilizada tanto en forma de *API* como de *GUI*, se decide presentar desde estas dos perspectivas el proceso de configuración y ejecución de un experimento.

El experimento a configurar estará conformado por dos tareas que se ejecutarán una sola vez, de forma secuencial y luego de presionar la tecla espacio: La primera tarea a configurar consistirá en un ejercicio de movimiento prosacádico bajo el paradigma de *overlap*, lo que implica que el estímulo central se encontrará presente en todos los cuadros. De esta forma, se propone un cuadro inicial que contenga solo dicho estímulo identificado por una cruz blanca, con duración de $1{,}7[s]$. Luego, se presenta un cuadro que contenga además el punto objetivo identificado por un cuadro rojo, $16°$ a la derecha del estímulo inicial, durante $1{,}5[s]$. La segunda tarea consiste en la presentación de una imagen centrada para analizar patrones de búsqueda. En este caso el cuadro no debe ser temporizado: La tarea termina luego de que el usuario presione la tecla espacio.

La presentación de los estímulos se realizará utilizando el monitor secundario[1] que se encuentra ubicado a $53[cm]$ del paciente[2]. Además, se hará uso de un *eye tracker* "Eyetribe" y los resultados obtenidos serán almacenados en el directorio `"C:/SaccadeApp/events/"`.

[1] Resolución de $1680x1050[px]$ y una pantalla con ancho efectivo de $47{,}5[cm]$.

[2] La configuración del monitor debe ser realizada mediante el *Monitor center* de *psychoPy*. Al perfil correspondiente se le asigna el nombre "my_monitor" por simplicidad.

4.1.1. Usando la *API*

Configurar

En este proceso se utilizarán los siguientes módulos:

```
from saccadeapp.api import SaccadeDB
from saccadeapp.api import Configuration
from saccadeapp.api import Experiment, Test, Frame, Component
```

El primer paso es contar con una base de datos disponible. Para crear una en el directorio donde se ejecutan los comandos se utiliza la siguiente instrucción:

```
database = SaccadeDB()
```

Luego, es posible crear el experimento propuesto:

```
# Experiment:
experiment = Experiment(db=database)
experiment.set_code(code=u"exp_0001")
experiment.set_info(name=u"Example Experiment", version=u"coded_1.0")
experiment.set_descripton(text=u"This experiment was created with code!")
experiment.set_space_start(status=True)
experiment.set_dialog(status=False)
experiment.set_random(status=False)
experiment.set_rest_conf(status=False)
```

Las tareas que lo conforman:

```
### Test_1: Prosaccade ######
test1 = Test()
test1.set_name(u"Test 1")
# Test_1 - Frame_1:
t1_fra1 = Frame()
t1_fra1.set_name(u"FP frame")
t1_fra1.set_color(u"black")
t1_fra1.set_as_task(False)
t1_fra1.set_time(1.7)
test1.add_item(t1_fra1)
# Test_1 - Frame_1 - FP
t1_fp = Component()
t1_fp.set_name(u"Fix Point")
t1_fp.set_shape(u"cross")
t1_fp.set_color(u"white")
t1_fp.set_position(0.0, 0.0)
t1_fra1.add_item(t1_fp)
# Test_1 - Frame_2:
t1_fra2 = t1_fra1.copy()
t1_fra2.set_name(u"TP frame")
t1_fra2.set_time(1.5)
test1.add_item(t1_fra2)
# Test_1 - Frame_2 - TP
t1_tp = Component()
t1_tp.set_name(u"Target Point")
t1_tp.set_shape(u"square")
t1_tp.set_color(u"red")
t1_tp.set_position(16.0, 0.0)
t1_fra2.add_item(t1_tp)
#Add test1 to experiment
experiment.add_item(test1)
#Add test1 to sequence
experiment.sequence_add(item_id=0, quantity=1)
```

```
### Test_2: Couple Image ####
test2 = Test()
test2.set_name(u"Test 2")
# Test_2 - Frame_1:
t2_fra1 = Frame()
t2_fra1.set_name(u"Couple")
t2_fra1.set_color(u"black")
t2_fra1.set_as_task(True)
t2_fra1.set_keys_allowed(u"space")
t2_fra1.set_keys_correct(u"space")
test2.add_item(t2_fra1)
# Test_2 - Frame_1 - Image:
t2_img = Component()
t2_img.set_name(u"Couple Image")
t2_img.set_image(u"couple.png")
t2_fra1.add_item(t2_img)
#Add test2 to experiment
experiment.add_item(test2)
#Add test2 to sequence
experiment.sequence_add(item_id=1, quantity=1)
```

Y el perfil de configuración requerido:

```
config_profile = Configuration(db=database)
config_profile.set_name(name=u"Test Profile (code)")
config_profile.set_events_path(path=u"C:/SaccadeApp/events/")
config_profile.set_tracker(tracker=u"eyetribe")
config_profile.set_monitor(monitor=u"my_Monitor")
config_profile.set_screen(screen=1)
```

Finalmente se almacena en la base de datos tanto el experimento como el perfil de configuración:

```
experiment.save()
config_profile.save()
```

Ejecutar

Para ejecutar el experimento que acaba de ser configurado y almacenado en la base de datos según las condiciones propuestas basta utilizar los siguientes comandos:

```
from saccadeapp.app import ExperimentHandler

handler = ExperimentHandler()
handler.prepare(exp_code=u"exp_0001", conf_name=u"Test Profile (code)")
handler.execute(frame_save=True)
```

4.1.2. Usando la *GUI*

Configurar

El primer paso consiste en añadir un perfil de configuración. Para esto es necesario ir a la pestaña `Configuration` en la ventana principal.

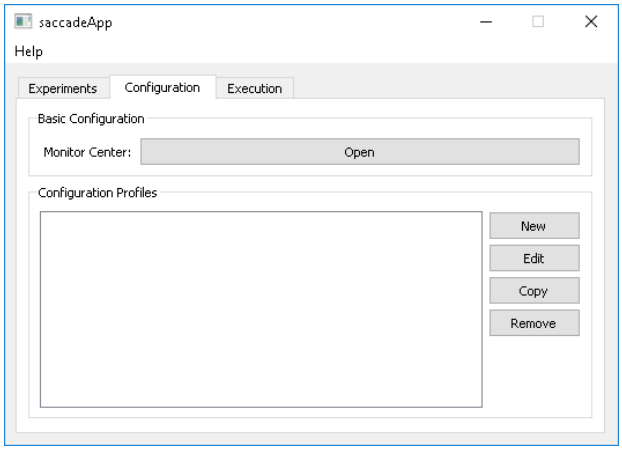


Figura 4.1: Pestaña de configuración.

Se crea un nuevo perfil donde se ingresan las configuraciones propuestas para el ejemplo, como se muestra a continuación:

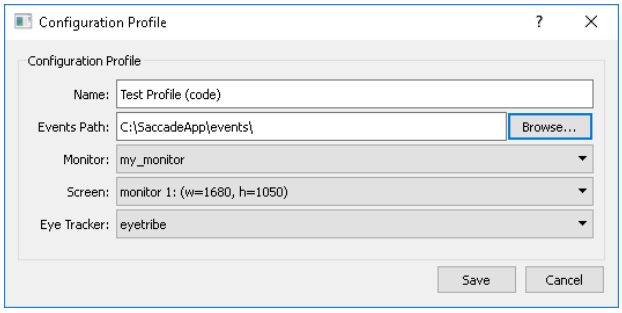


Figura 4.2: Perfil de configuración propuesto.

Luego, se finaliza la operación presionando el botón `Save`.

El segundo paso consiste en añadir un nuevo experimento. Para esto es necesario ir a la pestaña `Experiment` en la ventana principal.

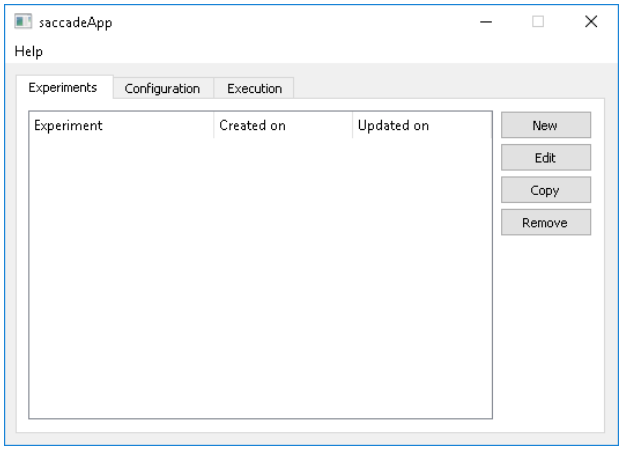


Figura 4.3: Pestaña de experimentos.

Se crea un nuevo experimento donde se ingresan las configuraciones propuestas y se añade la primera tarea.

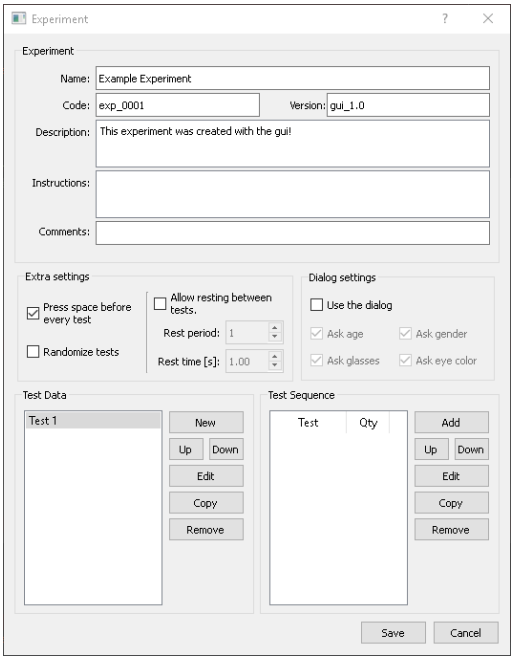


Figura 4.4: Experimento propuesto con la primera tarea configurada.

Dicha tarea se compone de dos cuadros:

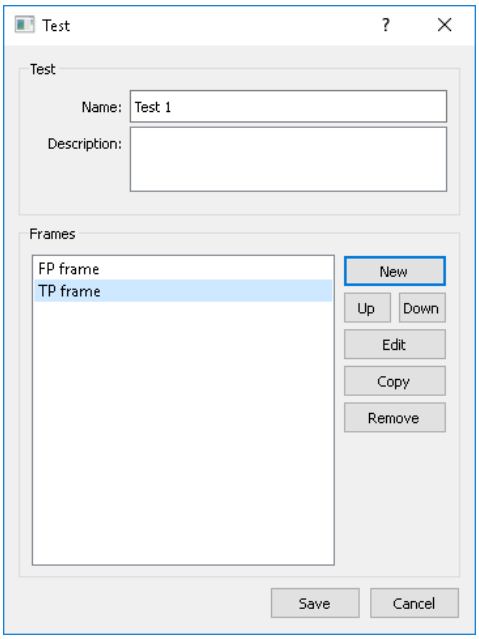
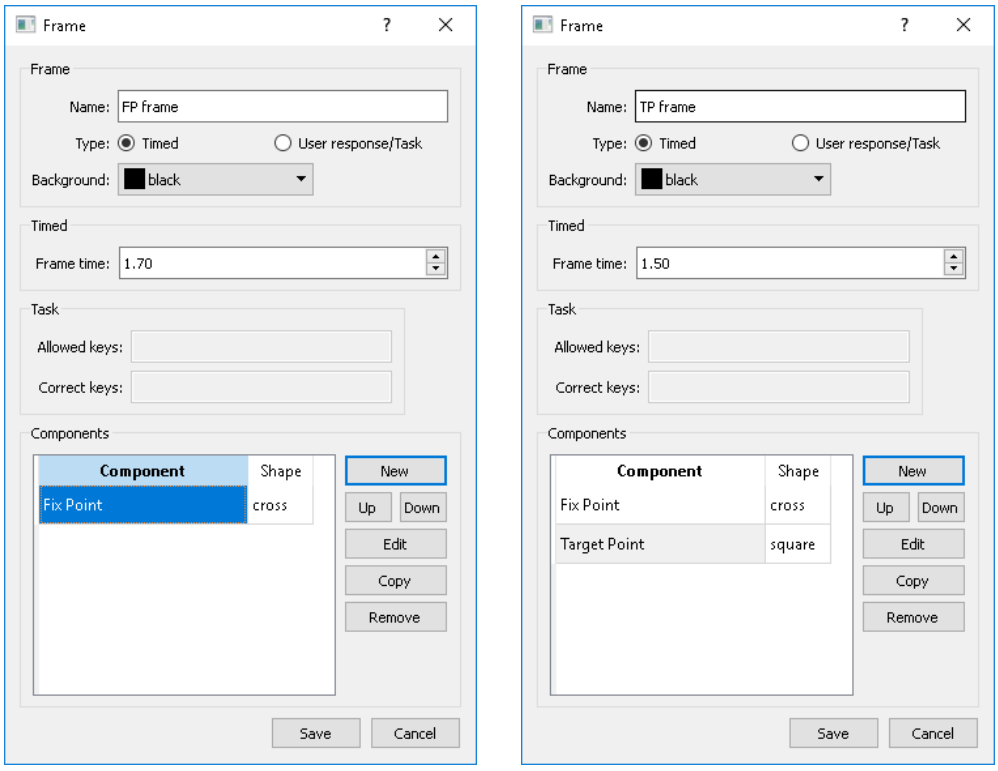


Figura 4.5: Ventana de configuración de la primera tarea.

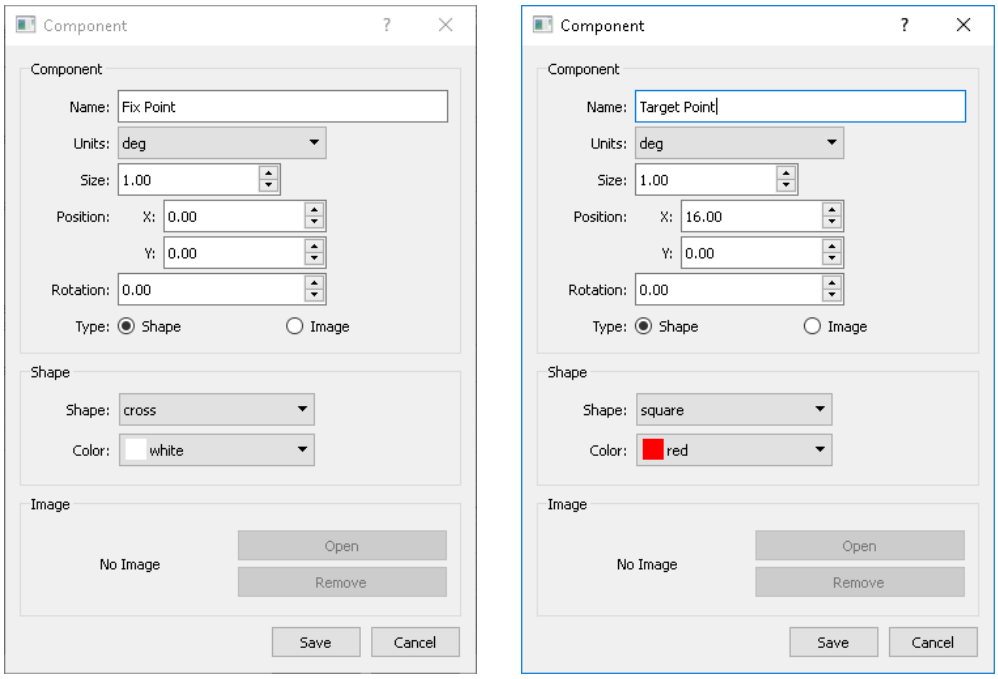
Que se encuentran configurados de la forma:



(a) Primer cuadro. (b) Segundo cuadro.

Figura 4.6: Configuración de los cuadros pertenecientes a la primera tarea.

Y donde los componentes que las conforman corresponden a:



(a) Punto de fijación. (b) Punto objetivo.

Figura 4.7: Componentes utilizados en la configuración de la primera tarea.

Para agregar la tarea a la secuencia de ejecución se utiliza el botón `Add` de la lista de secuencias, lo que despliega la siguiente ventana:

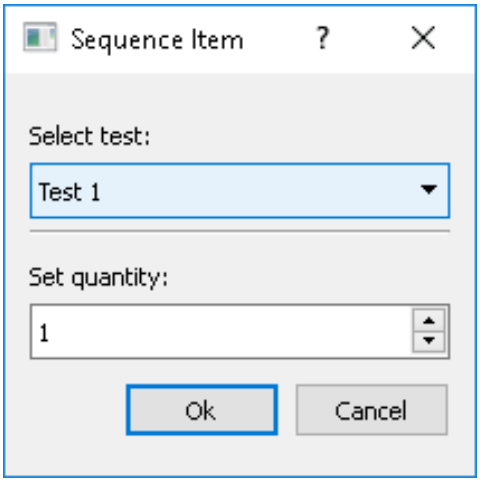


Figura 4.8: Agregando la primera tarea a la secuencia de ejecución.

Luego de añadir ese elemento se obtiene el siguiente resultado:

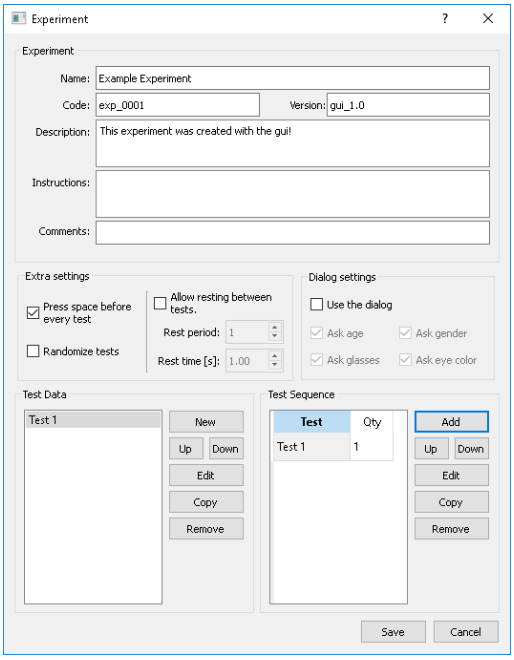


Figura 4.9: Experimento propuesto con la primera tarea configurada y en la secuencia de ejecución.

De forma posterior se añade la segunda tarea, que solo se compone de un cuadro.

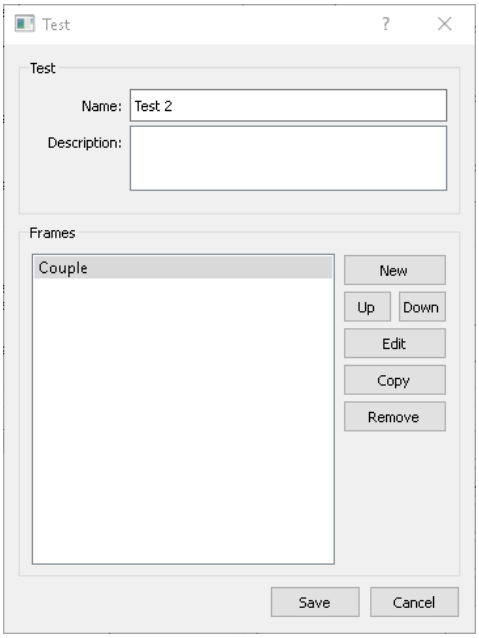
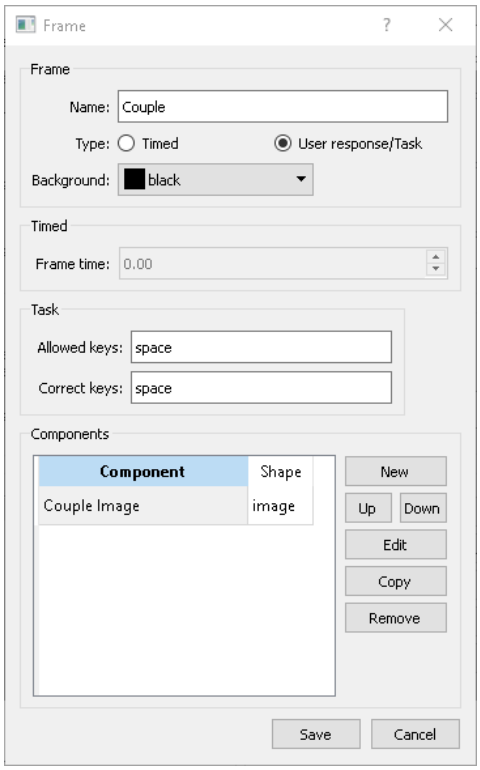
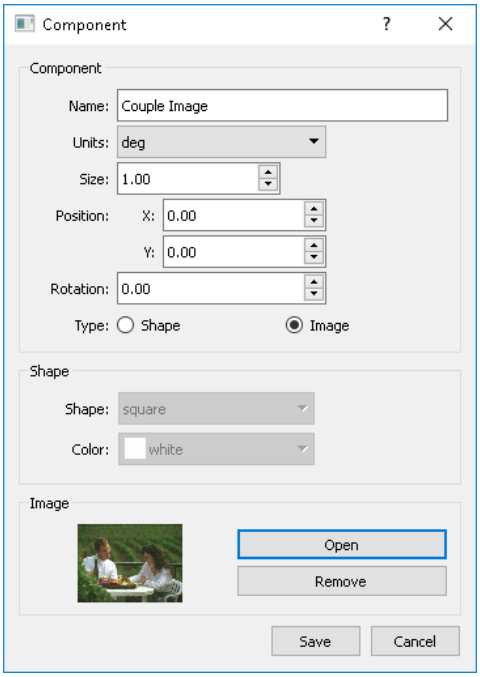


Figura 4.10: Ventana de configuración de la segunda tarea.

Que presenta las siguientes configuraciones:



(a) Cuadro que contiene la imagen.



(b) Imagen de la pareja.

Figura 4.11: Configuraciones de la segunda tarea.

Al añadir esta tarea a la secuencia de ejecución, el experimento queda de la forma:

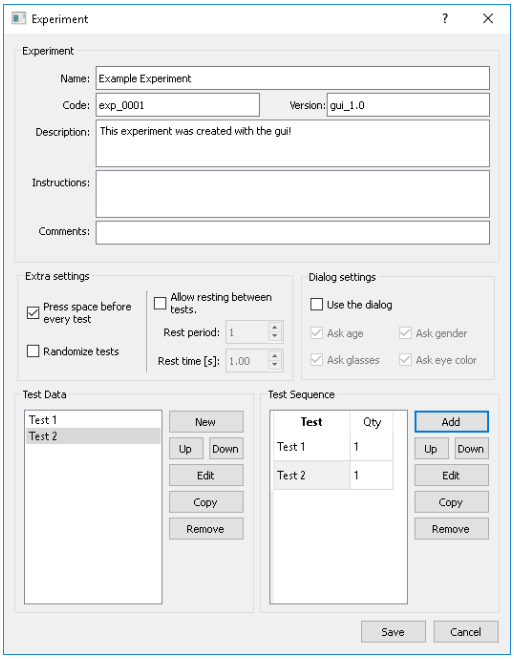
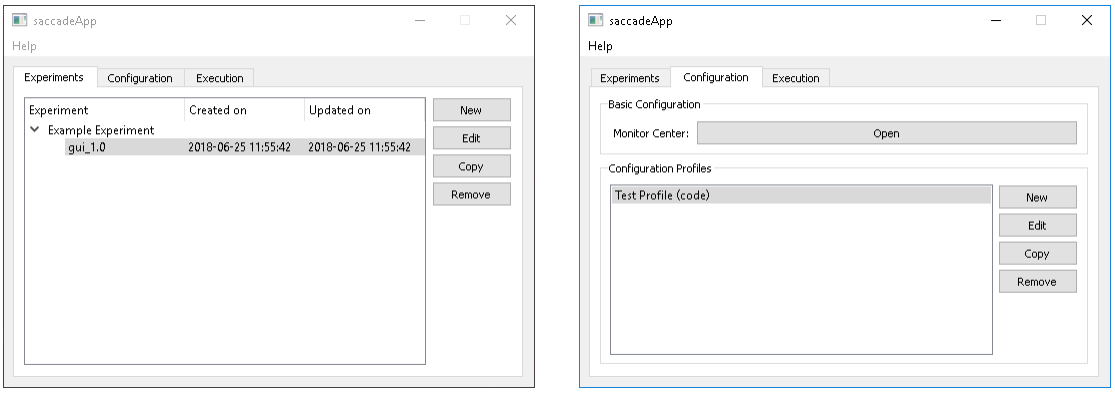


Figura 4.12: Experimento propuesto con configuración completa.

Luego, se finaliza la operación presionando el botón `Save`.

Con estas configuraciones ya es posible ejecutar un experimento. Las pestañas de configuración, luego de añadir estos elementos quedan de la siguiente forma:



(a) Pestaña de experimentos. (b) Pestaña de configuración.

Figura 4.13: Resultado obtenido.

Ejecutar

Para ejecutar el experimento es necesario ir a la pestaña `Execution` en la ventana principal y seleccionar los elementos de interés de las listas desplegables:

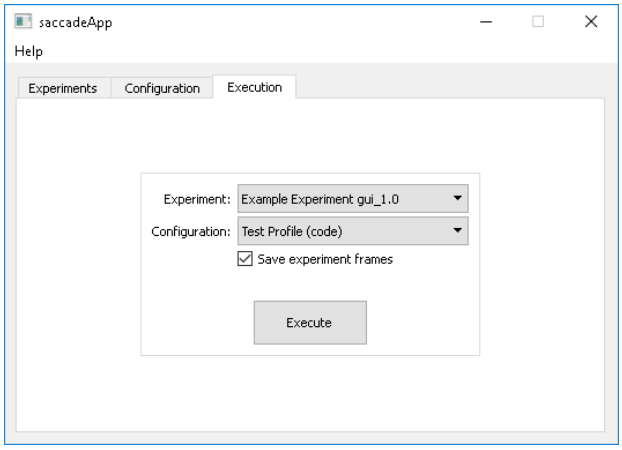


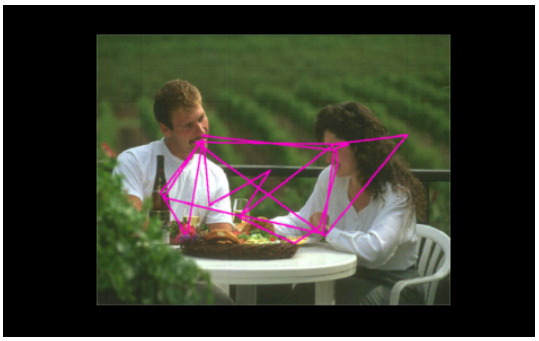
Figura 4.14: Aplicación lista para ejecutar *script*.

Finalmente se debe presionar el botón `Execute`.

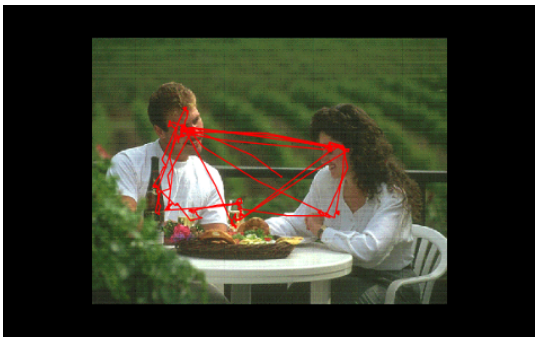
4.2. Comprobación de funcionamiento.

Con el fin de comprobar el funcionamiento de la aplicación y asegurar el correcto registro de los datos se creó un experimento cuya única tarea consistía en realizar un ejercicio de inspección visual sobre la imagen presentada en la figura 4.15 (a) durante $15[s]$.

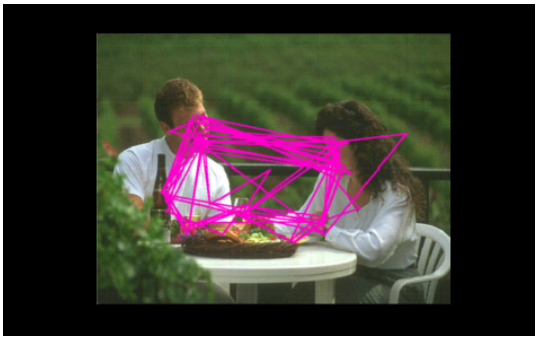
(a) Imagen original.



(b) Registro esperado a $2[s]$.



(c) Registro obtenido a $2[s]$.



(d) Registro esperado a $10[s]$.

(e) Registro obtenido a $10[s]$.

Figura 4.15: Datos registrados.

Los datos registrados desde el *eye tracker* en este experimento (4.15 (c) y (e)) muestran ser similares a los esperados (4.15 (b) y (d)), e indican que el movimiento ocular se realiza principalmente entre los rostros de la pareja y las proyecciones de hacia donde se encuentran dirigidas sus miradas.

Al consultar los registros asociados a los puntos anómalos presentes en 4.15 (e) se determina que se encuentran relacionados a los momentos en que el usuario cierra los ojos y por tanto no es posible determinar la posición ocular.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo de título fue posible construir un sistema de estimulación visual y registro ocular para la realización de experimentos que evalúan tareas sicomotoras que cumple con los objetivos propuestos al ser capaz de:

1. Permitir la configuración de experimentos y su posterior presentación.
2. Permitir sincronizar los eventos de estimulación y registro de movimiento ocular, almacenando los datos obtenidos en un archivo.

No obstante lo anterior y a pesar de que el sistema es capaz de montar y ejecutar los experimentos propuestos (para movimiento pro/anti-sacádico y ejercicios de memoria), presenta limitaciones para la realización de tareas mas complejas o añadir variaciones aleatorias de forma dinámica en el experimento, como cambiar los tiempos en que se presenta un cuadro durante la ejecución, lo que limita cierto tipo de análisis.

5.2. Trabajo futuro

Como trabajo para futuros desarrollos se proponen las siguientes tareas:

1. Añadir la opción de que un cuadro temporizado tenga tiempo variable dentro de un rango definido.
2. Añadir dentro de una tarea conjuntos de cuadros de selección aleatoria con el fin de optimizar el proceso de configuración. Eso quiere decir que tareas que tengan diferencias mínimas pueden ser almacenadas y posteriormente revisadas de forma más eficiente. Por ejemplo: Tarea de movimiento pro-sacádico en distintas direcciones o con distintas distancias al estímulo central.
3. Implementar una mayor variedad de opciones para la configuración de componentes de cuadro, ya sea añadiendo nuevos elementos gráficos como la posibilidad de incluir estímulos sonoros.
4. Agregar nuevos elementos al *script* para mejorar y complementar el sistema de registro de eventos en el archivo de salida. Algunas opciones son:

    a) Agregar una lista de los componentes añadidos para cada cuadro con el fin de verificar si fueron cargados correctamente.
    b) Modificar la tabla de registro de eventos de teclado para agregar información sobre si las teclas presionadas corresponden a las requeridas para una tarea específica. Una forma de realizar esto es con un *flag* que indique si la elección fue correcta.
    c) Para el caso en que el usuario detiene una sesión durante la ejecución: Incluir una ventana de diálogo para que pueda ser ingresado el motivo de la detención, agregando esta información al registro.

5. Incluir una mayor variedad de preguntas de usuario para el inicio de sesión de un experimento.
6. Incluir instancias para la pre-visualización de cuadros individuales y tareas.

Referencias

[1] K. R. Gegenfurtner, "The interaction between vision and eye movements," *Perception*, vol. 45(12), pp. 1333–1357, 2016. 3

[2] J. Findlay and R. Walker, "Human saccadic eye movements," *Scholarpedia*, vol. 7(7):5095, 2012. 3, 4

[3] M. Rucci and M. Poletti, "Control and functions of fixational eye movements," *Annual Review of Vision Science*, vol. 1, pp. 499–518, 2015. 4

[4] H.-k. Ko, M. Poletti, and M. Rucci, "Microsaccades precisely relocate gaze in a high visual acuity task," *Nature Neuroscience*, vol. 13, pp. 1549–1553, 2010. 4

[5] C. J. Luek [et al], "Antisaccades and remembered saccades in parkinson's disease," *Journal of Neurology, Neurosurgery, and Psychiatry*, vol. 53(4), pp. 284–288, 1990. 5, 12

[6] F. Chan [et al], "Deficits in saccadic eye-movement control in parkinson's disease," *Neuropsychologia*, vol. 43(5), pp. 784–796, 2005. 5, 12

[7] S. Amador [et al], "Dissociating cognitive deficits involved in voluntary eye movement dysfunctions in parkinson's disease patients," *Neuropsychologia*, vol. 44(8), pp. 1475–1482, 2006. 5, 12

[8] A. Srivastava [et al], "Saccadic eye movements in parkinson's disease," *Indian Journal of Ophthalmology*, vol. 62(5), pp. 538–544, 2014. 5, 12

[9] T. Eggert, "Eye movement recordings: methods," *Neuro-Ophthalmology*, vol. 40, pp. 15–34, 2007. 5, 6, 8, 9

[10] D. C. Richardson and M. J. Spivet, "Eye tracking: Characteristics and methods," *Encyclopedia of biomaterials and biomedical engineering*, vol. 3, p. 10281042, 2004. 5, 6, 8, 9

[11] E. B. Delabarre, "A method of recording eye movements," *American Journal of Psychology*, vol. 9(4), pp. 572–574, 1898. 5

[12] E. B. Huey, "Preliminary experiments in the physiology and psychology of reading," *American Journal of Psychology*, vol. 9(4), pp. 575–586, 1898. 5

[13] R. Dodge and T. S. Cline, "The angle velocity of eye movements," *Psychological Review*, vol. 8, pp. 145–157, 1901. 5

[14] G. Rakoczi, "Analysis of eye movements in the context of e-learning, recommendations for eye-efficient user interfaces," Ph.D. dissertation, Fakultät für Informatik der Technischen Universität Wien, 2014. 6, 8, 9

[15] B. Bauer, "A timely reminder about stimulus display times and other presentation parameters on crts and newer technologies," *Canadian Journal of Experimental Psychology*, vol. 69, pp. 264–273, 2015. 10

[16] P. Wang, "An lcd monitor with sufficiently precise timing for research in vision," *Encyclopedia of biomaterials and biomedical engineering*, vol. 5, 2011. 11

[17] T. Elze, T. G. Tanner, and B. Krekelberg, "Temporal properties of liquid crystal displays: Implications for vision science experiments," *PLoS ONE*, vol. 7(9), 2012. 11

[18] H. Strasburger, "Software for visual psychophysics: an overview," [consulta: 10 junio 2017]. [Online]. Available: http://www.visionscience.com/documents/strasburger/strasburger.html 11

[19] N. B. Systems, "Presentation software," [consulta: 13 junio 2017]. [Online]. Available: http://www.neurobs.com/ 12

[20] J. Peirce, "Psychopy," [consulta: 13 junio 2017]. [Online]. Available: http://www.psychopy.org/ 12

[21] M. Kleiner, "Psychtoolbox," [consulta: 13 junio 2017]. [Online]. Available: http://psychtoolbox.org/ 12

[22] A. D. Straw, "Vission egg," [consulta: 13 junio 2017]. [Online]. Available: https://visionegg.readthedocs.io/en/latest/index.html 12

[23] U. P. Mosimann [et al], "Saccadic eye movements changes in parkinson's disease dementian and dementia with lewy bodies," *Brain*, vol. 128, pp. 1267–1276, 2005. 12

[24] I. S. MacKenzie, "Evaluating eye tracking systems for computer input," in *Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies*, P. Majaranta [et al]., Ed. Hershey, PA: IGI Global, 2011, ch. 15, pp. 205–225. 15, 16

[25] J. Pierce, "Generating stimuli for neuroscience using psychopy," *Frontiers in Neuroinformatics*, vol. 2, p. 10, 2009. 17

[26] S. Simpson, "iohub," [consulta: 23 agosto 2017]. [Online]. Available: http://www.isolver-solutions.com/iohubdocs/index.html 21

Apéndice A

Lógica del script de ejecución

A continuación se muestra de forma general el funcionamiento del script de ejecución para los experimentos:

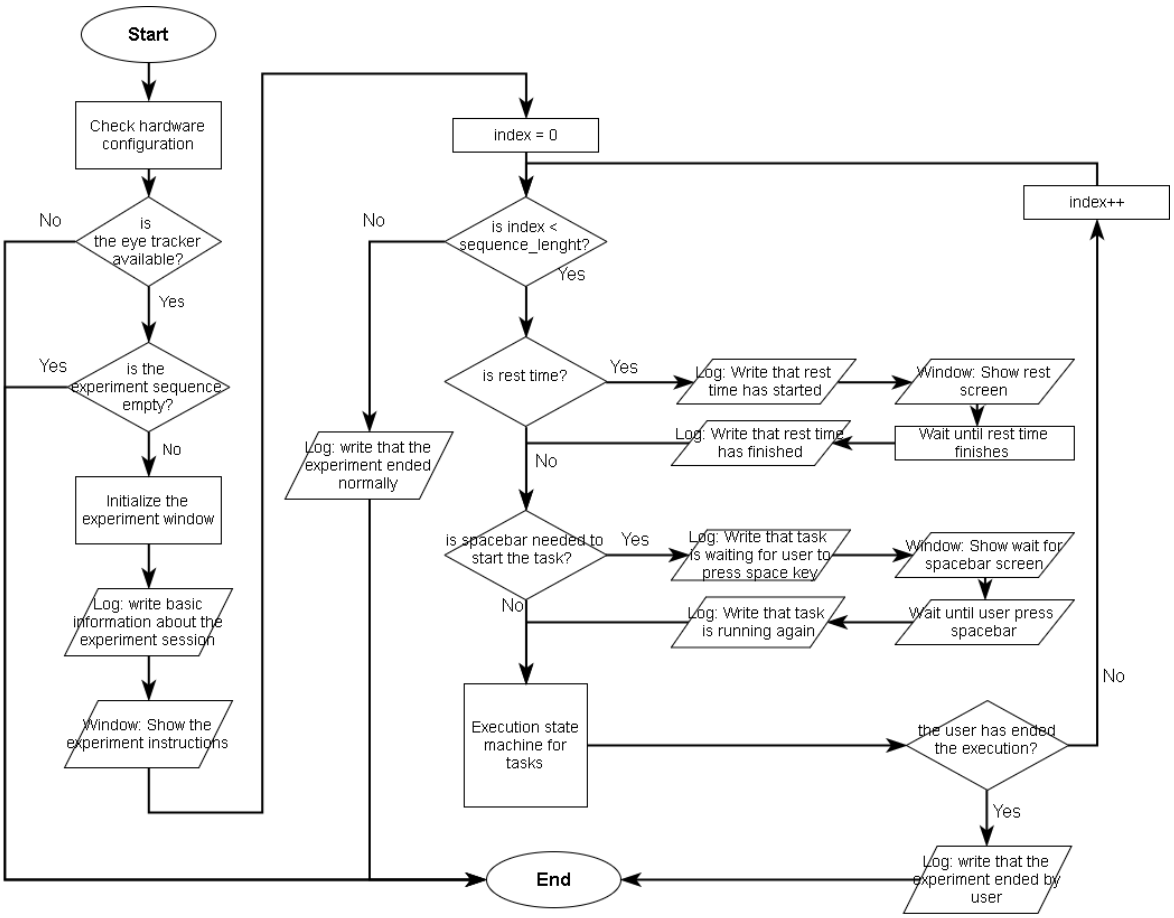


Figura A.1: Funcionamiento general del *script* de ejecución.

Apéndice B

Configuración del entorno de programación

Para el correcto funcionamiendo de *psychoPy* y por tanto, de esta aplicación, es necesaria la instalación de una serie de herramientas de software. Para un sistema con Windows de 64-bit es necesario instalar:

1. Librería para descodificación de video y audio AVbin10 en sus versiones de 32-bit y 64-bit. Descargar Aquí.
2. Reproductor multimedia VLC en su versión de 32-bit. Descargar Aquí.
3. Compilador de Microsoft Visual C++ para Python 2.7. Descargar Aquí.
4. Se necesita instalar la distribución Anaconda para uso con Python 2.7 en su versión de 32-bit. Aquí.
5. Sobre la instalación de Anaconda debe crearse el siguiente entorno de programación:

```
name: Memoria
channels:
  - menpo
  - jlaforet
  - cogsci
  - conda-forge
  - defaults
dependencies:
  - pygame=1.9.2a0
  - pyglet=1.2.4
  - alabaster=0.7.10
  - astroid=1.5.3
  - babel=2.5.1
  - backports=1.0
  - backports.functools_lru_cache=1.4
  - backports.shutil_get_terminal_size=1.0.0
  - backports_abc=0.5
  - bleach=2.0.0
  - bzip2=1.0.6
  - ca-certificates=2017.11.5
  - chardet=3.0.4
```

```
  - colorama=0.3.9
  - configparser=3.5.0
  - cycler=0.10.0
  - cython=0.24.1
  - decorator=4.1.2
  - docutils=0.14
  - entrypoints=0.2.3
  - enum34=1.1.6
  - et_xmlfile=1.0.1
  - freetype=2.6.3
  - functools32=3.2.3.2
  - future=0.16.0
  - h5py=2.6.0
  - html5lib=1.0.1
  - idna=2.6
  - imagesize=0.7.1
  - ipaddress=1.0.18
  - ipykernel=4.7.0
  - ipython=5.5.0
  - ipython_genutils=0.2.0
  - isort=4.2.15
  - jinja2=2.10
  - jpeg=9b
  - jsonschema=2.5.1
  - jupyter_client=5.2.1
  - jupyter_core=4.4.0
  - lazy-object-proxy=1.3.1
  - libiconv=1.15
  - libpng=1.6.34
  - libtiff=4.0.6
  - libwebp=0.5.2
  - libxml2=2.9.3
  - markupsafe=1.0
  - matplotlib=2.0.0
  - mccabe=0.6.1
  - mistune=0.8.3
  - nbconvert=5.3.1
  - nbformat=4.4.0
  - ndg-httpsclient=0.4.2
  - nibabel=2.0.2
  - numexpr=2.6.4
  - numpydoc=0.7.0
  - opencv=3.2.0
  - openssl=1.0.2n
  - pandas=0.18.1
  - pandoc=2.1.1
  - pandocfilters=1.4.1
  - path.py=10.3.1
  - pathlib2=2.1.0
  - pep8=1.7.1
  - pillow=3.2.0
  - pip=9.0.1
  - prompt_toolkit=1.0.15
  - py=1.4.31
  - pyasn1=0.1.9
  - pycparser=2.14
  - pydicom=0.9.9
  - pyflakes=1.6.0
  - pygments=2.1.3
  - pylint=1.8.1
  - pyopengl=3.1.1a1
  - pyopenssl=16.2.0
  - pyosf=1.0.4
  - pyparsing=2.2.0
  - pyqt=4.11.4
  - pyserial=3.4
  - pytest=2.9.2
  - python=2.7.14
```

```
  - python-dateutil=2.6.1
  - pytz=2017.3
  - pyxid=1.0
  - pyyaml=3.11
  - qt=4.8.7
  - qtawesome=0.4.4
  - qtconsole=4.3.1
  - qtpy=1.3.1
  - requests=2.10.0
  - setuptools=38.4.0
  - simplegeneric=0.8.1
  - singledispatch=3.4.0.3
  - sip=4.18
  - six=1.11.0
  - snowballstemmer=1.2.1
  - sphinx_rtd_theme=0.2.4
  - spyder=3.1.4
  - ssl_match_hostname=3.5.0.1
  - subprocess32=3.2.7
  - testpath=0.3.1
  - tornado=4.5.3
  - tqdm=4.7.2
  - traitlets=4.2.1
  - vc=9
  - vs2008_runtime=9.0.30729.6161
  - vs2015_runtime=14.0.25420
  - wcwidth=0.1.7
  - webencodings=0.5
  - wheel=0.30.0
  - win_unicode_console=0.5
  - wincertstore=0.2
  - wrapt=1.10.11
  - yaml=0.1.7
  - zlib=1.2.11
  - certifi=2017.11.5
  - cffi=1.6.0
  - comtypes=0.6.2
  - configobj=5.0.6
  - cryptography=1.4
  - gevent=1.1.1
  - greenlet=0.4.9
  - hdf5=1.8.15.1
  - icc_rt=2017.0.4
  - intel-openmp=2018.0.0
  - jedi=0.8.1
  - mkl=11.3.3
  - msgpack-python=0.4.7
  - numpy=1.11.2
  - pickleshare=0.7.2
  - psutil=4.2.0
  - pytables=3.2.2
  - pyzmq=15.2.0
  - rope=0.9.4
  - scipy=0.17.1
  - seaborn=0.7.0
  - sphinx=1.4.1
  - vs2010_runtime=10.00.40219.1
  - wxpython=3.0
  - xlrd=0.9.4
  - imageio=1.5
  - moviepy=0.2.2.11
  - ffmpeg=2.7.0
  - pip:
    - backports.ssl-match-hostname==3.5.0.1
    - egi==0.9.0
    - glumpy==1.0.6
    - h5pyviewer==0.0.1.6
    - hid==0.1.1
```

```
158     - jdcal==1.3
159     - openpyxl==2.3.5
160     - psychopy==1.84.2
161     - psychopy-ext==0.6.0.4
162     - pyfilesec==0.2.14
163     - pyparallel==0.2.2
164     - pypiwin32==219
165     - pysoundcard==0.5.2
166     - pysoundfile==0.8.1
167     - python-bidi==0.4.0
168     - svgwrite==1.1.7
169     - tables==3.2.2
170     - triangle==20170429
```

Al cual debe incluirse el módulo ”pyHook” en su versión 1.5.1 (descargar Aquí).