

# **Webcam-based eye tracking research for CSCL sessions**

**Matilla Miras, Adrián**

**Curs 2019-2020**



Universitat  
Pompeu Fabra  
Barcelona

Escola  
Superior Politècnica

**Treball de Fi de Grau**



## **Agradecimientos**

Para empezar, me gustaría agradecer a mi familia, a mis padres y a mi hermano por darme confianza y apoyo en momentos difíciles, y motivarme a continuar. Quiero agradecer también a mis directoras de proyecto, Ishari y Davinia, por haberme aconsejado en momentos difíciles, por la ayuda y guía proporcionada y la paciencia a lo largo del trabajo aun con la dificultad añadida que ha supuesto gestionarlo a distancia. Gracias.



## Resumen

Con la integración de herramientas para mejorar el aprendizaje en entornos colaborativos en las aulas, recientemente ha crecido el interés en proporcionar soporte a los profesores que guían a los participantes de estas sesiones.

Investigadores de la Universidad Pompeu Fabra han desarrollado una aplicación para visualizar el nivel de compromiso/ implicación de los alumnos durante las sesiones en tiempo real. Aun así, se desconoce qué información presentada en la aplicación (estadísticas, información de progreso, etc.) ha sido más útil para los profesores durante la toma de decisiones a lo largo de la sesión.

Por lo tanto, el objetivo del proyecto es investigar cómo los sistemas basados en *webcam eye tracking* funcionan y el software existente, para proponer una solución viable y de código abierto para captar la mirada y la atención de los profesores durante el uso de la aplicación, lo que permitiría determinar qué información ha sido la más consultada.

**Palabras clave:** eye-tracker, eye tracking, webcam, código abierto, visión artificial, mapa de calor, mirada.



## **Abstract**

Along with the integration of technological tools that aim to enhance learning in classroom collaborative learning settings, recently an increased research interest has been paid on how to support teachers who guide collaboration in these spaces.

A tablet dashboard application has been implemented by UPF researchers to visualize learner's collaborative learning engagement in real-time. However, which information presented in the dashboard (statistics, progress information, etc.) has been most useful for the teachers during orchestration decision making processes is unknown.

Hence, the goal of this project is to investigate how webcam-based eye-tracking works, the existing software that allows the process and how it can be used to track eye movements in real-time, in order to propose an affordable opensource solution able to record teachers gaze during the dashboard's use.

**Key words:** eye-tracker, eye tracking, webcam, opensource, computer vision, heatmap, gaze.



## **Prólogo**

A continuación se expone el trabajo de fin de grado realizado por el estudiante Adrián Matilla Miras de la Universidad Pompeu Fabra, y dirigido por las directoras Ishari Amarasinghe y Davinia Hernández-Leo.

El objetivo principal del trabajo es realizar una investigación enfocada al *webcam eye/gaze tracking* y a los software existentes actualmente, con tal de determinar una solución de código abierto basada en cámaras web corrientes que permita determinar la mirada y focos de atención de los usuarios, con el fin de averiguar qué información de una dashboard usada en sesiones CSCL es la más utilizada por los profesores que la usan.

Se expondrá la planificación del proyecto, así como los conceptos básicos en relación al eye/gaze tracking, el software encontrado durante el proceso de búsqueda, el proceso de selección de la solución final, y las conclusiones finales tras el proceso de investigación.



# Índice

	Pàg.
<b>Agradecimientos.....</b>	<b>iii</b>
<b>Resumen.....</b>	<b>v</b>
<b>Abstract.....</b>	<b>vii</b>
<b>Prólogo.....</b>	<b>ix</b>
<b>1. INTRODUCCIÓN.....</b>	<b>1</b>
1.1    Objetivo del proyecto.....	1
1.2    Planificación.....	2
<b>2. CONCEPTOS BÁSICOS.....</b>	<b>5</b>
2.1    Estructura del ojo humano.....	5
a)    Tipos de movimientos oculares.....	6
2.2    Introducción al Eye Tracking y Gaze Tracking.....	7
a)    Proceso de calibración.....	7
2.3    Clasificación y tipos de eye-tracker.....	9
a)    Tabla comparativa de los tipos de eye-tracker establecidos.....	12
2.4    Herramientas y métricas para medir rendimiento y resultados.....	13
<b>3. WEBCAM EYE TRACKING.....</b>	<b>17</b>
<b>4. BÚSQUEDA DE SOFTWARE.....</b>	<b>21</b>
4.1    GazeRecorder.....	21
a)    Limitaciones.....	25
4.2    Sentigaze SKD.....	25
a)    Limitaciones.....	27
4.3    Matlab Projects.....	28

4.4	XLABS chrome extension.....	31
a)	Limitaciones.....	33
4.5	XLABS: Eyes Decide.....	33
a)	Limitaciones.....	34
4.6	OpenGazer.....	35
a)	Limitaciones.....	36
4.7	CVC Eye-Tracker.....	37
a)	Limitaciones.....	40
4.8	WebGazer.js.....	42
4.9	Tabla comparativa.....	43
4.10	Otro software encontrado.....	43
4.11	Selección final y problemas encontrados.....	44
<b>5.</b>	<b>MODIFICACIONES FINALES Y FASE DE TESTEO.....</b>	<b>47</b>
5.1	Estructura del código de CVC Eye-tracker.....	47
5.2	Posibles modificaciones planteadas.....	48
5.3	Testeo final.....	51
5.4	Planteamiento de la prueba a realizar con la Dashboard.....	53
<b>6.</b>	<b>CONCLUSIONES.....</b>	<b>57</b>
6.1	Reflexión.....	57
6.2	Sugerencias para posibles mejoras.....	58
<b>Bibliografia.....</b>		<b>59</b>

# 1. INTRODUCCIÓN

En este capítulo se introducen las motivaciones del proyecto, sus objetivos, así como la planificación de éste y el tiempo usado para cada fase.

Los beneficios de implementar actividades CSCL (*Computer Supported Collaborative Learning* [0]) en clases presenciales son conocidas. Junto con la integración de tecnología y herramientas que tienen como objetivo mejorar el proceso de aprendizaje en entornos de aprendizaje colaborativo, recientemente ha crecido interés en investigar métodos para mejorar y dar soporte a los profesores que guían el proceso de colaboración en las sesiones. Sin embargo, en el contexto de CSCL, la gestión de la colaboración en tiempo real ya de por si es una tarea desafiante para los maestros, que deben prestar atención a las interacciones entre alumnos de un mismo grupo o entre múltiples grupos simultáneamente durante la ejecución de las actividades realizadas.

Con el fin de proporcionar este soporte, investigadores de la Universidad Pompeu Fabra han implementado una aplicación para el control de datos (*dashboard*) que puede ser usada durante las actividades CSCL, y visualiza el compromiso de aprendizaje colaborativo de los alumnos participantes en tiempo real. Sin embargo, se desconoce qué información presentada en la aplicación (estadísticas, progreso, etc.) ha sido la más útil para los maestros durante los procesos de toma de decisiones.

## 1.1 Objetivo del proyecto

El objetivo del proyecto es investigar cómo funcionan los sistemas basados en detección de la mirada, y el software que existe actualmente para llevar a cabo un proceso de *gaze tracking* mediante el uso de webcams convencionales, con el fin de determinar si existe una solución viable y de código abierto capaz de captar la mirada de los profesores y su foco de atención durante el uso de la aplicación, lo que permitiría determinar qué información de la aplicación ha sido la más consultada durante las sesiones CSCL.

## 1.2 Planificación

La planificación inicial planteada al inicio del proyecto consta de 3 fases:

- **Fase 1: Investigación inicial.** En esta fase se establecen los objetivos del proyecto y se inicia la investigación sobre el funcionamiento de los eye-trackers y del proceso de detección de la mirada, así como las métricas utilizadas para la evaluación de estos.
- **Fase 2: Búsqueda de software.** Esta fase se centra en la búsqueda de software capaz de realizar el proceso de *gaze tracking* mediante una webcam convencional. Se explorarán las opciones actuales, priorizando aquellas que sean de código abierto y que no requieran modificación alguna de hardware.
- **Fase 3: Testeo y conclusión.** Corresponde a la fase de testeo de las diferentes opciones encontradas, con el fin de encontrar una solución viable que pudiera ser usada como webcam eye-tracker. Posteriormente, los resultados de la investigación serán reportados en la memoria final del proyecto.

El siguiente diagrama de Gantt muestra una organización inicial aproximada del proyecto:

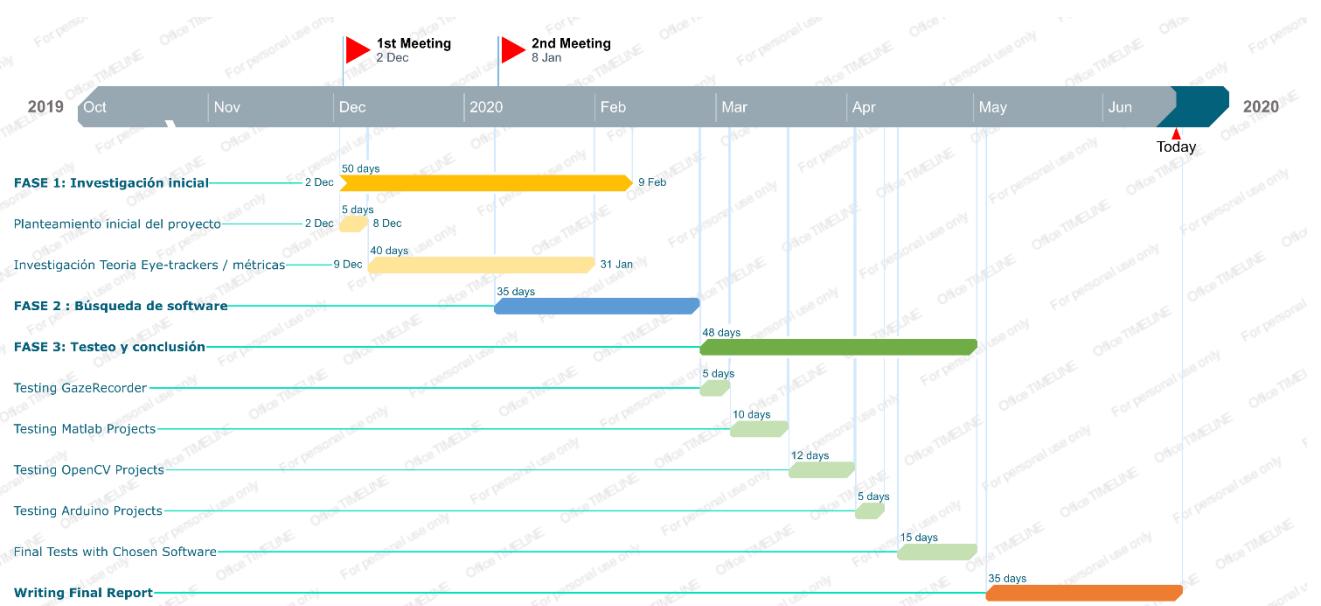


Figura 1.1. Diagrama de Gantt inicial.

Debido a la situación externa provocada por el COVID-19 a partir del mes de marzo, algunos de los plazos y fases no se cumplieron acorde a la aproximación inicial. El siguiente diagrama corresponde a la versión final:

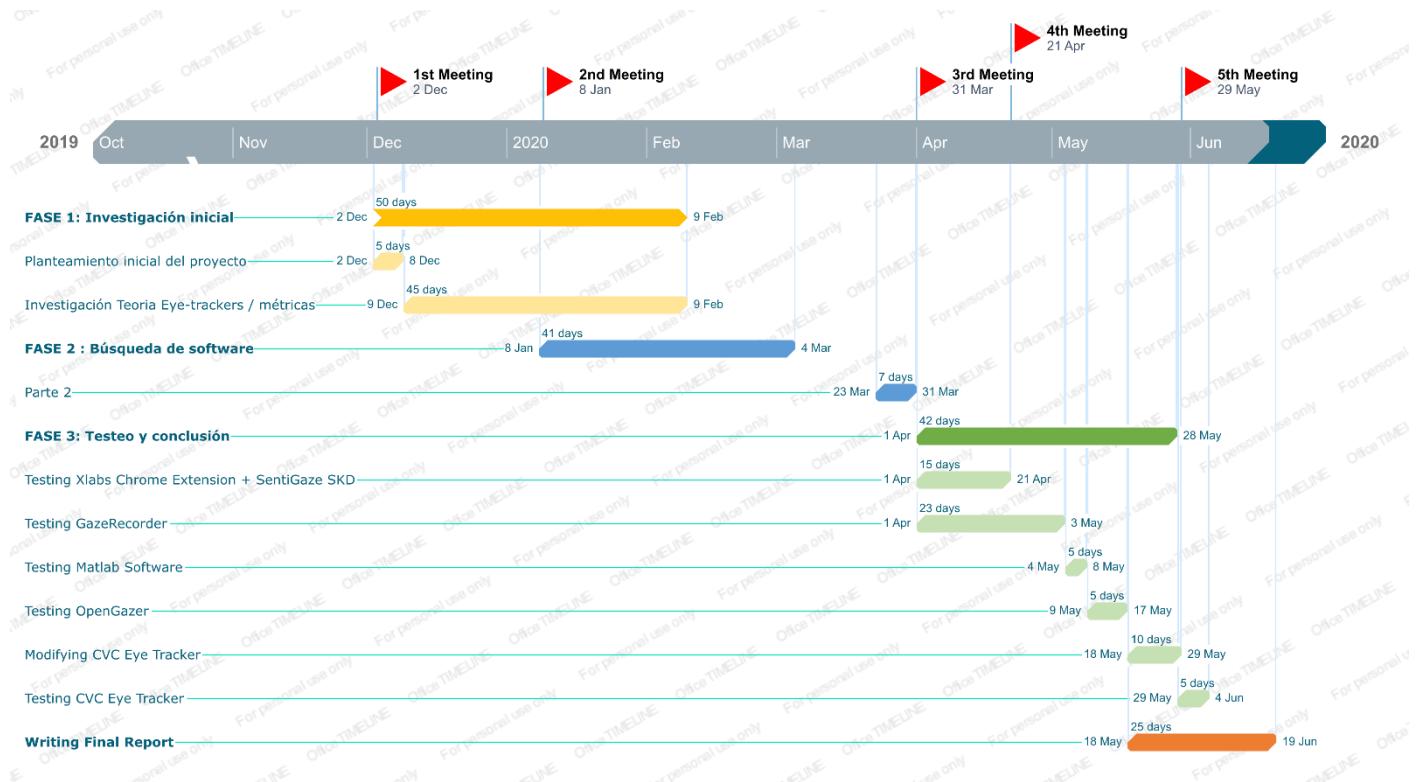


Figura 1.2. Diagrama de Gantt final del proyecto.



## 2. CONCEPTOS BÁSICOS

En este capítulo se introducirán conceptos básicos necesarios para entender el proceso de *eye* y *gaze tracking*, el funcionamiento de los eye-tracker y las distintas métricas usadas para evaluar los resultados y la eficiencia de éstos.

### 2.1 Estructura del ojo humano

Nuestros ojos son una de las principales herramientas que usamos para la toma de decisiones, por lo que campos como el *eye tracking* y el *gaze tracking* han sido objeto de estudio de empresas e investigadores [1].

Para entender en mayor medida el proceso de análisis de la mirada y el funcionamiento de los eye-trackers para determinar coordenadas y dirección hay que tener en cuenta ciertos conceptos de la estructura del ojo humano [2]. La figura 2.1 muestra las partes relevantes del ojo que engloba el campo del *gaze tracking* [3]:

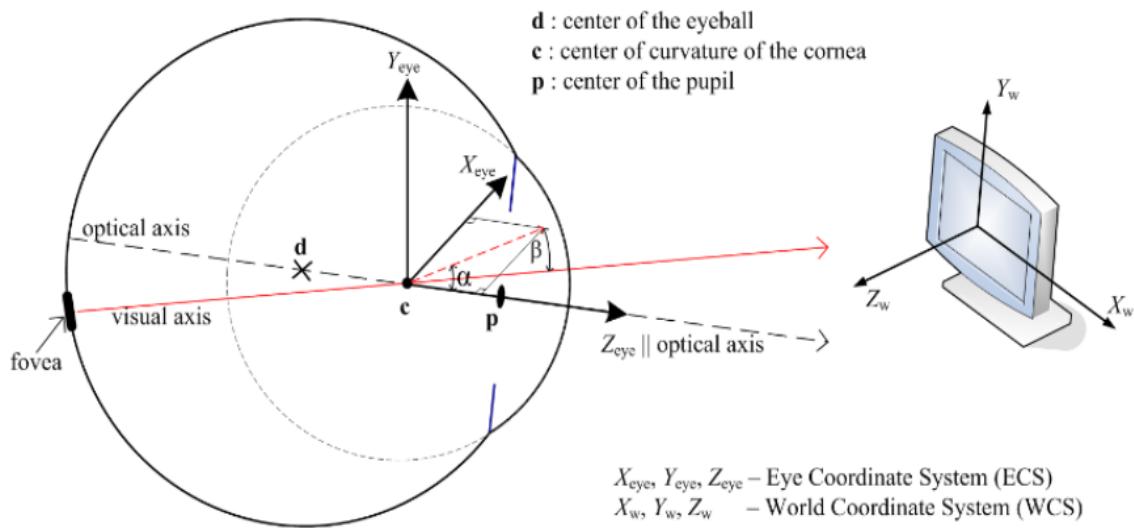


Figura 2.1. Estructura del ojo humano.

La cornea<sup>1</sup> (una capa transparente en la parte frontal del ojo) deja pasar la luz hacia el interior del ojo a través de la pupila. El iris (la parte colorida alrededor de la pupila) controla la cantidad de luz que se filtra hacia el interior, y ésta a través de lente llega a la retina, siendo la fóvea la zona en la retina donde mayormente se dirige la atención visual. La parte blanca del ojo corresponde a la esclerótica. Cuando una persona mira a distintas direcciones, tanto el centro de la córnea (c) como el centro de la pupila (p) rotan en función al centro del ojo (d) [4].

Tanto el eje óptico como el eje visual mostrados en la figura 2.1 son necesarios para determinar el PoG<sup>2</sup>. El eje óptico (*optical axis*) conecta el centro de la pupila con el centro de la córnea. El eje visual (*visual axis*), que indica la dirección del PoG, es la línea que conecta la fóvea con el centro de la córnea. El ángulo formado entre ambos ejes es el llamado ángulo kappa<sup>3</sup>. Tanto el eje óptico como dicho ángulo son detectados por los eye-trackers, con los que finalmente mapean el PoG en un plano o imagen 2D o en un volumen 3D<sup>4</sup>, determinando el eje visual [3].

### a) Tipos de movimientos oculares

Otro aspecto importante es el tipo de movimiento que realiza el ojo a la hora de fijar la atención con la mirada. En términos generales se clasifican en 3 tipos [2]:

- **Saccades:** Son los correspondientes a movimientos rápidos que realiza el ojo cuando cambia el foco de atención a un punto distinto. La duración aproximada de este movimiento es de entre 10ms y 100ms, periodo durante el cual el sistema ocular es considerado prácticamente ciego ya que es imposible fijar la atención. Un ejemplo de dichos movimientos podría darse cuando un usuario cambia de una ventana a otra en un navegador.
- **Smooth pursuits:** Estos movimientos se dan cuando la mirada sigue un foco de atención se encuentra en movimiento. La velocidad de dicho foco debe estar en

---

<sup>1</sup> Capa transparente en la parte frontal del ojo.

<sup>2</sup> Point of Gaze. Es el punto al que se dirige la mirada.

<sup>3</sup> Los angulos  $\alpha$  y  $\beta$  mostrados en la figura 2.1 son los componentes vertical y horizontal de kappa.

<sup>4</sup> También llamado Point of Regard (PoR) cuando el punto se mapea en un sistema de referencia 3D.

un rango suficiente para ser detectado por el ojo humano. Un ejemplo sería cuando una persona sigue un coche en marcha con la mirada.

- **Fixations:** Son los movimientos que estabilizan la retina en el foco de atención. Durante dichos movimientos, el eje visual del ojo apunta al foco de atención, y es cuando los PoG son estimables.

Además de dichos movimientos, se pueden dar otra clase de movimientos involuntarios como los pestañeos. El tener en cuenta dichos movimientos a la hora de optimizar los sistemas de *eye tracking* afectara en mayor o menor medida a la exactitud y precisión de los resultados.

## 2.2 Introducción al Eye Tracking y Gaze Tracking

La tecnología de seguimiento ocular, mejor conocida como *eye tracking*, es aquella que estudia y analiza información de los movimientos oculares. Engloba tanto la detección del ojo como el estudio de sus movimientos. El campo del *gaze tracking* permite, además, monitorizar la forma en que un usuario observa una imagen o una pantalla mapeando y extrayendo información tanto de la dirección como de las áreas en que fija su atención con la mirada, o por cuanto tiempo ha fijado su mirada en un punto determinado. Ambos campos se combinan para crear sistemas capaces de mapear la mirada y realizar estudios con los datos recogidos.

Para llevar a cabo el proceso de detección de la mirada, es necesario combinar un sistema de *eye tracking* para la detección del ojo, y un sistema de *gaze tracking* para la estimación de los PoG. Adicionalmente, y dependiendo de los sistemas usados, será necesario un tercer sistema para la detección de la posición de la cabeza del usuario y un proceso de calibración de la cámara [5].

### a) Proceso de calibración

Para muchos de los sistemas dedicados al *Gaze Tracking*, un proceso de calibración de la cámara o dispositivo que recibe el input visual es necesario para llevar acabo la estimación de los PoG. Durante este proceso, el sistema mediante la cámara detecta y

determina parámetros y datos relacionados con el ojo del usuario y sus movimientos, necesarios para realizar las estimaciones que determinaran las coordenadas de la mirada.

Este proceso es comúnmente realizado haciendo al usuario observar un conjunto de puntos de posición fija (y en algunos casos en movimiento) en la pantalla en que se realiza el experimento, los cuales tiene que seguir con la mirada durante un tiempo determinado. Hay que tener en cuenta, según el modelo usado, la cantidad de puntos a usar, la colocación de dichos puntos en la pantalla, el tiempo de observación asignado a cada punto, los datos que el sistema necesita recoger durante el proceso y las condiciones externas (iluminación, movimiento del usuario) [6].

Comúnmente, el patrón más usado por los eye-trackers es una cuadrícula de 9-25 puntos distribuidos uniformemente por la pantalla, aunque existen proyectos menos de 9 puntos,

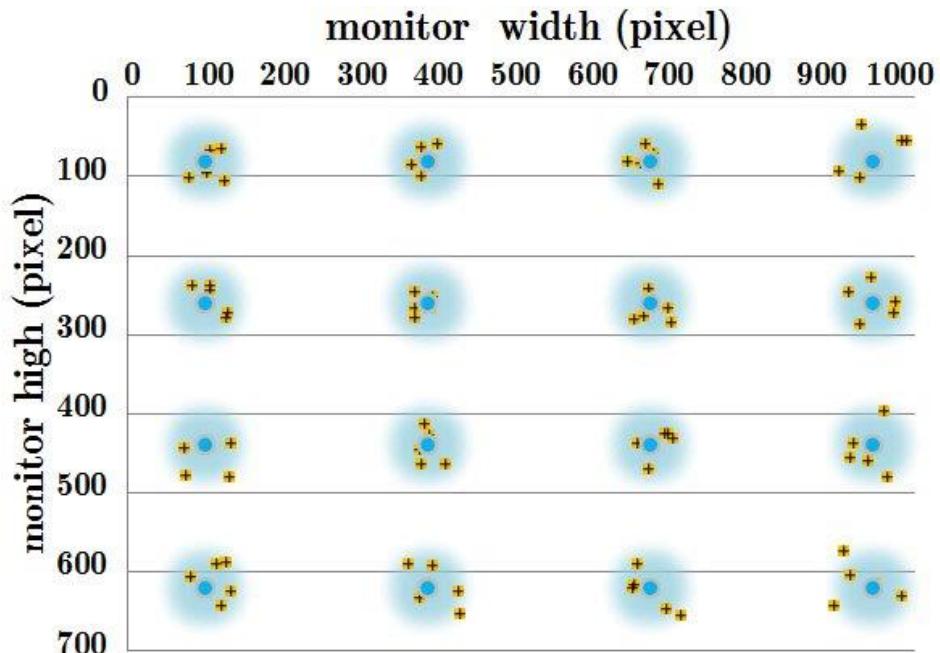


Figura 2.2. Resultado de la calibración de un DIY<sup>5</sup> eye tracker [7]. Los puntos azules son los usados para la calibración, y los amarillos corresponden a los gaze points detectados.

---

<sup>5</sup> DIY Eyetracker, o Do It Yourself EyeTracker, es un dispositivo para eye/gaze tracking no comercial, fabricado por los propios investigadores.

Es importante que el proceso de calibración no sea muy complejo y largo, ya que podría cansar o confundir a los usuarios, pero que dure lo suficiente para captar la información necesaria.

## 2.3 Clasificación y tipos de eye-tracker

La realización de estudios relacionados con el *gaze tracking* requieren de dispositivos para obtener un input visual, el cual es analizado posteriormente mediante el uso de software. Entre ellos encontramos distintos tipos de eye-trackers, desde hardware especializado para dichos estudios hasta webcams o modelos basados en webcams, que aun ofreciendo resultados menos exactos en muchos casos son suficientes para realizar estimaciones. Estos sistemas y modelos pueden clasificarse de distintos modos [2]:

- **Según el montaje del eye-tracker:** Distinguimos entre sistemas montados en la cabeza del usuario o invasivos, y sistemas remotos o también llamados no invasivos.

Entre los sistemas invasivos encontramos desde cámaras instaladas en la cabeza del usuario usando cintas, gorros o cascos hasta las conocidas *eye tracking glasses* [8]. Dichos sistemas permiten al usuario moverse libremente sin afectar al rendimiento y eficiencia del eye-tracker. Aun así, pueden llegar a resultar incomodos en un uso prolongado. Los sistemas remotos son una alternativa que, aun viéndose más afectada por los movimientos del usuario, puede obtener buenos resultados incorporando una buena calibración de la cámara [9].

- **Según el número de cámaras usado:** Distinguimos entre sistemas usando una sola cámara y sistemas que usan múltiples cámaras.

El hecho de usar múltiples cámaras en comparación a los sistemas tradicionales con una sola cámara permite a dichos sistemas obtener información en 3D de la mirada, y de la posición de la cabeza. Existen estudios que utilizan múltiples cámaras y focos de luz para generar un sistema que no necesita de proceso de

calibración para detectar los PoG [10]. Los sistemas de múltiple cámara son usualmente usados en sistemas invasivos.

- **Según el tipo de cámara usada:** Distinguimos entre sistemas IR<sup>6</sup> y sistemas en el espectro de luz visible.

Muchos de los eye-trackers existentes funcionan iluminando el ojo con una fuente de luz infrarroja. Los sistemas IR requieren de dicha fuente para funcionar, y deducen información de los reflejos de la luz IR en la córnea. Dicho reflejo es usado como punto de referencia para la estimación de los PoG. El uso de luz IR facilita también la detección de la pupila y el iris, ya que la mayor parte de la luz IR que pasa a través del iris es reflejada [11].

Muchas cámaras comerciales / convencionales son capaces de capturar el espectro de la luz IR, aunque en la mayoría de los casos esto requiere una modificación del hardware de las cámaras.

Los sistemas en el espectro de luz visible presentan diversas desventajas en comparación a los sistemas IR. En dichos sistemas es necesario un nivel mínimo de resolución a la hora de captar imágenes de los ojos. Al no poder detectarse el reflejo de la luz con la misma precisión que con la luz IR, dichos sistemas requieren de otros modelos y algoritmos para determinar el eje visual. No es viable usar fuentes de luz adicionales, pues incomodan al usuario provocando que se encoja la pupila.

Teniendo en cuenta las desventajas, se han realizado proyectos que realizan estimaciones en 2D [12] y 3D [4, 13, 14], en combinación con otros algoritmos y modelos y condiciones de luz ambiente ideales.

- **Según la fuente de luz usada:** Distinguimos entre sistemas que trabajan con una o más de una fuente de luz, y sistemas que trabajan con la luz ambiente.

---

<sup>6</sup> Luz infrarroja, de menor frecuencia que la luz visible.

- **Según el método usado para estimar la mirada:** Distinguimos dos tipos distintos según esta clasificación, sistemas basados en un modelo ocular, y sistemas basados en la apariencia.

El objetivo de los sistemas basados en un modelo es crear un modelo 2D o 3D del ojo o la cara del sujeto, para de esta forma estimar el eje visual y determinar los PoG. Dichos modelos se generan detectando distintas características y variables respecto al ojo, como el contorno de la pupila, durante el proceso de calibración de la cámara.

Usualmente los modelos 2D se basan en funciones de transformación polinómicas para mapear el eje visual del ojo en coordenadas de la pantalla o imagen.

En cuanto a los modelos 3D, se basan en modelos geométricos para detectar los ejes óptico y visual, el centro de la córnea y determinar las coordenadas en la pantalla [15]. Generalmente requieren de un proceso de calibración, aunque hay proyectos que han conseguido el mapeado sin el proceso de calibración mediante el uso de múltiples cámaras y focos de luz [10].

En cuanto a los sistemas basados en la apariencia, no se genera un modelo ocular con el cual trabajar, si no que básicamente se utilizan imágenes del ojo dirigiendo la mirada a posiciones conocidas para generar coordenadas. Las imágenes se usan como parte de un dataset para usar/entrenar modelos de machine learning como neural networks o Gaussian Process Regression [16], que determinan luego los PoG, usualmente en base a vectores generados con la intensidad de los píxeles de las imágenes. Estos sistemas son bastante sensibles a los movimientos del usuario, y no requieren de calibración de la cámara ya que el mapeo se calcula directamente mediante el contenido de las imágenes / *frames* del ojo captadas [15].

Puede considerarse una tercera categoría en esta sección, sistemas basados en *features*, que al igual que los basados en la apariencia mapean los PoG mediante modelos de *machine learning* pero usando vectores más complejos basados en la apariencia que los generados por la intensidad de los píxeles de las imágenes.

Este trabajo se centrará específicamente en sistemas remotos / no invasivos y en el espectro de la luz visible, ya que son los que más se ajustan al campo de *webcam eye/gaze tracking* sin requerir de una modificación de hardware.

a) Tabla comparativa de los tipos de eye-tracker establecidos

Sistema	Head movement tolerance	Necesita calibración	Detección de reflejos	Depende de luz ambiente	High Resolution needed
Invasivo	✓	-	-	X	-
No Invasivo	X	-	-	-	-
1 cámara	-	-	✓	-	-
Mult. cámaras	✓	X	✓	-	-
IR	-	-	✓	X	X
Luz visible	-	-	X	✓	✓
Fuente de luz	✓	-	✓	X	-
Luz ambiente	X	-	X	✓	-
Modelo	X	✓ / X	-	-	-
Apariencia	X	X	-	-	-

Tabla 1. Comparativa y features de los distintos tipos de eye-tracker. Las casillas marcadas con ‘-’ indican features no son relevantes para dicha clasificación.

## 2.4 Herramientas y métricas para medir rendimiento y resultados

La **exactitud** y **precisión** de un eye-tracker son dos de las métricas más usadas para definir su rendimiento y evaluar los datos obtenidos durante el proceso de *gaze tracking*. Estos indicadores representan la mayoría de los errores en los datos obtenidos, y son usadas como indicadores para determinar si dichos datos se consideran o no válidos.

Ambas métricas se ven afectadas por propiedades externas y movimientos del sujeto. Un sistema es considerado eficiente si su nivel de exactitud y precisión es bueno, ya que es capaz de determinar con menor cantidad de errores la mirada de los usuarios, y por lo tanto se considera que proporcionará datos y estimaciones más válidas [17.18].

La **exactitud** de un eye-tracker es la media de la diferencia entre los PoG que el eye-tracker ha detectado y el objetivo real al cual se dirigía la mirada. Se mide en ángulos visuales. Valores típicos de la exactitud de un eye-tracker comercial oscilan entre 0.1° (mayor exactitud) y 1° (menor exactitud) [18.20], aunque se han dado casos en que se registran valores mayores de 1° en sistemas remotos con condiciones externas controladas [2 ].

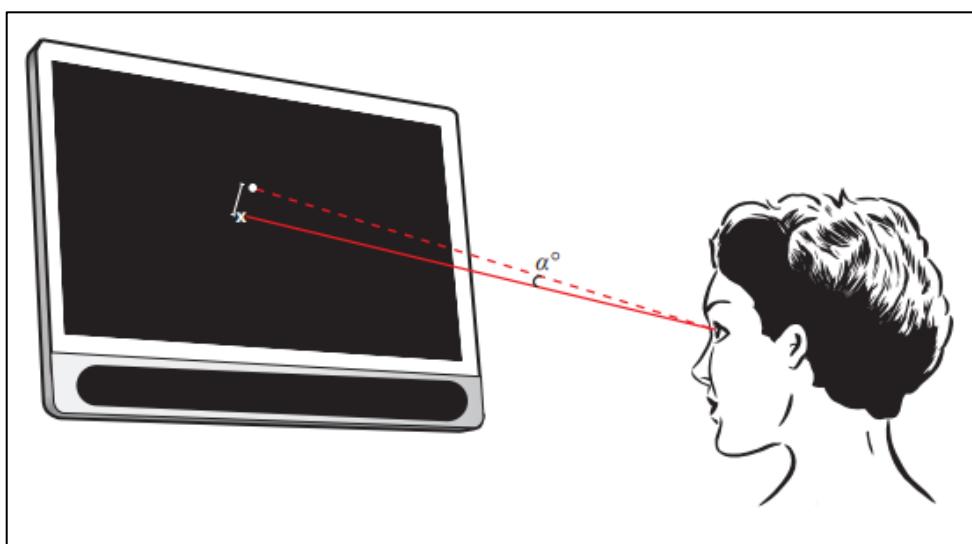


Figura 2.3. La línea discontinua indica el target real de la mirada del usuario, y la línea continua indica el Point of Gaze detectado por el eye tracker, correspondiendo  $\alpha$  al ángulo visual [18].

La **precisión** se define como la capacidad del eye-tracker de reproducir fielmente el mismo punto de *gaze*. Idealmente, en un caso de precisión perfecta, si el usuario mira durante un tiempo determinado al mismo punto, los puntos sucesivos trackeados deberían ser representados en el mismo lugar. Los valores correspondientes a la precisión, también medidos en ángulos visuales, se calculan mediante la RMS<sup>7</sup> de la distancia entre los diferentes puntos sucesivos trackeados cuando el usuario mira a un mismo punto [18,19].

Resumiendo ambos conceptos, la exactitud hace referencia al nivel en que los PoG trackeados coinciden con el objetivo de la mirada, y la precisión hace referencia al nivel de diferencia de los PoG trackeados para un mismo foco de atención.

La siguiente tabla muestra la relación exactitud – precisión con respecto a los puntos de mirada trackeados por un eye tracker [19]:

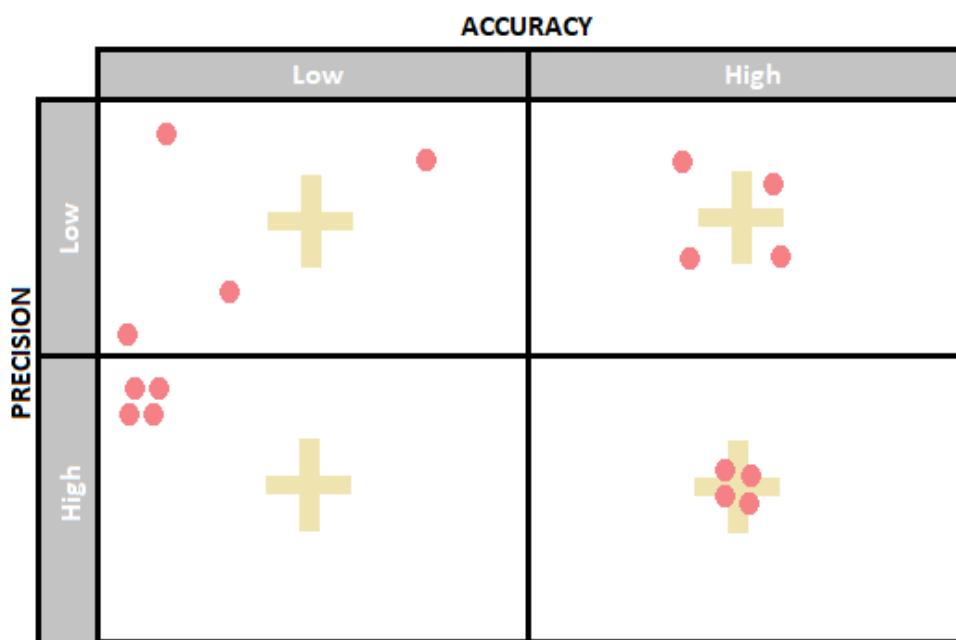


Tabla 2. Las cruces corresponden al objetivo real de la mirada. Los círculos rojos corresponden a los PoG trackeados por un eye tracker.

<sup>7</sup> Root Mean Square, o media cuadrática, corresponde a la raíz cuadrada de la media aritmética de los valores correspondientes al cuadrado.

Ambos indicadores son usualmente medidos durante el proceso de calibración, y en diversos estudios se estudian también en diferentes condiciones para determinar la robustez del sistema [21].

La **robustez** de un eye-tracker es un indicador que mide qué porcentaje de usuarios en la población son aptos para la extracción de los movimientos oculares mediante el dispositivo. Depende directamente de características físicas, como por ejemplo la etnia del usuario, la capacidad del dispositivo para detectar movimientos como el pestañeo o la detección de movimientos oculares cuando el usuario usa gafas [18].

En cuanto a herramientas para medir los resultados obtenidos tras mapear la mirada, encontramos entre los más destacados y utilizados los mapas de calor, o **heatmaps**. Los mapas de calor son representaciones visuales de las áreas donde el usuario ha mantenido el foco de atención a lo largo del proceso de *gaze tracking* [22]. El nivel de atención es representado por una escala de colores, que usualmente corresponde a la siguiente:

- **Azul** para un nivel prácticamente nulo de atención
- **Verde** para un nivel bajo de atención
- **Amarillo** para un nivel moderado de atención
- **Rojo** para un alto de nivel de atención, en las zonas rojas se ha fijado la mirada por un tiempo prolongado en una o varias ocasiones.

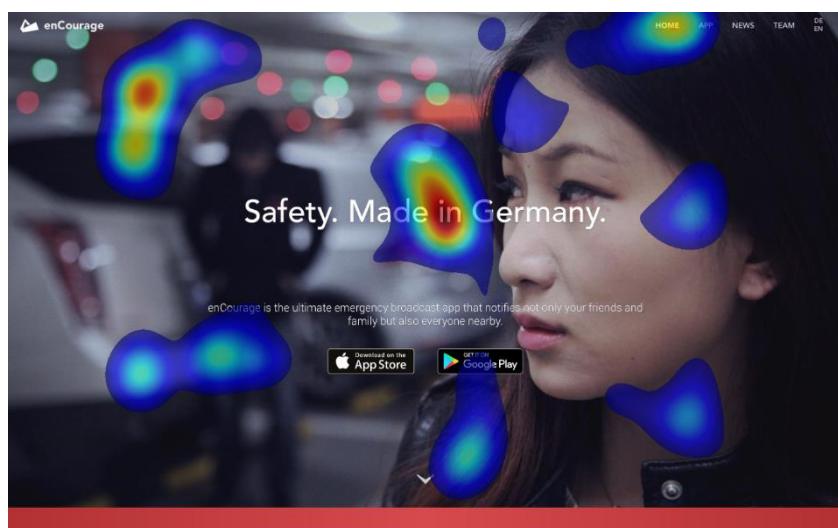


Figura 2.4. Ejemplo de mapa de calor de un eye-tracker.



### 3. WEBCAM EYE TRACKING

Este capítulo introducirá en específico el concepto de *webcam eye tracking*, el funcionamiento de los webcam eye-trackers y las ventajas y desventajas que suponen respecto a otros sistemas.

El proceso de *webcam gaze tracking* se basa en extraer la información necesaria para el proceso de *eye* y *gaze tracking* mediante sistemas que utilizan una o varias cámaras que detectan luz en el espectro visible para el ojo humano [23].

La forma en que sistemas basados en webcam para el *eye* y *gaze tracking* detectan los movimientos oculares y estima los PoG es bastante similar a la usada por algunos eye-tracker comerciales. El proceso incluye la detección de los movimientos de los ojos (excluyendo métodos basados en luz IR y detección precisa de la pupila y/o los reflejos de la luz), y un posterior mapeo de la mirada del usuario, que en muchos casos es realizada mediante un proceso de calibración.

Actualmente es usual que los sistemas de eye-tracking comerciales funcionen con cámaras especializadas de alta definición o que detectan espectros cercanos a la luz IR, en combinación con fuentes de luz para detectar el ojo humano de forma precisa, proporcionando resultados de alta exactitud. La mayoría de dichos sistemas están preparados para capturar el ojo sin verse afectado en gran medida por la luz ambiente, o los movimientos del usuario [8]. En comparación, los sistemas basados en webcams convencionales presentan diversas limitaciones [23]:

- **Resolución:** Una cámara convencional puede grabar imágenes de hasta alta definición, con un coste en latencia y *frame rate*. A menos definición, menor coste de latencia y *frame rate* supone el sistema, pero implica una cantidad menor de pixeles que describen la región del ojo, por lo que dificulta la obtención de imágenes precisas, bajando la exactitud [16].

- **Luz ambiente:** El hecho de que los webcam eye-trackers dependan de las condiciones de luz ambiente afecta directamente a la exactitud de los resultados. Cualquier cambio o malas condiciones de luz ambiente puede implicar una mala detección del rostro y los ojos del usuario.
- **Espectro de luz visible:** La limitación en cuanto al espectro de luz detectado por cámaras convencionales supone que sea imposible detectar la pupila de la misma forma que con los sistemas IR. Es posible detectar la zona correspondiente al iris y la pupila, sin ser posible diferenciar ambas claramente, por lo que sistemas basados en los reflejos de luz en la pupila no pueden ser usados.

Existen sistemas basados en webcams convencionales (DIY eye-trackers) capaces de detectar imágenes en el espectro de la luz IR, pero requieren de una modificación del propio hardware de la cámara [7, 24].

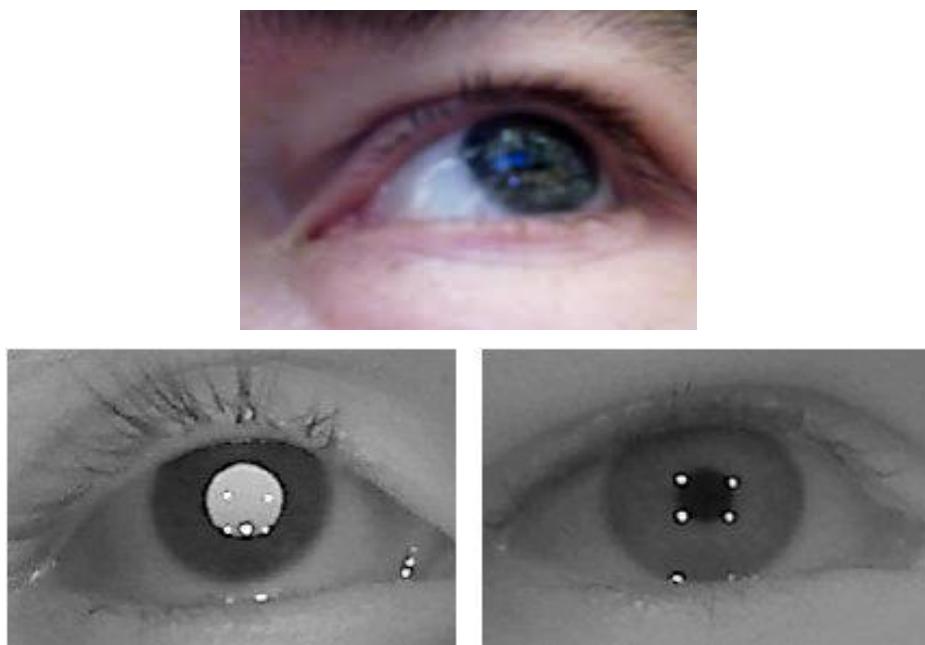


Figura 3.1. Diferencia de detección de la pupila entre una cámara convencional y una cámara IR [25,26].

El uso de algoritmos para reducir errores detectados durante la detección y estimación de la mirada y el uso de algoritmos para el tratamiento de imágenes o basados en *machine learning* son comúnmente usados en muchos de los sistemas existentes para

reducir la perdida de exactitud que suponen los webcam eye-trackers respecto a otros dispositivos comerciales especializados.

En comparación a los sistemas IR, los webcam eye-trackers presentan resultados de menor exactitud y precisión. Son sistemas muy sensibles al movimiento ya las condiciones exteriores, por lo que requieren ser complementados con algoritmos de detección facial o que el usuario mantenga una posición inmóvil durante el proceso de detección. La exactitud de los resultados se ve directamente afectada por la luz ambiente, y por la resolución y los fotogramas por segundo captados por la cámara.

Aun con ciertas limitaciones, presenta algunas ventajas respecto a los eye-trackers comerciales:

- Son una opción barata ya que solo requieren de una webcam convencional y en la mayoría de casos de un ordenador (los eye-trackers comerciales alcanzan precios aproximados de entre 100 euros para gamas bajas y 1000-10000 euros para dispositivos de gama media alta).
- No requieren de laboratorios, hardware o personal especializados para testear dichos sistemas, por lo que son más accesibles a la hora de realizar un estudio.
- Los resultados obtenidos, aun siendo de menor exactitud y precisión, son aceptables para muchos de los estudios realizados, como por ejemplo para captar la atención del usuario mientras visita una página web. Los eye-trackers comerciales alcanzan niveles de exactitud de entre  $0.1^\circ$  y  $1^\circ$ , mientras que los webcam eye-trackers alcanzan niveles de entre  $1.5^\circ$  y  $5^\circ$ .



## 4. BÚSQUEDA DE SOFTWARE

En este capítulo se expondrán los diferentes softwares encontrados durante la fase 2 del proyecto que permiten usar una webcam como eye-tracker para obtener las coordenadas de los PoG. Se describirán sus características y sus limitaciones, para una posterior comparación. Posteriormente se expondrán los problemas encontrados durante el proceso de selección. Finalmente, se expondrá qué software ha sido el escogido para ser presentado como solución para el objetivo del proyecto.

### 4.1 GazeRecorder

GazeRecorder, desarrollado por Szymon Deja [27], es un software de acceso gratuito para uso no comercial que permite detectar y mapear la mirada de la pantalla usando la webcam como eye-tracker. El software mediante el uso de diferentes algoritmos para identificar el rostro y los ojos del usuario del usuario genera un modelo 3d y posteriormente mediante un proceso de calibración mapea la mirada mediante algoritmos basados en análisis del flujo óptico (OF<sup>8</sup>) de las imágenes captadas por la cámara.

La interfaz de la aplicación es bastante simple y fácil de utilizar, por lo que es fácilmente usable por usuarios no expertos en el campo del *gaze tracking*. El programa te permite establecer ciertos parámetros iniciales, como por ejemplo la resolución de la webcam usada, si se quiere mostrar la mirada mapeada en pantalla durante los testeos realizados, la distancia entre ambas pupilas del usuario o el ángulo de inclinación de la cámara respecto al usuario. También permite determinar ciertos parámetros con relación a los reportes de resultados generados por el propio programa.

En las siguientes figuras se puede observar la interfaz principal de la aplicación, que consta de 3 ventanas: la ventana principal para iniciar los testeos y el proceso de calibración, la ventana para establecer ciertos parámetros relacionados con los resultados obtenidos al finalizar un testeo, y la ventana de parámetros generales:

---

<sup>8</sup> Mapea la dirección y rotación de la mirada desde un modelo 3D al plano de la pantalla en 2D.

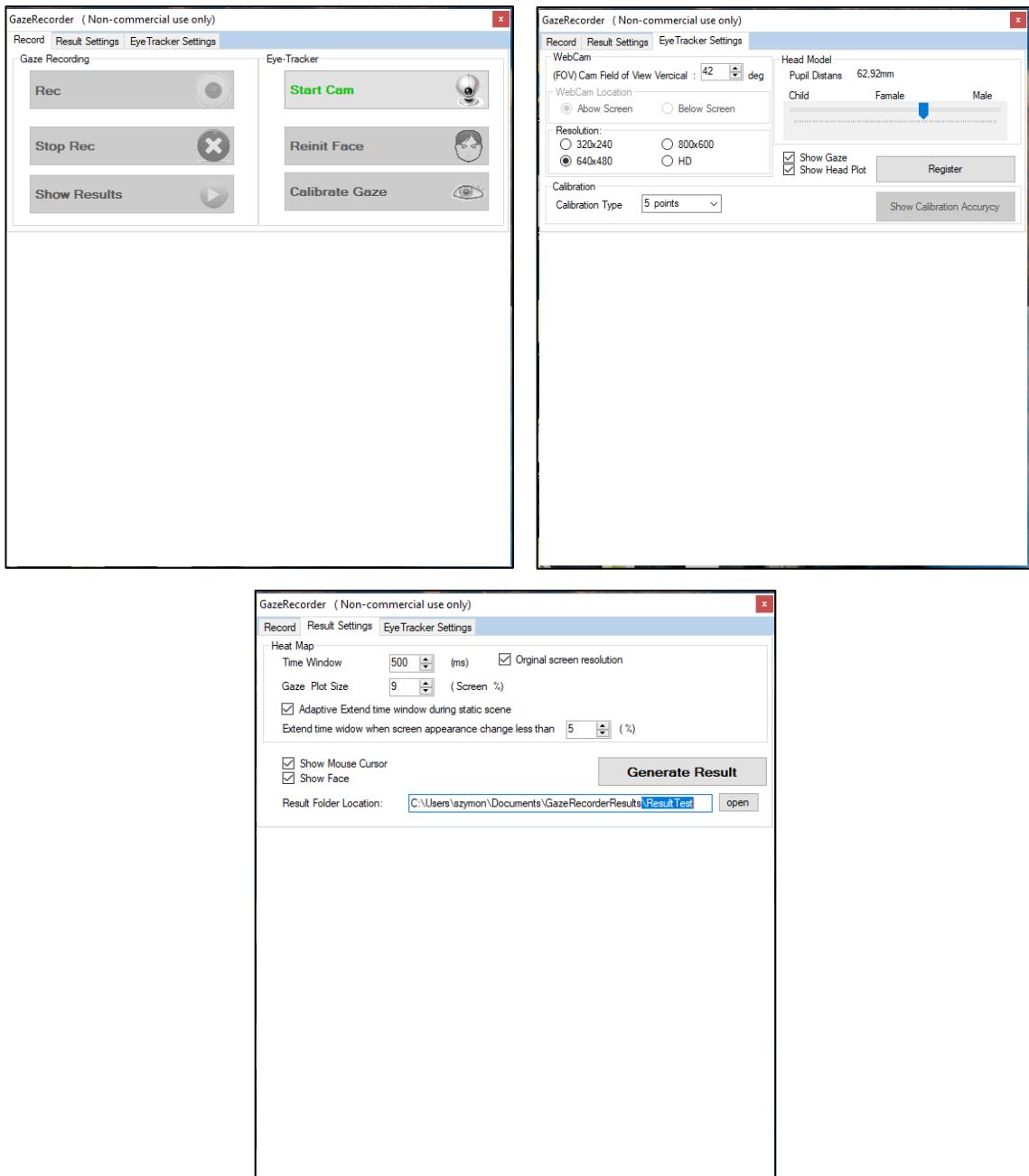


Figura 4.1. Interfaz de la aplicación de GazeRecorder.

El primer paso para usar el programa es iniciar el proceso de calibración. Pueden usarse 1,5, 9 o 16 puntos de calibración, a elección del usuario. El tiempo requerido para el proceso depende de los puntos usados. Una vez realizada, se genera un modelo en 3D del rostro del usuario en tiempo real, y es posible iniciar el *gaze tracking* pulsando el botón Rec. Se puede repetir la calibración para una mejor exactitud en los resultados o en el caso de que el rostro no haya sido detectado correctamente. Además, el software permite visualizar la exactitud y precisión con que se estima la mirada detectadas durante la calibración. El proceso depende directamente de las condiciones de luz

ambiente y de los movimientos del usuario, proporcionando mejores resultados si este mantiene una posición fija. Aun así, puede considerarse robusto, pues soporta el uso de gafas y proporciona resultados aceptables para un webcam eye-tracker una vez el proceso de calibración se realiza con éxito, con exactitud de entre 1° y 3° de ángulo visual.

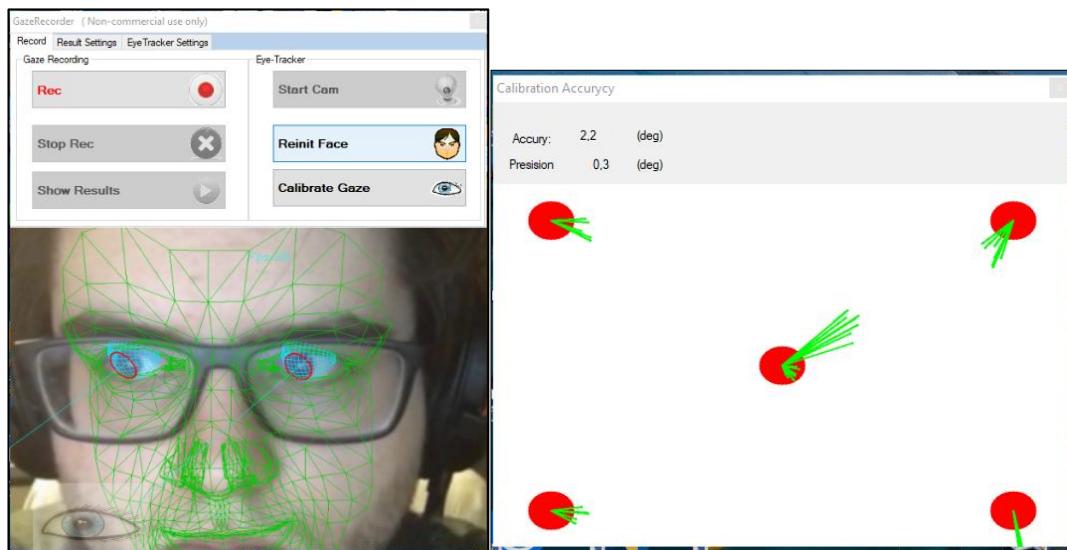


Figura 4.2. Modelo 3D generado después de la calibración y resultados de la calibración.

Después del proceso de grabación, el software genera automáticamente una carpeta con los resultados de la grabación, incluyendo coordenadas, mapas de calor estático y dinámico, y un video de la grabación del usuario a lo largo del uso del software.

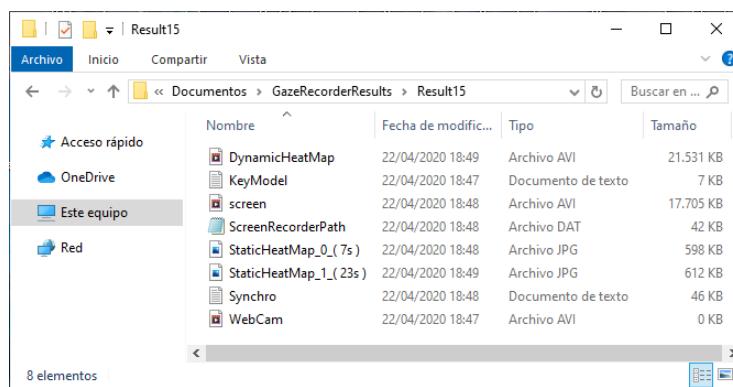


Figura 4.3. Resultados generados después de el proceso de gaze tracking

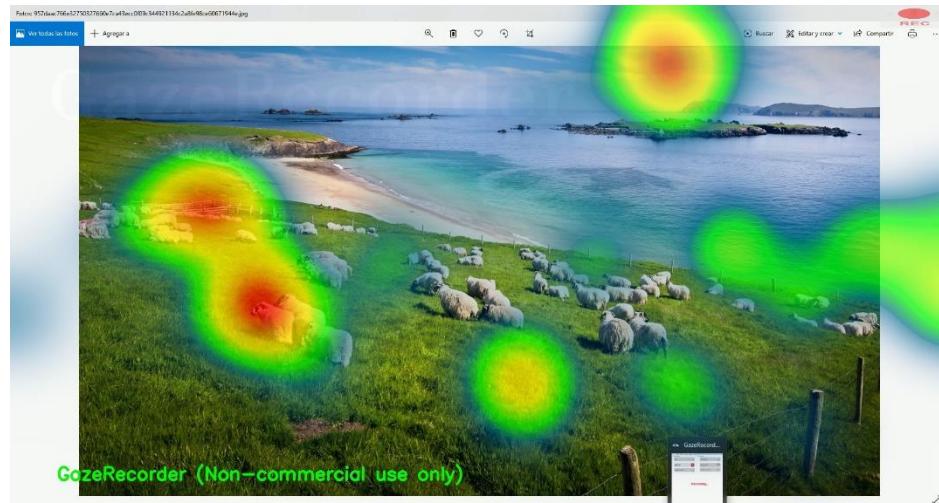


Figura 4.4. Mapa de calor estático generado por GazeRecorder.

GazeRecorder también dispone de una plataforma online mediante la cual permite realizar experimentos para registrar la mirada de una página web o una imagen o video. Para ello, es necesario crear una cuenta en la página web de la aplicación [28]. Una vez registrado, se crea un nuevo estudio insertando la página web o la imagen a ser observada, y la duración. La plataforma genera un enlace que permite a los participantes acceder al test con la imagen o web elegida, mediante el cual se registrara la mirada de estos. Una vez realizado el test, se genera un video con un mapa de calor dinámico al cual el creador del test puede acceder de forma online a través de su cuenta.

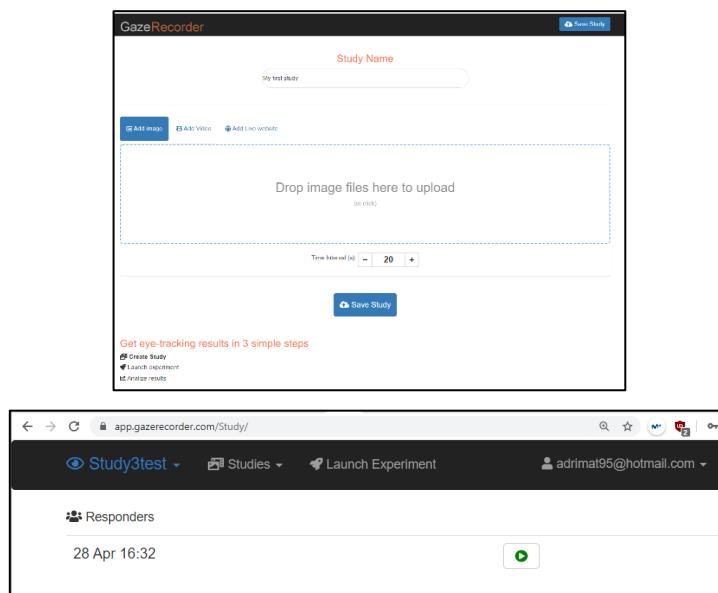


Figura 4.5. Interfaz de la plataforma online de GazeRecorder.

### a) Limitaciones

GazeRecorder presenta las siguientes limitaciones:

- La grabación de la mirada está limitada a 1 minuto, tanto en la aplicación como en la plataforma online.
- El sistema requiere de múltiples procesos de calibración para ofrecer resultados aceptables para un webcam eye-tracker.
- No se trata de un proyecto de código abierto, el software se encuentra bajo una licencia de uso no comercial con el objetivo de comercializarse en un futuro [29].
- En la versión online solo se pueden generar mapas de calor dinámicos a partir del estudio creado, a los cuales solo se puede acceder online, a diferencia de la aplicación de escritorio. Además, aún se encuentra en desarrollo, por lo que presenta errores y problemas de compatibilidad con algunas páginas web.

Aun con ciertas limitaciones, se trata de un software bastante completo, eficiente, y un posible y potencial candidato para tener en cuenta.

## 4.2 Sentigaze SKD

Sentigaze es un software que forma parte del kit de desarrollo de software creado por la empresa Neurotechnology [30]. Es capaz de realizar *gaze tracking* a tiempo real, guardando la sesión grabada y proporcionando resultados a tiempo real, en forma de mapa de calor.

Requiere de un proceso de calibración para detectar el tamaño de la pantalla y la posición del usuario. De esta forma, se adapta a distintos tamaños de pantalla y a la distancia a la que el usuario se encuentra de ella. El proceso de calibración se realiza de forma automática, mostrando varios puntos por pantalla. Aun así, es necesario realizarlo

antes de iniciar cualquier estudio/grabación de la mirada para asegurar resultados óptimos. Entre las recomendaciones proporcionadas, para asegurar un buen proceso de calibración el usuario debe estar a 45-80cm de la pantalla, y la webcam debe estar debajo de la pantalla. Por lo tanto, el software no es compatible con webcams incorporadas.

El software es compatible con otros productos de la empresa Neurotechnology, y puede ser usado para el control de aplicaciones o del ordenador usando la mirada como puntero.

La empresa, de forma gratuita, ofrece una versión de prueba con la funcionalidad de generar mapas de calor a partir del *gaze tracking*.

La interfaz del software es sencilla de usar, permitiendo seleccionar la cámara usada y el formato del input de la cámara, además de diferentes opciones con relación al proceso de calibración (puntos de calibración, color del fondo) y al mapa de calor generado. Una vez la cámara es detectada, puede iniciarse el proceso de calibración, para su posterior uso.

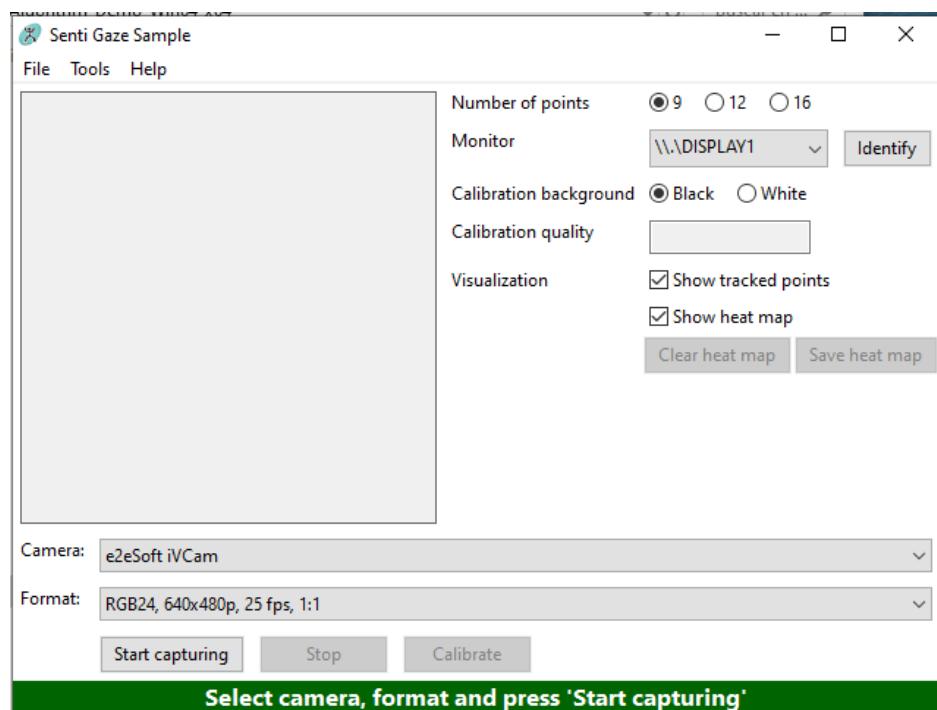


Figura 4.6. Interfaz de SentiGaze.



Figura 4.7. Mapa de calor generado por SentiGaze [31].

#### a) Limitaciones

- Es un software ligado a una licencia comercial, con una versión de prueba muy limitada. Neurotechnology ofrece un trial de la versión completa que dura 30 días.
- El software no está actualmente actualizado para funcionar en la versión actual de Windows 10 debido a problemas de compatibilidad con la detección de la cámara. Para su correcto funcionamiento, es necesario contactar con la empresa para obtener los archivos necesarios para actualizar la versión del software. La corrección proporcionada por la empresa solo es aplicable a la versión trial de 30 días, por lo que la prueba gratuita no es funcional.
- El proceso de calibración no ofrece información suficiente sobre la eficiencia y precisión detectadas. El software usa un indicador numérico propio para indicar la calidad de la calibración.
- Los resultados obtenidos son de baja eficiencia y precisión en comparación a otros software de licencia comercial, ofreciendo únicamente un reporte en forma de mapa de calor.

### 4.3 Matlab Projects

La mayoría de los proyectos de Matlab existentes hoy en día enfocados a trabajar con una cámara web convencional se basan exclusivamente en el proceso de *eye tracking* (detección del ojo), o en el tratamiento de la data obtenida de eye-trackers [31] (visualización de gráficos, *heatmaps*).

Los software basados en eye tracking se basan en la detección facial y de los ojos mediante el algoritmo de Viola Jones, y la transformada de Hough para la detección del iris y la pupila.

El detector basado en el algoritmo de Viola Jones, propuesto en el 2001 por Paul Viola y Michael Jones [32], se basa en tres pasos:

- Extracción de las características de imágenes correspondientes a un set de entrenamiento, también conocidas como *Haar features*. El proceso se realiza para todo un conjunto de imágenes con las características a detectar. Las características se dividen en 4 tipos, de las cuales 2 son las relevantes para la detección facial y de los ojos:

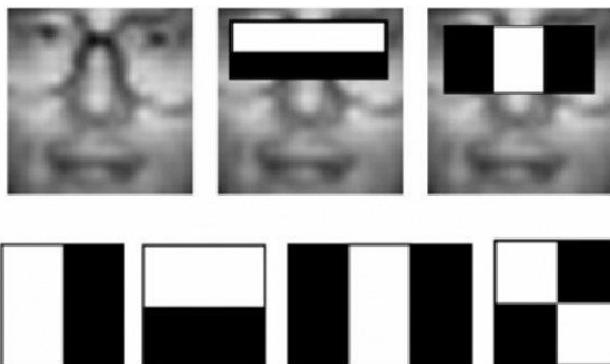


Figura 4.8. Haar Features [33].

- Clasificador en cascada que descartará las partes de las imágenes que sean menos relevantes a la hora de detectar el rostro, que aumenta su velocidad de aprendizaje con cada imagen priorizando las características más probables a encontrar.

- Detección de las características de la imagen.

La transformada de Hough es un algoritmo usado para la detección de formas en imágenes digitales. En este caso, se usa la detección de círculos para la detección del iris y la pupila.

Dos ejemplos de proyectos que usan esta metodología son los siguientes: Gaze-Tracker, desarrollado por Iván Munguía [34], y Eye Gaze, desarrollado por Tan-Phuc Le [35].

El software **Gaze-Tracker** detecta el movimiento de la pupila para mover el cursor en la dirección en que se mueve la pupila. Dispone de una interfaz sencilla, que permite pausar y reanudar la detección, ajustar la velocidad a la que se mueve el cursor, e indica de forma gráfica la pupila detectada y el centro.

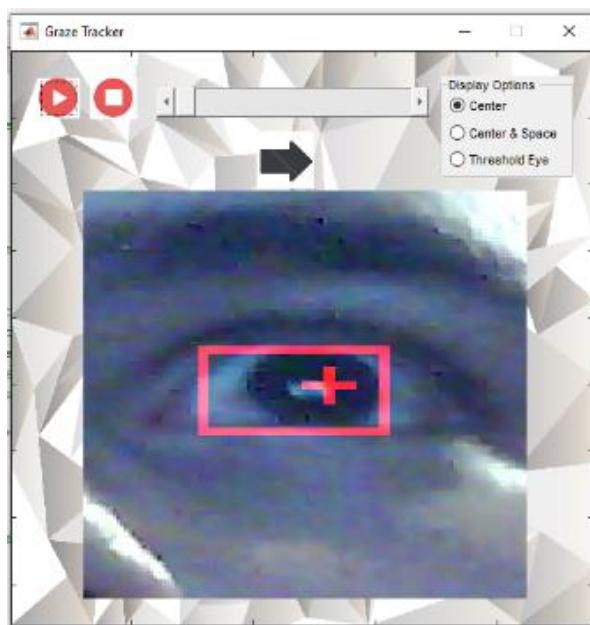


Figura 4.9. Interfaz de Gaze-Tracker

En cuanto a **Eye Gaze**, se limita exclusivamente a la detección del iris y sus movimientos, indicando la dirección a la que se mueve de forma gráfica. En la interfaz proporcionada por el software se aprecian tanto la detección facial como la detección ocular:

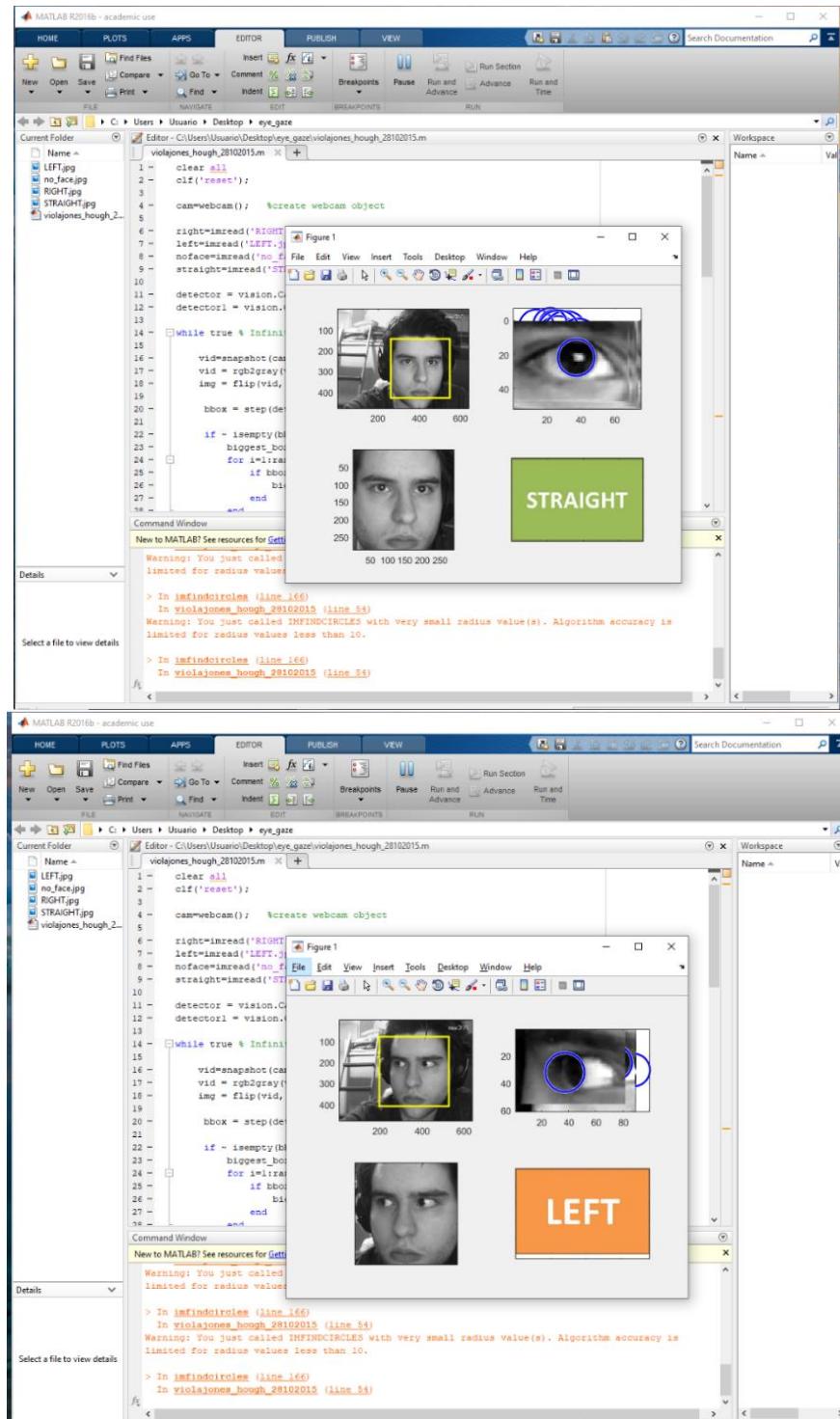


Figura 4.10. Interfaz de Eye Gaze.

Como limitación principal de estos proyectos hay que destacar que no disponen de un sistema de mapeado y estimación de la mirada en la pantalla implementado, centrándose exclusivamente en el proceso de *eye tracking*. Existen proyectos de Matlab con el proceso de gaze tracking implementado, pero dan soporte solo a sistemas IR o eye-trackers comerciales.

## 4.4 XLABS chrome extension

Esta extensión de Google Chrome creada por la empresa Xlabs permite a los usuarios usar el software de Xlabs para *eye / gaze tracking* y mouse pointer en cualquier página web. La extensión consta de un menú sencillo de varios modos activables. Cada uno de ellos ofrece una funcionalidad distinta en relación con el *eye* y *gaze tracking*. Es requerido el permiso de uso de la cámara web para el navegador [36].

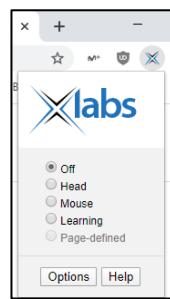


Figura 4.11. Menú de la extensión.

Los modos que ofrece la extensión son las siguientes:

- **Head Mode:** Este modo detecta y realiza un seguimiento de la posición de tu cabeza. En la pantalla un indicador en forma de dos cruces indica lo que está detectando la extensión, correspondiendo la cruz verde a la posición de la cabeza y la cruz azul a la orientación.

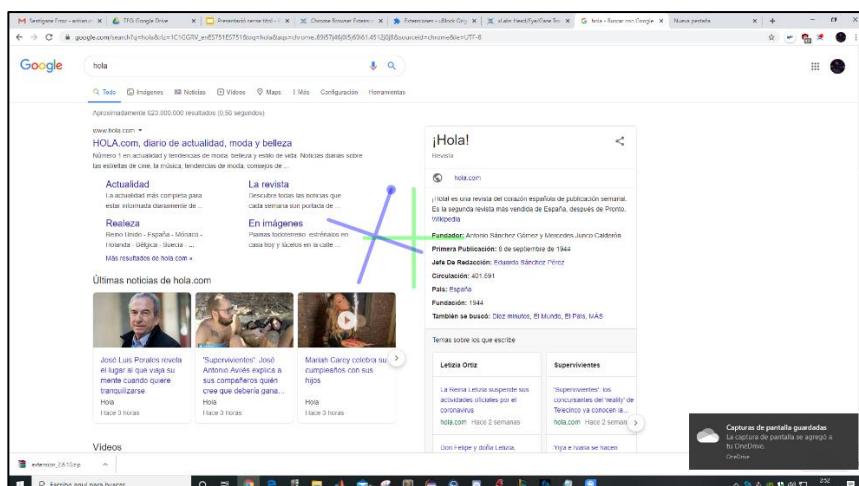


Figura 4.12. Head Mode activado.

- **Mouse Mode:** Con este modo es posible controlar el navegador sin usar el ratón. Mediante movimientos con la cabeza, la extensión navega por la página web seleccionada actuando como mouse pointer. Para este modo es necesaria una calibración que se inicia automáticamente una vez se activa. Con este modo activo, aparece un puntero rojo correspondiente a la posición de la cabeza, y mediante la orientación y los movimientos es posible desplazarse e interactuar con links.
- **Learning Mode:** Corresponde al modo de *gaze tracking*. Una vez activo, aparecerá un círculo rojo en la página web en que se esté usando que indicará la posición aproximada de los PoG detectados a tiempo real por la extensión. Este modo no necesita de un proceso de calibración, pues se auto calibra con cada clic que el usuario haga mientras navega por la página.

Para mejores resultados, es recomendable realizar clics por diferentes zonas de la pantalla mientras se mira el cursor. Eventualmente el círculo rojo estimará de forma más exacta la mirada. Es posible borrar los datos de calibración en caso de que se quiera cambiar de usuario o se produzca algún error, para que de esta forma la extensión vuelva a recalibrarse de forma automática. El circulo que indica la mirada cambia de tamaño, indicando mayor eficacia con menor tamaño.

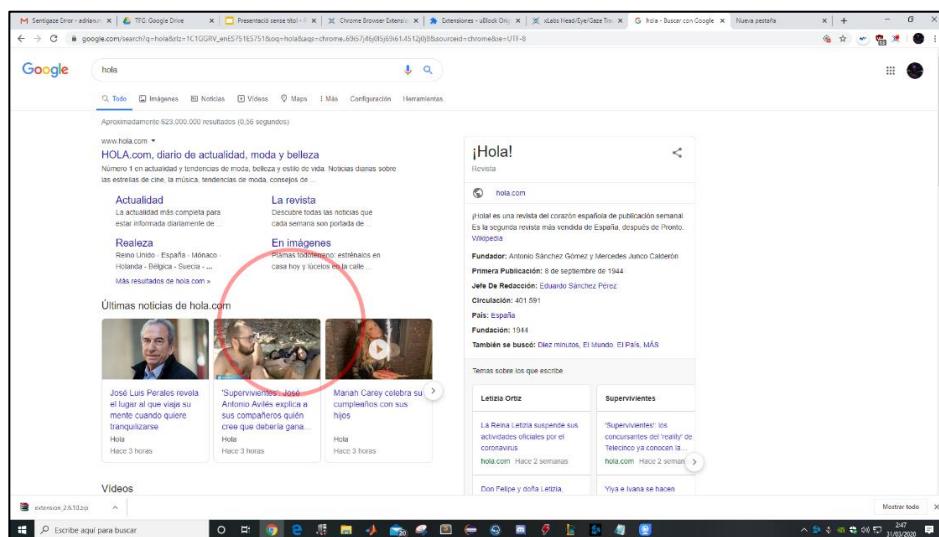


Figura 4.13. Learning mode activado.

- ***Page-Defined Mode:*** Este modo está destinado a páginas específicas que funcionan con Xlabs, no tiene efecto alguno en páginas web convencionales.
- ***Off Mode:*** Sirve para activar y desactivar la extensión.

La luz ambiente y condiciones externas afectan directamente a la eficiencia de los resultados. Para asegurar buenos resultados se recomienda no situar ningún foco de luz al alcance o detrás de la cámara, para evitar cualquier tipo de sombra sobre el rostro que dificulte la detección. El software presenta errores en el modo de *gaze tracking* cuando los usuarios usan gafas, y no está entrenado para reconocer caras de usuarios de 10 años o menos.

#### a) Limitaciones

La limitación principal de la extensión es que no es posible obtener los datos correspondiente al seguimiento de la mirada y detección facial para su correspondiente estudio. No es un proyecto de código abierto, así que para utilizar la API de la aplicación para modificarla o para obtener dichos datos habría que establecer un acuerdo con la empresa.

## 4.5 XLABS: Eyes Decide

Eyes Decide es una plataforma online dedicada a realizar estudios con la tecnología de webcam gaze tracking de Xlabs. Requiere la creación de una cuenta para acceder a sus servicios. La plataforma dispone de distintos niveles de cuenta, desde la cuenta gratuita hasta distintos niveles de cuenta que eliminan limitaciones en función a la tarifa pagada.

Eyes Decide permite crear estudios customizados a base de imágenes, incluyendo transiciones, animaciones, *triggers* y temporizadores. Dispone también de un panel para gestionar los participantes de dichos estudios, con la posibilidad de generar un link para invitar a participar. Es posible recibir participantes anónimos.

La plataforma dispone de herramientas de análisis para gestionar los resultados obtenidos en los estudios creados, incluyendo repeticiones en video del estudio de cada participante y generación de mapas de calor.

Figura 4.14. Interfaz de Eyes Decide.

#### a) Limitaciones

- El nivel de cuenta gratuito dispone de características limitadas. Hay un límite de estudios que se pueden crear, y un límite de participantes para dichos estudios, así como limitaciones a la hora de crear un estudio y usar herramientas de análisis de resultados. Para remover las limitaciones es necesario pagar una tarifa.
- La plataforma no funciona con páginas web existentes, requiere de la creación de estudios mediante imágenes.

## 4.6 OpenGazer

Opengazer es una aplicación de código abierto, escrita en lenguaje C++ y basada en OpenCV<sup>9</sup>, que va dirigida a usuarios que usen Linux y Mac OS [2]. El software usa una cámara web convencional para la estimación de la mirada. La información y data de las estimaciones realizadas pueden ser usadas por otras aplicaciones.

La primera versión de la aplicación fue desarrollada por Piotr Zielíns y su grupo de investigación en la Universidad de Cambridge, con soporte de Samsung y la Gatsby Charitable Foundation, y presentada como una alternativa low-cost a los eye-trackers comerciales [37].

El software sigue el siguiente proceso para realizar estimaciones:

- **Feature Point Selection:** Al iniciarse, el usuario debe seleccionar diferentes puntos característicos en su rostro determinantes para calcular la orientación del rostro. Corresponden como mínimo dos puntos a las corneas de los ojos y al menos otros dos a otras características faciales. Una vez seleccionados, el usuario puede guardar los puntos.
- **Calibración:** El proceso de calibración coincide con el de la mayoría de eye-trackers. Son mostrados diversos puntos rojos por toda la pantalla, a los cuales el usuario tiene que prestarles atención. El sistema es sensible al movimiento del usuario, por lo que cualquier cambio de posición requiere reiniciar el proceso de calibración.

OpenGazer extrae imágenes del ojo para cada punto, usadas en escala de grises para entrenar un sistema de *machine learning* basado en un proceso gaussiano<sup>10</sup>, que junto con cálculos basados en flujo óptico (OF) mapea los PoG en la pantalla.

---

<sup>9</sup> Biblioteca de código abierto dedicada a la Visión Artificial (Open Computer Vision), originalmente desarrollada por Intel. Engloba desde reconocimiento facial y de gestos hasta tracking o realidad aumentada.

<sup>10</sup> Sistema de aprendizaje lento que se usa para predecir correctamente los Gaze Points en base a los puntos detectados durante la calibración

- **Gaze Tracking:** Una vez el sistema ha sido calibrado y entrenado, el usuario carga de nuevo los *feature points* seleccionados al inicio del proceso, y mediante un detector de Viola Jones y *Optical Flow* los nuevos feature points son actualizados y se muestra por pantalla la mirada estimada por las imágenes del ojo y el proceso gaussiano. Las coordenadas de los PoG son calculadas usando solo las imágenes de un solo ojo.

En el siguiente video [38] se puede observar el proceso de calibración y testeo de Opengazer, correspondiendo los puntos rojos a los puntos de calibración, el punto morado al punto detectado como gaze, y los puntos verdes que forman parte del proceso de testeo.

Las coordenadas de los puntos de calibración son guardadas en el archivo “calpoints.txt”, permitiendo al usuario cambiarlas si lo desea, o cambiar la cantidad de puntos usados en la calibración. Además, proporciona como output dichas coordenadas por el Terminal usado. OpenGazer además abre un socket udp local que permite enviar las coordenadas de gaze a otros software sin necesidad de modificar el código base de la aplicación, lo que supondría una dificultad añadida a la hora de desarrollar nuevos software en base a OpenGazer.

Opengazer ha sido la base y referente de muchos de los proyectos de webcam gaze tracking basados en C++ y OpenCV que han surgido en los últimos diez años [37,39,40], siendo de los pocos proyectos de código libre con una implementación funcional para el webcam gaze tracking, aportando resultados de una exactitud de entre 1.6° y 2.5° aproximadamente [16].

#### a) Limitaciones

La principal limitación encontrada ha sido que el proyecto no ha sido mantenido y actualizado en los últimos años, surgiendo así problemas de incompatibilidad en el proceso de instalación, ya que el compilador necesario u las versiones de las librerías usadas son muy antiguos. Por este motivo, no ha sido posible el testeo de su versión original.

Entre otras limitaciones encontramos las siguientes:

- El proceso de ‘*Feature Point Selection*’ es manual, lo que puede causar que deba repetirse en mas de una ocasión, pues depende exclusivamente del usuario.
- El webcam eye-tracker no tiene en cuenta el movimiento causado por los pestañeos.
- Las estimaciones solo tienen en cuenta un solo ojo.

## 4.7 CVC Eye-Tracker

La necesidad de encontrar un modo para arreglar los problemas de compatibilidad con OpenGazer me redirigió al software llamado CVC Eye-Tracker.

CVC Eye-Tracker es un software de código libre basado en el proyecto de OpenGazer. El software es un proyecto resultado de la investigación del mantenedor no oficial del proyecto, Onur Ferhat, junto con investigadores del Computer Vision Center (CVC) [40] y la Universidad Autònoma de Barcelona [41].

Debido a que OpenGazer cumple con los requisitos propios de un webcam eye-tracker funcional, el objetivo del desarrollo CVC Eye- Tracker fue mejorar el proyecto original para mejorar así la performance y los resultados obtenidos en procesos de *gaze tracking* y aumentar la robustez. Aunque el código base de OpenGazer no comentado de forma óptima, sigue una estructura modular simple que habilita una fácil extensión del sistema original.

Las mejoras presentadas por CVC Eye-tracker con respecto al proyecto original pueden observarse en el siguiente diagrama:

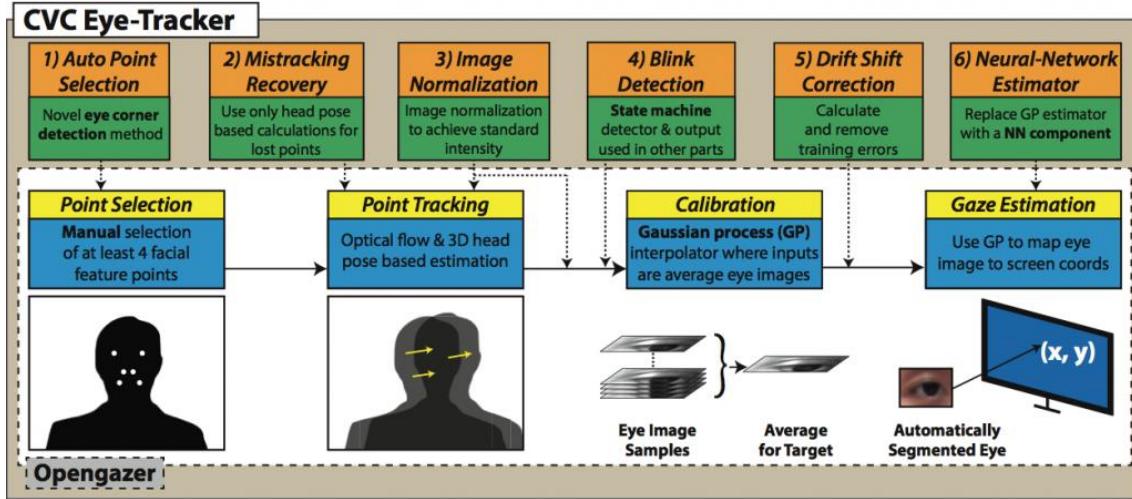


Figura 4.15. Diagrama de la implementación de CVC Eye-Tracker [42].

En el **proceso de selección de *feature points***, se ha automatizado todo el proceso para reducir errores generados por la operación manual. Combina detectores de *Haar features* en cascada que detectan la región que contiene ambos ojos y las *features* faciales, y un método novel de detección, que junto a cálculos geométricos determina los puntos exteriores de la córnea y hace uso de un detector Harris<sup>11</sup> para calcular el centro de ambas corneas [2, 16] y establecer un punto central en común (a diferencia de Opengazer, que solo usaba data referente a un sólo ojo) que será usado para determinar el eje visual y el PoG.



Figura 4.16. Detección automática de los feature points.

<sup>11</sup> Detector comúnmente usado para la detección de las córneas en el campo de Computer Vision.

En cuanto a la **fase de detección y seguimiento** de los PoG, se añade a la implementación inicial un filtro a las imágenes, y se realizan los cálculos de flujo óptico en base a la imagen captada inicialmente. De esta forma se reducen errores de tracking y se aumenta la robustez del sistema.

En cuanto a la **detección de pestaños**, Opengazer ofrece un componente no funcional destinado a esta función. Los desarrolladores de CVC Eye-tracker han arreglado este componente para hacerlo completamente funcional.

Durante el **proceso de calibración**, se han aplicado correcciones para tener en cuenta la detección de pestaños y excluir los datos generados durante esos movimientos. Se reducen además errores durante la calibración mediante la aplicación de dos interpoladores que se entrenan durante el mismo proceso, y generan la aproximación de los *gaze points* finales en base a los puntos de calibración y la mirada detectada.

En cuanto a la **estimación de la mirada**, al tener en cuenta las imágenes captadas de ambos ojos, se calculan dos estimaciones que se combinan realizando el promedio. Se ha implementado además el uso de un sistema basado red neuronal en lugar del sistema basado en proceso gaussiano para mejorar los resultados.



Figura 4.17. Software funcional mapeando la mirada mediante ambos ojos (punto verde).

Después de varios experimentos, los desarrolladores han observado que tanto la posición de la cámara, la estabilidad y características del usuario y las condiciones de luz ambiente siguen afectando de forma directa a los resultados obtenidos, aunque en algunos casos en menor medida, reportando valores de exactitud de entre 1.1° y 2.3° aproximadamente [16]. CVC eye-tracker además soporta la grabación en video de las sesiones para su correspondiente estudio offline, y la posibilidad de recolocar de forma eficiente los *feature points* en caso de producirse cualquier error durante su funcionamiento.

El proceso de instalación del software también se ha mejorado, renovando dependencias obsoletas y removiendo las innecesarias, simplificando el proceso y corrigiendo la mayoría de los problemas que presentaba OpenGazer. El software, testeado en un sistema Linux Ubuntu 12.04 de 32bits, es actualmente funcional, y un potencial candidato para los objetivos de este proyecto.

#### a) Limitaciones

- El componente que envía en red las coordenadas obtenidas durante el proceso de *gaze tracking* está desactivado, así como el output generado por el terminal.
- Al finalizar la calibración, como fase de testeo se inicia automáticamente un videojuego que usa la mirada estimada para mover una ventana en un fondo negro y tratar de encontrar una rana, que una vez encontrada, cambia automáticamente de lugar en la pantalla.
- Cualquier ventana no correspondiente al software sobrepuerta a la ventana de control del eye-tracker causa que la detección de la mirada pare y se pierda la calibración, así como obstáculo entre la cámara y el usuario.
- Aun habiendo mejorado la eficiencia en comparación al proyecto original, el software sigue siendo muy sensible a las condiciones de luz ambiente y a los movimientos del usuario, provocando que haya que recalibrar el sistema si ocurre cualquier error.

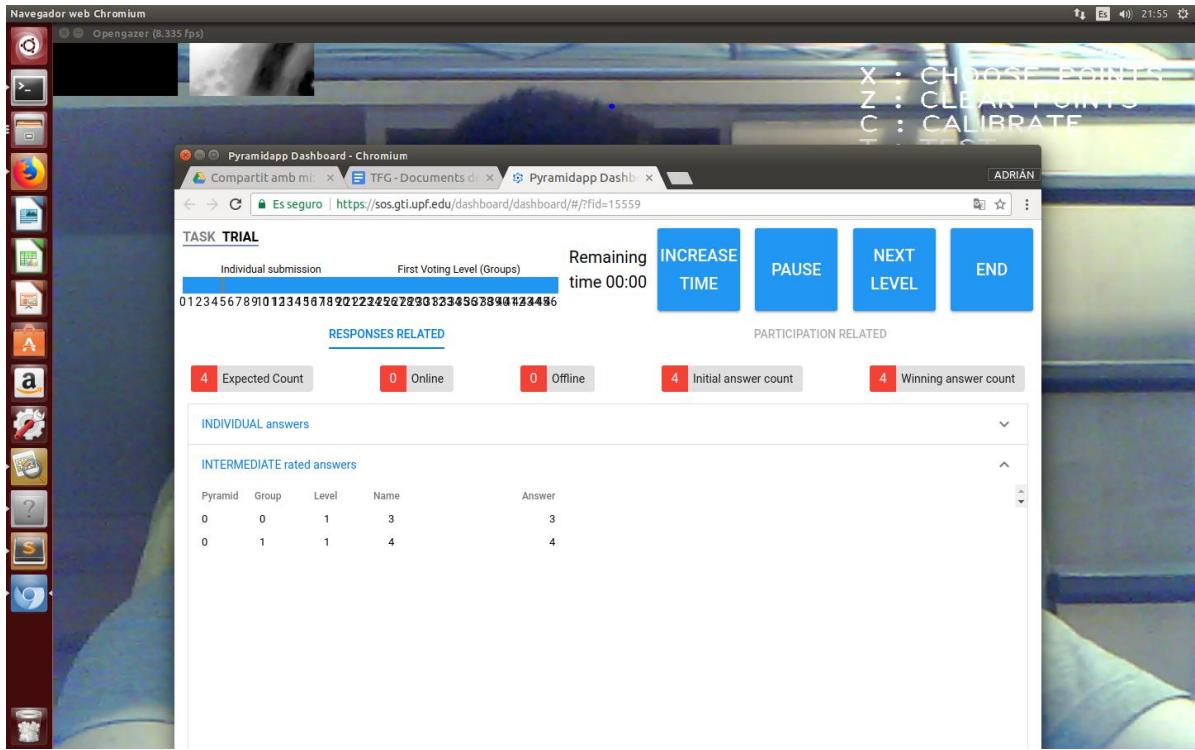


Figura 4.18. Fallo de calibración (esquina superior izquierda) al sobreponer la ventana de otro software.

La ventana usada como ejemplo es una muestra de la aplicación en cuestión a la que hace el proyecto referencia [42].



Figura 4.19. Fallo de calibración al sobreponer un objeto entre la cámara y el usuario.

## 4.8 WebGazer.js

WebGazer.js es una librería de *eye tracking* desarrollada por un grupo de investigación de la Brown University [43] que permite añadir a páginas web las herramientas necesarias para analizar y rastrear la mirada de los usuarios a tiempo real mientras navegan por la web, mediante el uso de una webcam convencional. Está escrita en JavaScript, y es fácilmente integrable en cualquier página que quiera disponer de dichas funcionalidades. Se ejecuta por completo en el navegador web, por lo que no es necesario enviar ningún tipo de input visual, tan solo dar permisos al navegador para usar la webcam. Webgazer.js es además soportado por la mayoría de los navegadores actuales. La única limitación es que requiere la modificación de la página web para usar todas las funcionalidades [44].

El modelo usado se auto calibra mediante los clics realizados por el usuario cuando interactúan por la página, y entrena un sistema de mapeado entre los movimientos del ojo y las posiciones en pantalla.

La página oficial, además de ofrecer el código necesario e instrucciones para incluir esta funcionalidad en tu propia página web, ofrece un data set [45] para entrenar inicialmente el sistema, y ejemplos para testear sus funcionalidades sin necesidad de instalar la librería.

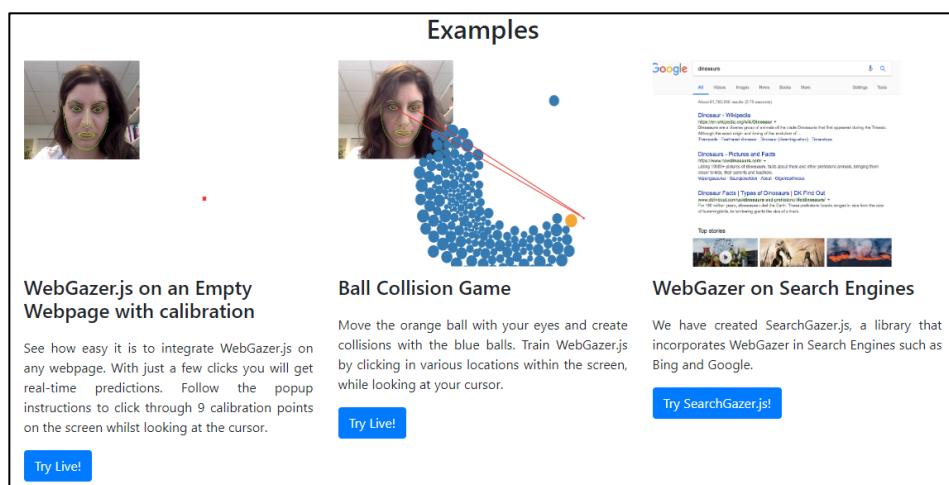


Figura 4.20. Ejemplos ofrecidos por WebGazer.js para testear la aplicación.

## 4.9 Tabla comparativa

EYE-TRACKER	Afectado por condiciones externas	Eye /Gaze /Face Tracking	Reporta la eficacia del sistema	Reporta resultados del trackeo	Genera mapas de calor	Outdated	Free / Código abierto
<b>Opengazer APP</b>	++	Face, eye, gaze	Si	Si	Si	No	Free con limitación
<b>Opengazer Online Platf.</b>	++	Face, eye, gaze	Si	No	Si	No	Free con limitación
<b>Sentigaze SKD</b>	+++	Face, eye, gaze	Si, pero con métrica no oficial	No	Si	Si	Trial limitado, comercial
<b>Matlab Projects</b>	+++	Face, eye	X	X	X	No	Free y Cód. abierto
<b>Xlabs Chrome ext.</b>	+++	Face, Eye, Gaze	X	No	No	No	Free
<b>Xlabs Online Platf.</b>	+++	Face, Eye, Gaze	Si	Si	Si	No	Free con limitación, comercial
<b>OpenGaze</b>	++++	Face, Eye, Gaze	No	Si	No	Si	Free y Cód. abierto
<b>CVC Eye-tracker</b>	+++	Face, Eye, Gaze	Si	Si	No	No	Free y Cód. abierto
<b>WebGazer.js</b>	+++	Eye, Gaze	No	Si	No	No	Free

Figura 4.21. Tabla comparativa de las características de los software encontrados durante la fase 2 del proyecto.

## 4.10 Otro software encontrado

Este apartado está dedicado a hacer mención a proyectos destacados encontrados durante el proceso de búsqueda, basados en el uso de webcams convencionales con una pequeña modificación en el hardware para crear sistemas IR, además de proyectos para el análisis de data producida por eye-trackers comerciales o proyectos de código abierto

basados en luz IR. Entre los más destacables se encuentran ITU Gaze Tracker [46], un software basado en un sistema IR que ya aunque actualmente no se encuentra disponible y no ofrece soporte ha sido objeto de muchos estudios, y Ogama [47], un software de soporte compatible con múltiples eye-trackers y proyectos de código abierto.

Este tipo de software, al requerir una modificación del hardware de la propia webcam que requiere un mínimo de experiencia y precisión para realizarse, no se ha tenido en consideración en el proceso de selección, descartando la modificación de una cámara web como opción viable.

## 4.11 Selección final y problemas encontrados

Para la selección final de un software con el que trabajar se ha tenido en cuenta que cumplan las siguientes premisas en la mayor medida posible:

- Que el software sea gratuito
- Que el software sea de código abierto
- Que permita el Gaze Tracking
- Que sea posible obtener las coordenadas de la mirada estimadas para su posterior uso en la generación de un mapa de calor<sup>12</sup>.

En base a esas premisas, como candidatos en orden de importancia se consideraron los siguientes, descartando el resto:

- **Xlabs Chrome Extension**, siendo un servicio gratuito, fácil de instalar y usar, accesible, y se puede usar directamente sobre una página web sin problema.

---

<sup>12</sup> Solo en el caso de que el propio software no proporcione esa funcionalidad.

- **GazeRecorder**, priorizando la versión para Escritorio sobre la online, ambas opciones siendo bastante completas excepto por la limitación de grabación de 1 minuto.
- **Opengazer / CVC Eye-tracker**, siendo la opción de código abierto más completa y con la posibilidad de modificarla si se considera necesario.
- **Matlab Software**, como cuarta opción, debido a que requiere una implementación completa del sistema de gaze tracking.
- **WebGazer.js**, siendo la última opción a considerar ya que requiere modificar la página web donde quiere realizarse el proceso de Gaze Tracking.

Con la primera opción, Xlabs Chrome Extension, se estudió la posibilidad de obtener los *gaze points* detectados. La opción quedó descartada ya que no era posible modificar la extensión debido a la licencia comercial a la que está ligada la API que gestiona el software.

Considerando la segunda opción, GazeRecorder, no era posible remover la limitación de 1 minuto de la versión de escritorio. Con permiso del dueño del software, conseguimos una versión online sin limitaciones, que tuvimos que descartar debido a incompatibilidades con la página web correspondiente a la aplicación CSCL y las limitaciones a la hora de obtener los datos de los estudios realizados.

Por lo tanto, la elección final corresponde a la tercera opción, **CVC Eye-tracker**, siendo la única opción que permite modificar el código para obtener los *gaze points* estimados y la obtención de los datos referentes al *gaze tracking*.



## 5. MODIFICACIONES FINALES Y FASE DE TESTEO

En este capítulo se expondrán las modificaciones realizadas al software CVC Eye-tracker, con el fin de obtener las coordenadas de la mirada para su posterior procesado, y las pruebas realizadas para demostrar que la modificación es funcional. El sistema operativo utilizado para realizar las modificaciones y el testeo es Linux Ubuntu 12.04 32 bits,

### 5.1 Estructura del código de CVC Eye-tracker

El código de CVC Eye-tracker, de igual forma que OpenGazer, está estructurado en base a módulos esenciales (Ventana de debug, calibrador, etc.) y módulos adicionales (EyeExtractor, GazeTracker, etc.) [48].

El archivo '**MainGazeTracker.cpp**' contiene bucle principal o *main loop* que controla la aplicación. Todos los módulos son creados en su constructor y ejecutados (mediante los métodos process() y draw()) en cada iteración del bucle principal. El software divide los módulos en 3 categorías:

- **Pre-módulos**, correspondientes a los módulos que hay que ejecutar con anterioridad a los módulos correspondientes al *gaze tracking* (selección de feature points, etc.).
- **Módulos Gaze**, se encargan de calcular y estimar la mirada.
- **Post-módulos**, que se ejecutan una vez está activa la estimación de la mirada, corresponden a juegos o aplicaciones que usan la mirada detectada una vez calibrado el sistema, como el correspondiente a '**FrogGame.cpp**'.

Según la categoría, los módulos heredan tanto la clase Component como la clase GazeTrackerComponent, que contienen métodos necesarios para su funcionamiento, como por ejemplo el método process().

## 5.2 Posibles modificaciones planteadas

La primera opción considerada era crear un nuevo módulo correspondiente a la categoría de Post-modulos. Considerando el tiempo disponible y el grado de experiencia y entendimiento del código se descartó esta opción.

La segunda opción considerada era modificar el post-módulo existente en el software, FrogGame. Este módulo en concreto debería desactivarse posteriormente para el uso único del software como gaze-tracker, por lo que era viable modificarlo para usarlo únicamente las coordenadas de la mirada estimadas después del proceso de calibración.

La modificación propuesta es la siguiente:

- En el archivo '**FrogGame.cpp**':
  - o Se añade el siguiente fragmento de código para ocultar la ventana correspondiente al videojuego:

```
63     if(Application::status == Application::STATUS_CALIBRATED) {  
64         //window.show();  
65         _window.hide(); //ocultamos ventana correspondiente al videojuego  
66     }  
67     . . .
```

Figura 5.1. Código añadido para ocultar la ventana del videojuego en el archivo '*FrogGame.cpp*'.

- o Se añade el código correspondiente a guardar en variables las coordenadas de *gaze*, printarlas por el terminal y simultáneamente guardarlas en un archivo llamado *test\_gaze\_coordinates.txt*, previamente inicializado en el *main loop*:

```
76     int lastEstimationX = mappedEstimation.x;  
77     int lastEstimationY = mappedEstimation.y;  
78     //const int width = backgroundWithFrog.size().width;  
79     //const int height = backgroundWithFrog.size().height;  
80     double alpha = 0.6;  
81     estimationXRunningAverage = (1 - alpha) * lastEstimationX + alpha * estimationXRunningAverage;  
82     estimationYRunningAverage = (1 - alpha) * lastEstimationY + alpha * estimationYRunningAverage;  
83     . . .  
84     //Saving gaze coordinates to txt file  
85     int xgaze = (int)estimationXRunningAverage;  
86     int ygaze = (int)estimationYRunningAverage;  
87     std::cout << "Point x: " << xgaze << " " << "Point y: " << ygaze << "\n" << std::endl; //coordinates printed on the terminal  
88     std::ofstream myfile ("test_gaze_coordinates.txt", std::ios::in | std::ios::out | std::ios::ate); //coord saved on .txt  
89     myfile.is_open());  
90     {  
91         myfile << xgaze << "," << ygaze << "\n";  
92     }  
93     myfile.close();  
94 }  
95 else std::cout << "Unable to open txt file";  
96 //FIN DEL CODIGO EDITADO  
97  
98  
99  
100  
101  
102
```

Figura 5.2. Fragmento de código correspondiente al guardado de coordenadas y print por Terminal en el archivo '*FrogGame.cpp*'.

- Se comenta el código destinado al videojuego, tal y como se muestra en las siguientes figuras (5.3, 5.4, 5.5):

```
Esc Borrado/OpenGazer/master/FrogGame.cpp - Sublime Text (UNREGISTERED)



```

1 // FrogGame.cpp
2
3 #include "FrogGame.h"
4 #include "utilis.h"
5
6 FrogGame::FrogGame();
7
8     window1, false
9 {
10
11     // Read the source images for background (leftview) and Frog
12     clutter = cv::imread("../background_full.jpg", CV_LOAD_IMAGE_COLOR);
13     cvtColor(clutter, clutter, CV_RGB2BGR);
14
15     // Frog = cv::imread("../frog.png", CV_LOAD_IMAGE_COLOR);
16     cvtColor(frog, frog, CV_RGB2BGR);
17
18     // Read the mask images and convert the gaussian mask to float type
19     // FrogMask = cv::imread("../frog-mask.png", CV_LOAD_IMAGE_COLOR);
20     // BackgroundMask = cv::imread("../background-mask.png", CV_LOAD_IMAGE_COLOR);
21     // gaussianMaskConversionMask = CV_32FC3;
22     gaussianMask = gaussianMask(55, 6);
23
24     // Create the negative of the gaussian mask create a patch of solid background (to make operations faster)
25     // getBackgroundCreate(cv::Size(gaussianMask.size().width), gaussianMask.size().height, (CV_32FC3));
26     // gaussianMaskNegative = _solidBackground - gaussianMask;
27
28     // _solidBackground.setTo(cv::Scalar(155, 75, 75));
29
30     // BackgroundWithFrog = cv::Size(_window.size().width), _window.size().height(), CV_RGB);
31     // cvZero(BackgroundWithFrog);
32     // cvRect(BackgroundWithFrog, cv::Rect(_window.size().width / 2, _window.size().height / 2, (CV_RK));
33
34     // std::cout << "IMAGES CREATED WITH SIZE: " << _backgroundWithFrog.size().width << "x" << _backgroundWithFrog.size().height << std::endl;
35
36     // fill background with black
37     // _backgroundWithFrog.setTo(cv::Scalar(0,0,0));
38
39     // calculate the camera area
40     cameraX = (_window.size().width) - _clutter.size().width / 2;
41     cameraY = (_window.size().height) - _clutter.size().height / 2;
42     cameraWidth = _clutter.size().width;
43     cameraHeight = _clutter.size().height;
44
45
46     // Initialize timing variables
47     // Frequency = 0;
48     // calculateFuturePosition();
49
50
51     // Set the time limit
52     gettimeofday(&time, NULL);
53     futureTime = (time.tv_sec * 1000) + (time.tv_usec / 1000);
54     futureTime = futureTime + 1000 * 20000;
55
56     startime = _futureTime;
57
58     FrogGame::~FrogGame() { }
59 }
```


```

Figura 5.3. Lineas de código comentadas en FrogGame.cpp.

```
#include "FrogGame.h"
#include "OpenGaze.h"
#include "GaussianBlur.h"

FrogGame::FrogGame()
{
    _startTime = _futureTime;
}

void FrogGame::process()
{
    double distanceEstimationRunningAverage = 0;
    double estimationRunningAverage = 0;

    if(Application::status == Application::STATUS_CALIBRATED) {
        // window.show();
        _window.hide(); // Ocultamos la ventana correspondiente al videojuego
    }
    else {
        _window.hide();
        return;
    }

    Point mappedEstimation;
    if(GazePoint::implIsMonitorCoordinates(Application::Data::gazePoints[0], mappedEstimation)) {
        int lastEstimationX = mappedEstimation.x;
        int lastEstimationY = mappedEstimation.y;

        //const int width = backgroundWidth(FrogSize().width);
        //const int height = backgroundWidth(FrogSize().height);

        double alpha = 0.8;
        estimationRunningAverage = (1 - alpha) * lastEstimationX + alpha * estimationRunningAverage;
        estimationRunningAverage = (1 - alpha) * lastEstimationY + alpha * estimationRunningAverage;

        // Saving gaze coordinates to file
        int xgaze = (int)estimationRunningAverage;
        int ygaze = (int)estimationRunningAverage;

        std::string myFile = "xgaze " + std::to_string(xgaze) + " " + std::to_string(ygaze) + ".txt";
        std::ofstream myfile ("test_gaze_coordinates.txt");
        std::ios::in sid; std::ios::out | std::ios::ate; // record saved on .txt
        if(myfile.is_open())
        {
            myfile << xgaze << "\n";
            myfile.close();
        }
        else std::cout << "Unable to open txt file";
    }
}

// Determining the bounds to copy the corresponding area (under the gaze point) to the screen
Rect bounds = cv::Rect(estimationRunningAverage - gaussianMask.size().width/2,
                      estimationRunningAverage - gaussianMask.size().height/2,
                      gaussianMask.size().width,
                      gaussianMask.size().height);
cv::Rect gaussianBounds = cv::Rect(0, 0, gaussianMask.size().width, gaussianMask.size().height);

if(bounds.x < 0) { // Remove the amount from the width
    bounds.width += bounds.x;
}
```

Figura 5.4. Lineas de código comentadas en FrogGame.cpp.

```

160 cvt::Mat visibleBackground(cv::Size(gaussianBounds.width, gaussianBounds.height), CV_32FC3);
161 background3dBackground.convertTo(visibleBackground, CV_32FC3);
162 cvt::Mat copiedArea1 = cvt::Mat(gaussianBounds.width, gaussianBounds.height, CV_32FC3);
163 cvt::Mat copiedArea2 = cvt::Mat(gaussianBounds.width, gaussianBounds.height, CV_32FC3);
164 copiedArea1.convertTo(copiedArea1, CV_32FC3);
165 copiedArea2.convertTo(copiedArea2, CV_32FC3);
166
167 // Apply the gaussian mask so that the visible area has a nice gradient around it
168 copiedArea1 = copiedArea1 * gaussianMask(gaussianBounds) + outsideBackground(gaussianBounds).mul(gaussianMaskNegative(gaussianBounds));
169 copiedArea2.convertTo(copiedArea2, CV_32FC3);
170
171 // Copy the shown area to the screen image
172 copiedArea2.copyTo(screenImage);
173
174 // Draw on the screen
175 window.showImage(screenImage);
176
177 // Timing stuff
178 int diff = (estimationXRunningAverage - _frogX) * (estimationYRunningAverage - _frogY) + ((estimationXRunningAverage - _frogX) * (estimationYRunningAverage - _frogY));
179 if (diff > 3500) { // If more than 100 pix. count
180     _tempTime = time - _startTime;
181     _tempTime -= _tempTime % DIFFERENCE_IN_TIME; // End;
182     _tempTime += DIFFERENCE_IN_TIME; // Start;
183     _tempTime = (time.tv_sec * 1000) + (time.tv_usec / 1000);
184 }
185 if (_startime == _futureTime) {
186     _startime = _tempTime;
187 } else if (_tempTime - _startime > 2500) { // If Fixes on the same point for 2.5 seconds
188     _startime = _tempTime;
189     _startime = _futureTime;
190 } else { // If cannot focus for a while, reset
191     _startime = _futureTime;
192 }
193
194 void FrogGame::draw() {}
195
196 void FrogGame::calculateFrogPosition() {
197     cvt::Rect gameAreaRect(0, 0, _gameAreaWidth, _gameAreaHeight);
198     cvt::Rect frogRect((rand() % _gameAreaWidth - _frog.size().width/2) + _gameAreaRect.x, _gameAreaRect.y, _frog.size().width/2, _frog.size().height/2);
199     cvt::Rect backgroundRect((rand() % _gameAreaWidth - _frog.size().height/2) + _gameAreaRect.x, _gameAreaRect.y, _frog.size().width/2, _frog.size().height/2);
200
201     // Copy the initial background image to "background"
202     clutter.copyTo(backgroundRect);
203     // Copy the frog image to the background
204     _frog.copyTo(backgroundRect);
205     // Copy the frog image to the background
206     _frog.copyTo(backgroundRect);
207     _frog.copyTo(backgroundRect);
208     _frog.copyTo(backgroundRect);
209     _frog.copyTo(backgroundRect);
210     _frog.copyTo(backgroundRect);
211 }
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245

```

Figura 5.5. Lineas de código comentadas en FrogGame.cpp, desde la linea 160 correspondiente al mensaje ‘*//Fin del código editado.*’ Hasta la linea 185 con el mensaje ‘*//Timing stuff*’ el código se encuentra comentado.

- En el archivo ‘**MainGazeTracker.cpp**’ se añade el siguiente fragmento de código para inicializar el archivo de texto donde se guardaran las coordenadas:

```

16 #include "Application.h"
17 #include "utils.h"
18 #include "FaceDetector.h"
19 #include "AnchorPointSelector.h"
20 #include "DebugWindow.h"
21 #include "TestWindow.h"
22 #include "HiResTimer.h"
23
24 MainGazeTracker::MainGazeTracker(int argc, char **argv)
25 {
26     _stores(Application::getStores());
27     _commandIndex(-1),
28     args(argc, argv)
29 {
30     //RESET TXT FILE WHERE THE GAZE POINTS ARE STORED
31     std::cout << "Text file reseted" << "\n" << std::endl;
32     std::ofstream myfile ("test_gaze_coordinates.txt");
33     if (!myfile.is_open())
34     {
35         myfile << "";
36         myfile.close();
37     }
38     else std::cout << "Unable to open txt file";
39     //END OF THE ADDED CODE
40     |
41     Application::Components::mainTracker = this;
42
43     if (args.getOptionValue("help").length())
44         std::cout << std::endl;
45

```

Figura 5.6. Código añadido correspondiente al archivo ‘*MainGazeTracker.cpp*’.

Mediante estos cambios, cada vez que se inicie el software se creara un archivo de texto llamado ‘**test\_gaze\_coordinates.txt**’ en la carpeta del software si este no existe. En caso de existir el archivo porque se ha creado en sesiones anteriores, al iniciarse el software

el contenido de dicho archivo se borrará y reiniciará para grabar los PoG de la sesión actual. Los PoG solo son grabados en el archivo de texto y printados por el terminal una vez terminada la calibración.

Este diseño permite que los usuarios no tengan que preocuparse por sustituir el archivo de texto entre sesión y sesión o en caso de error, pudiendo este extraerse sin problema para su posterior procesado sin perder información. Inicializando el archivo de texto al inicio del *main loop* antes de los *calls* a la función *proces()* que “activa” el resto de módulos se evita que el archivo de texto se reinicialice repetidamente con cada iteración del gaze-tracker, asegurando que todos los PoG detectados después de la calibración serán registrados.

### 5.3 Testeo final

Para el testeo de las correspondientes modificaciones, se ha usado una cámara Sweek ViewPlus Webcam USB [49]. Se ha realizado un testeo de duración de 5 minutos posterior a la calibración del sistema. Las condiciones de luz ambiente no eran perfectas, como puede observarse en la figura 5.7:



Figura 5.7. CVC Eye-Tracker registrando los PoG una vez calibrado el sistema.

Como puede observarse, el sistema detecta y mapea correctamente la mirada, printándolas coordenadas por el terminal. Así mismo, se genera el archivo de texto con los PoG correspondientes a la sesión, como puede observarse en la figura 5.8:

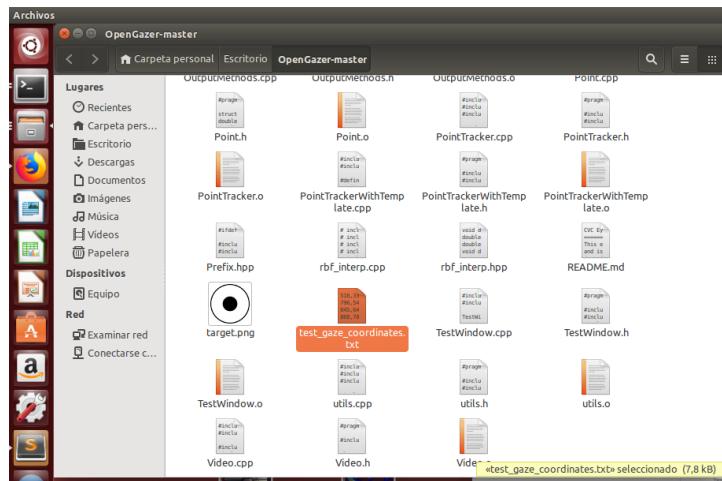


Figura 5.8. Archivo .txt generado una vez se inicializa el software.

Mediante los *gaze points* registrados en el archivo de texto, se puede generar un mapa de calor mediante herramientas/proyectos que reciben como input coordenadas de *gaze*. En la figura 5.9 se observa un mapa de calor generado con el siguiente proyecto: [50]<sup>13</sup>.

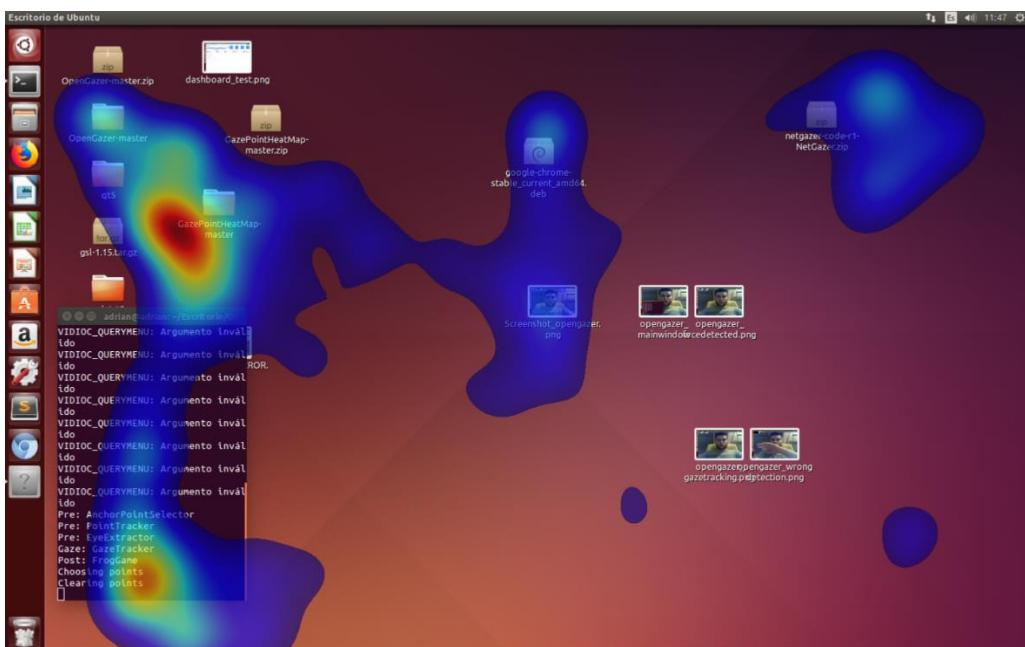


Figura 5.9. Mapa de calor generado mediante las coordenadas registradas de CVC Eye-tracker.

<sup>13</sup> Script basado en el generador de mapas de calor ofrecido por la librería PyGaze.

Como puede verse, aun no siendo un sistema tan eficaz como el que puede proporcionar un eye-tracker comercial, ofrece resultados aceptables para determinar las zonas que han captado la atención del usuario.

## 5.4 Planteamiento de la prueba a realizar con la Dashboard

Como puede observarse, se ha conseguido un sistema basado en *webcam eye tracking* que es capaz de mapear la mirada de los usuarios en la pantalla, con una fácil extracción de los puntos registrados por el eye-tracker para su posterior evaluación. Cabe destacar que aún existe una limitación a tener en cuenta:

- Cualquier ventana no correspondiente al software sobrepuerta a la ventana de control del eye-tracker causa que la detección de la mirada pare y se pierda la calibración, así como obstáculo entre la cámara y el usuario (figura 4.18, página 41).

El paso final del proyecto es plantear una solución viable que pueda ser utilizada para complementar la *dashboard*, y así determinar qué secciones son de más interés para los profesores. Para ello, se han planteado la siguiente metodología para habilitar el testeo de CVC Eye-tracker durante el uso de la dashboard:

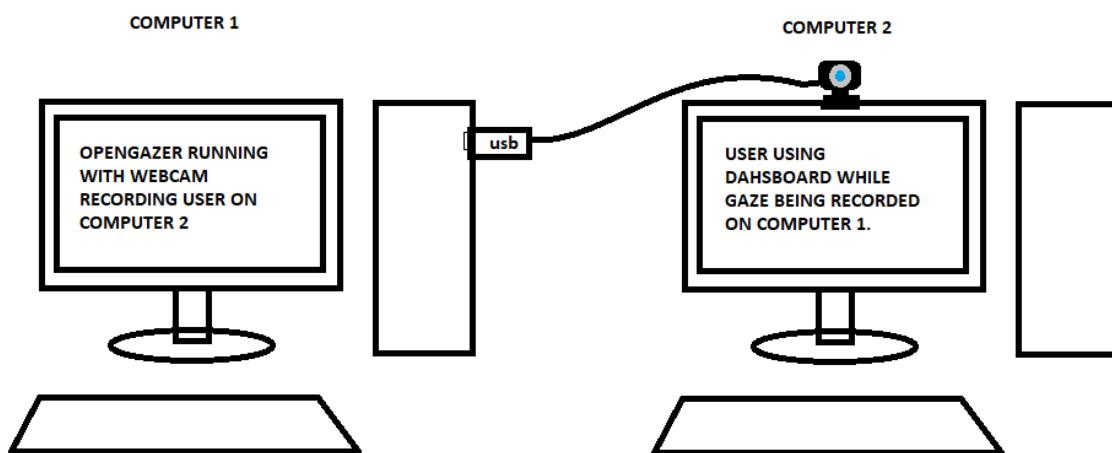


Figura 5.10. Esquema del test planteado para realizar pruebas con la Dashboard usando CVC Eye-tracker.

Para la realización de la prueba de forma óptima los siguientes elementos son necesarios:

- Dos ordenadores (computer 1 y computer 2 en la figura 5.10.) con pantallas del mismo tamaño y resolución. Cualquier diferencia de tamaño o resolución puede reducir la eficiencia de la prueba. El sistema operativo debe ser Windows 10, y ambos ordenadores deben disponer de una máquina virtual de Ubuntu Linux 12.04.
- Un software para grabar la pantalla del participante en todo momento [51].
- Un software para que Computer 1 pueda compartir pantalla con Computer 2 [52].
- Una webcam convencional con una resolución mínima de 640 x 480, colocada de forma centrada en Computer 2, que será el ordenador que manipularán los participantes.

El procedimiento para realizar la prueba es el siguiente:

- 1) Ambos, Computer 1 y Computer 2, deben iniciar la máquina virtual en pantalla completa, y el software para compartir pantalla. Una vez la máquina virtual se haya inicializado, Computer 1 compartirá su pantalla a Computer 2.
- 2) Computer 2 abrirá la *dashboard* correspondiente en el navegador Chromium de la máquina virtual. Una vez abierta y preparada para iniciar el test, minimizará la máquina virtual y abrirá la pantalla compartida por Computer 1 en pantalla completa.
- 3) Computer 1 iniciará CVC Eye-tracker. El participante de computer 2 deberá mirar fijamente su pantalla y sin realizar movimientos bruscos, por donde computer 1 estará compartiendo la máquina virtual en pantalla completa donde se está ejecutando el eye-tracker. Computer 1 inicializará el proceso de detección de *feature points* y calibración, mediante el cual el participante desde Computer 2 calibrará el sistema con su mirada.

- 4) Una vez calibrado el sistema, el participante en computer 2 minimizará la ventana de la pantalla compartida, y abrirá su máquina virtual con la *dashboard* inicializada. Empezará a realizar los tests pertinentes mientras la persona en el computer 1 gestiona la prueba, chequeando en todo momento que la calibración se mantiene y no hay ningún problema.
- 5) Una vez finalizado el test, Computer 1 cerrará CVC Eye-tracker y extraerá de la carpeta del programa el archivo de texto con las coordenadas registradas del participante. Llegados a este punto finaliza el test, analizando posteriormente las coordenadas para generar un mapa de calor.

De esta forma, se evita la limitación actual del eye-tracker en cuestión, además de proporcionar una metodología donde el participante no debe preocuparse de gestionar y controlar el software de detección de mirada.

Debido a la situación causada por el COVID-19, la falta de recursos e instalaciones y la imposibilidad de cumplir los requisitos adecuados para la realización del test, no se ha podido demostrar el funcionamiento de la metodología planteada. Aun así, tomando en referencia proyectos que han usado CVC Eye-tracker en configuraciones similares [16] y los requisitos ya cumplidos, podemos considerar el software una potencial solución viable para el objetivo inicial de este trabajo.



## **6. CONCLUSIONES**

En este capítulo se expondrán las conclusiones obtenidas tras el proceso de investigación realizado, y las posteriores pruebas realizadas con la solución planteada. Se realizará una reflexión global analizando resultados y limitaciones, y posteriormente se plantearán mejoras para la extensión del proyecto.

### **6.1 Reflexión**

Si bien es cierto que la tecnología referente a *webcam eye tracking* basada en luz ambiente es limitada y ofrece resultados que no igualan el rendimiento de los eye-tracker comerciales actuales, ha quedado demostrado que aunque dichos dispositivos no puedan igualar a sus competidores en eficiencia y precisión, los resultados son suficientemente aceptables como para realizar análisis relevantes.

A nivel de proyectos y software, existe una gran variedad de implementaciones de distintas fuentes dedicadas a este campo. Aun así, las opciones de código abierto siguen siendo muy limitadas. Aunque la gran mayoría de implementaciones han dejado de mantenerse o quedan obsoletas en frente a competidores comerciales y sistemas basados en luz IR que cada vez son más asequibles, se siguen desarrollando mejoras y nuevos proyectos que consiguen niveles de eficiencia cada vez más cercanos a un dispositivo especializado en el *gaze tracking*.

Teniendo en mente lo ya mencionado, y como conclusión personal, he aprendido que con la tecnología actual el uso de proyectos de código abierto especializados en el campo de *webcam eye/gaze tracking* es viable para la realización de estudios de monitorización de la mirada y posterior estudio de la atención de los usuarios, aunque es necesaria una actualización de dichos proyectos, ya que la gran mayoría se encuentran obsoletos o dejaron de recibir soporte de los desarrolladores originales una vez realizados los estudios para los que fueron desarrollados.

En cuanto al objetivo principal del proyecto, aunque no se han podido realizar todas las pruebas planificadas en un inicio por las limitaciones y problemas surgidos a lo largo de

su desarrollo, se ha encontrado una opción de código abierto que cumple con la mayoría de los requisitos buscados. Además, el software proporciona una estructura robusta, con margen de mejora y desarrollo que puede dar lugar a implementaciones más elaboradas, que se adecuen y se especialicen a objetivos más específicos.

Cabe destacar que debido a la situación provocada por el COVID-19, se ha adaptado en la medida de lo posible la realización del proyecto con los recursos disponibles. Las reuniones se han realizado mediante videoconferencia, y se ha usado el correo electrónico como otro medio principal de comunicación, tanto con las directoras del proyecto como para la comunicación con los desarrolladores de los software estudiados. Aunque estaba planificado el testeo del software seleccionado incluyendo la *dashboard*, considerando opciones para realizarlo online, éste no ha sido posible con los recursos disponibles, por lo que se ha optado por presentar el plan considerado y proporcionar sugerencias y mejoras para futuros proyectos.

## 6.2 Sugerencias para posibles mejoras

El software escogido como candidato final, CVC Eye-tracker, mantiene la estructura modular de su predecesor, fácilmente manipulable para añadir nuevas funcionalidades.

La primera sugerencia de posible mejora sería el desarrollo de un módulo dentro de la categoría de post-módulos especializado para trabajar de forma específica con la *dashboard*. De esta forma, se podrían establecer ciertas condiciones para agilizar el proceso de testeo y facilitar la participación de los usuarios y las personas que gestionen las pruebas.

Como segunda sugerencia para mejorar el software, propondría implementar un módulo dedicado a analizar los datos recogidos para proporcionar automáticamente métricas como un mapa de calor sin necesidad de usar software externo.

Hablando en términos generales propondría como mejora, si es posible, obtener un software comercial especializado, pues ofrecen resultados más exactos, y son sistemas más robustos que permiten más libertad de movimiento y comodidad a los participantes en comparación que los basados en webcams convencionales.

## Bibliografia

- [0] “Aprendizaje colaborativo: PyramidApp” [Online]. Available: <https://www.upf.edu/es/web/tide/collaborative-learning-pyramidapp> . [Accessed: 28-June-2020]
- [1] “What is eye tracking?” [Online]. Available: <https://www.tobiipro.com/blog/what-is-eye-tracking/> . [Accessed: 5-June-2020]
- [2] O. Ferhat, F.Vilariño, “Eye-Tracking with Webcam-Based Setups: Implementation of a Real-Time System and an Analysis of Factors Affecting Performance,” Universitat Autònoma de Barcelona, 2012.
- [3] M.Eizenman, D.Model, E.Guestrin, “Covert monitoring of the point-of-gaze,” University of Toronto, 2009.
- [4] J.Chen, Q.Jim, “3D Gaze Estimation with a Single Camera without IR Illumination,” Rensselaer Polytechnic Institute, 2009.
- [5] S.Höffner, “Gaze Tracking Using Common Webcams,” Osnabrück University, 2018.
- [6] P.Kasprowski, K.Harezlak, M.Stasch, “Guidelines for eye tracker calibration using points of regard,” Silesian University of Technology, 2014.
- [7] V.Janthanasub, P.Meesad, “Evaluation of a Low-cost Eye Tracking System for Computer Input,” King Mongkut’s University of Technology North Bangkok, 2015.
- [8] “Tobii Pro Glasses 2” [Online]. Available: <https://www.tobiipro.com/es/products/tobii-pro-glasses-2/> . [Accessed: 5-June-2020]
- [9] B.Farnsworth, “Eye Tracking: The Complete Pocket Guide,” 2018. [Online]. Available: <https://imotions.com/blog/eye-tracking/> . [Accessed: 5-June-2020]
- [10] S.Shih, Y.Wu, J.Liu, “A calibration-free gaze tracking technique,” National Chi Nan University, 2000.
- [11] H.R.Chennamma, “A survey on eye-gaze tracking techniques,” Sri Jayachamarajendra College of Engineering, 2013.
- [12] I.F.Ince, J.W.Kim, “A 2D eye gaze estimation System with low-resolution webcam images,” Journal on Advances in Signal Processing, 2011.
- [13] B.Kunka, B.Kostek, “Non-intrusive infrared-free eye tracking method,” Gdansk University of Technology, 2009.

- [14] O.Ferhat, A.L.Carmona, F.Vilariño, “A Feature-Based Gaze Estimation Algorithm for Natural Light Scenarios,” Universitat Autònoma de Barcelona, 2015.
- [15] J.Lemley, A.Kar, A.Drimbarean, P.Corcoran, “Efficient CNN Implementation for Eye-Gaze Estimation on Low-Power/Low-Quality Consumer Imaging Systems,” IEEE, 2018.
- [16] O.Ferhat, F.Vilariño, “A Cheap Portable Eye-Tracker Solution for Common Setups,” Universitat Autònoma de Barcelona, 2013.
- [17] “Eye tracker accuracy and precision” [Online]. Available: <https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-affects-the-accuracy-and-precision-of-an-eye-tracker/>. [Accessed: 6-June-2020]
- [18] “Accuracy and precision test method for remote eye trackers” [Online]. Available: [https://f.hypotheses.org/wp-content/blogs.dir/2484/files/2015/03/Tobii\\_Test\\_Specifications\\_Accuracy\\_and\\_PrecisionTestMethod\\_version-2\\_1\\_1.pdf](https://f.hypotheses.org/wp-content/blogs.dir/2484/files/2015/03/Tobii_Test_Specifications_Accuracy_and_PrecisionTestMethod_version-2_1_1.pdf). [Accessed: 6-June-2020]
- [19] A.Bojko, “The Most Precise (or Most Accurate?) Eye Tracker,” 2011. [Online]. Available: <https://www.gfk.com/blog/2011/05/the-most-precise-or-most-accurate-eye-tracker>. [Accessed: 6-June-2020]
- [20] A.M.Feit, S.Williams, A.Toledo, A.Paradiso, H.Kulkarni, S.Kane, M.R.Morris, “Toward Everyday Gaze Input: Accuracy and Precision of Eye Tracking and Implications for Design,” University of Colorado, 2017.
- [21] “Let’s talk accuracy and precision” [Online]. Available: <https://help.tobii.com/hc/en-us/articles/213534825-Let-s-talk-accuracy-and-precision>. [Accessed: 6-June-2020]
- [22] “Working with Heat Maps and Gaze Plots” [Online], Available: <https://www.tobiipro.com/learn-and-support/learn/steps-in-an-eye-tracking-study/interpret/working-with-heat-maps-and-gaze-plots/>. [Accessed: 6-June-2020]
- [23] O.B.Jensen, “Webcam-Based Eye Tracking vs. an Eye Tracker [Pros & Cons]” [Online]. Available: <https://imotions.com/blog/webcam-eye-tracking-vs-an-eye-tracker/>. [Accessed: 6-June-2020]
- [24] R.Mantiuk, M.Kowalik, A.Nowosielski, B.Bazyluk, “Do-It-Yourself Eye Tracker: Low-Cost Pupil-Based Eye Tracker for Computer Graphics Applications,” West Pomeranian University of Technology in Szczecin, 2012.
- [25] Y.Zhang, A.Bulling, H.Gellersen, “Towards pervasive eye tracking using low-level image features,” Lancaster University, 2012.

- [26] B.Kunka, B.Kostek, M.Kulesza, “Gaze-tracking-based audio-visual correlation analysis employing quality of experience methodology,” Gdansk University of Technology, 2010.
- [27] “GazeRecorder” [Online]. Available: <https://gazerecorder.com/.> [Accessed: 6-June-2020]
- [28] “Online Eye-Tracking made easy” [Online]. Available: <https://app.gazerecorder.com/.> [Accessed: 6-June-2020]
- [29] S.Deja, “The comparison of accuracy and precision of eye tracking: GazeFlow vs. SMI RED 250.,” User Experience Lab Kraków, 2013.
- [30] “Sentigaze SDK” [Online]. Available: <https://www.neurotechnology.com/sentigaze.html.> [Accessed: 10-June-2020]
- [31] “GazeAlyze” [Online]. Available: <http://gazealyze.sourceforge.net/.> [Accessed: 10-June-2020]
- [32] P.Viola, M.Jones, “Robust Real-time Object Detection,” Vancouver, Canada, 2001.
- [33] H.T.Ngo, R.Rakvic, R.P.Broussard, R.W.Ives, “An FPGA-based design of a modular approach for integral images in a real-time face detection System,” U. S. Naval Academy, 2009.
- [34] I.Munguía, "Simple Gaze Tracker application built with Matlab" 2018. [Online]. Accesible: <https://github.com/warborn/gaze-tracker.> [Accessed: 10-June-2020]
- [35] L.T.Phuc, “Simple eye gaze using Matlab” 2015. [Online]. Accesible: <https://www.youtube.com/watch?v=d-EpJehXnG8.> [Accessed: 10-June-2020]
- [36] “Chrome Browser Extension” [Online]. Available: <https://www.xlabsgaze.com/browser-extension.> [Accessed: 10-June-2020]
- [37] P. Zielinski, “Opengazer: open-source gaze tracker for ordinary webcams (software)” 2012. [Online]. Available: <http://www.inference.phy.cam.ac.uk/opengazer/.> [Accessed: 10-June-2020]
- [38] “opengazer: open-source gaze tracking” [Online]. Available: <https://youtu.be/bySO5-rXoBw.> [Accessed: 10-June-2020]
- [39] M.Onkar, “Multimodal interface integrating eye gaze tracking and speech recognition,” The University of Toledo, 2015.
- [40] “Machine Vision Group: Low Cost Eye-Tracker” [Online]. Available: <http://mv.cvc.uab.es/projects/eye-tracker.> [Accessed: 11-June-2020]
- [41] “CONTRIBUTORS.md” [Online]. Available: <https://github.com/tiandan/OpenGazer/blob/master/CONTRIBUTORS.md.> [Accessed: 11-June-2020]

- [42] “Pyramidapp” [Online]. Available: [https://www.upf.edu/es/web/tide/tools-/asset\\_publisher/W2iQtvwlOQI/content/id/183108101/maximized](https://www.upf.edu/es/web/tide/tools-/asset_publisher/W2iQtvwlOQI/content/id/183108101/maximized). [Accessed: 11-June-2020]
- [43] “WebGazer.js” [Online]. Available: <https://webgazer.cs.brown.edu/#contact>. [Accessed: 11-June-2020]
- [44] A.Papoutsaki, P.Sangkloy, J.Laskey, N.Daskalova, J.Huang, J.Hays, “WebGazer: Scalable Webcam Eye Tracking Using User Interactions,” Brown University, 2016.
- [45] A.Papoutsaki, A.Gokaslan, J.Tompkin, Y.He, J.Huang, “Proceedings of the 2018 ACM Symposium on Eye Tracking Research \& Applications,” Brown University, 2018.
- [46] “ITU Gaze Tracker” [Online]. Available: <https://www.eyecomtec.com/3104-itu-gaze-tracker>. [Accessed: 15-June-2020]
- [47] “Ogama” [Online]. Available: <http://www.ogama.net/>. [Accessed: 15-June-2020]
- [48] “DEVELOP.md” [Online]. Available: <https://github.com/tiendan/OpenGazer/blob/master/DEVELOP.md>. [Accessed: 15-June-2020]
- [49] “Webcam USB 2 MPixel 720p Plastic Black” [Online]. Available: <http://www.sweex.com/es/ordenadores/dispositivos-de-entrada/webcam-usb-2-mpixel-720p-plastic-black-550512340>. [Accessed: 15-June-2020]
- [50] T.Roeddiger, “GazePointHeatMap” 2018. [Online]. Available: <https://github.com/TobiasRoeddiger/GazePointHeatMap>. [Accessed: 15-June-2020]
- [51] “Free Cam” [Online]. Available: <https://www.freescreenrecording.com/>. [Accessed: 15-June-2020]
- [52] “Videollamadas y pantalla compartida” [Online]. Available: <https://support.discord.com/hc/es/articles/115000982752-Videollamadas-y-pantalla-compartida>. [Accessed: 15-June-2020]