Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC

María Esther Pineda Izquierdo
Asesorado por el Ing. Iván René Morales Argueta

Guatemala, septiembre de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

MARÍA ESTHER PINEDA IZQUIERDO
ASESORADO POR EL ING. IVÁN RENÉ MORALES ARGUETA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERA EN ELECTRÓNICA

GUATEMALA, SEPTIEMBRE DE 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgen Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Carlos Eduardo Guzmán Salazar
EXAMINADORA	Inga. María Magdalena Puente Romero
EXAMINADOR	Ing. Guillermo Antonio Puente Romero
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC

Tema que me fue asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 9 de agosto de 2016.

María Esther Pineda Izquierdo

Guatemala, Julio 19 de 2017

Ingeniero
Julio César Solares Peñate
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Estimado Ingeniero Solares:

Me permito dar aprobación al trabajo de graduación titular: **" IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC",** de la señorita **MARIA ESTHER PINEDA IZQUIERDO**, quien se identifica con número de carné **200917200**, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo y, yo, como su asesor, nos hacemos responsables por el contenido y conclusiones del mismo.

Sin otro particular, me es grato suscribirme.

Atentamente:

Iván René Morales Argueta
Ingeniero Electrónico
Colegiado 12489

Ing. Iván Morales
Ingeniero Electrónica
Colegiada Activo 12489
Asesor

FACULTAD DE INGENIERIA
Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.
Ciudad Universitaria, zona 12
Guatemala, Centroamérica

Guatemala, 27 de julio de 2017

Señor Director
Ing. Otto Fernando Andrino González
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"IMPLEMENTACIÓN DE UN SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC"**, desarrollado por la estudiante **María Esther Pineda Izquierdo**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Julio César Solares Peñate
Coordinador de Electrónica

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 37. 2017.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación de la estudiante; MARÍA ESTHER PINEDA IZQUIERDO titulado: IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC, procede a la autorización del mismo.

Ing. Otto Fernando Andrino González

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 28 DE AGOSTO 2,017.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.450-2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **IMPLEMENTACIÓN DE SISTEMA BIOMÉTRICO PARA ANÁLISIS PSICOFÍSICOS RELACIONADOS CON PSICOSEMIÓTICA, FACULTAD DE INGENIERÍA, USAC,** presentado por la estudiante universitaria: **María Esther Pineda Izquierdo,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, septiembre de 2017

/c c

ACTO QUE DEDICO A:

Dios	Por haberme dado la inteligencia, perseverancia y ánimo para poder seguir adelante.
Mis padres	Por sus múltiples esfuerzos.
Mi hermano	Pablo Pineda, por su apoyo y ayuda durante mi carrera.
Mis amigos	Gabriel Cabrera, Juan de Dios Mérida y Haroldo López por haberme apoyado y haber convertido este viaje en uno inolvidable.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por haberme brindado los recursos necesarios para poder graduarme.
Mis maestros y amigos	Anaí Ortiz, Andy Rodríguez y Juan Carlos Argueta, por haberme apoyado y aconsejado. Gracias por ser mis maestros y amigos de vida.
Departamento de Matemática	Por haberme permitido laborar y crecer profesionalmente.
Mi asesor	Iván Morales, por su apoyo y ayuda en la realización del presente trabajo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperios
D	Derivada
mm	Milímetros
Ω	Ohm

GLOSARIO

GND	Valor de referencia del circuito, normalmente 0 voltios.
TkInter	Librería para creación de interfaz gráfica para Python.
Eye traking	Es el proceso de evaluar el movimiento del ojo con respecto al entorno.
Opencv	Librería utilizada para visión artificial. Contiene métodos para adquirir, procesar, analizar y comprender imágenes.
Psicofísica	Estudia la reacción del cuerpo humano ante un fenómeno físico.
Psicosemiótica	Estudia la interpretación del individuo ante diferentes signos.

RESUMEN

El sistema visual puede ser estudiado para recabar datos pertinentes a la reacción de los usuarios con estímulos físicos. Por medio del sistema biométrico se pretende grabar la percepción del usuario que lo porta.

El sistema biométrico permite al usuario portar dos cámaras, una dirigida hacia el ojo y otra dirigida hacia el entorno. Por medio de la visión por computadora se analizaron y procesaron las imágenes captadas por la cámara dirigida hacia el ojo; se recolectaron coordenadas que posteriormente se dibujaron encima de la segunda cámara.

El sistema es portable y cuenta con una interfaz que permite a un segundo usuario controlar y observar el desarrollo de la captura. Se presenta tanto el desarrollo de la estructura física como el código de programación utilizado para el procesamiento de imagen. El trabajo finaliza con un presupuesto que permite conocer el costo total del sistema biométrico.

OBJETIVOS

General

Implementar un sistema biométrico para análisis psicofísicos relacionados con psicosemiótica.

Específicos

1. Implementar un sistema de bajo costo capaz de seguir el movimiento del ojo para analizar el punto donde se fija la mirada.

2. Capturar datos representativos en el sistema que sirvan para la investigación en los sistemas visuales.

3. Almacenar y desplegar de forma amigable al usuario los datos capturados para que posteriormente sean analizados, según la aplicación requerida.

INTRODUCCIÓN

El entorno de desarrollo humano ha cambiado a lo largo de los años. El aumento de la tecnología y la publicidad han causado un efecto en el cuerpo humano, que ha tenido que enfrentar nuevos retos. La visión es uno de los órganos más afectados en este sentido.

Los estudios acerca de la visión se han hecho a lo largo de varios años; debido al avance de la tecnología se ha facilitado y ha sido de gran interés en diversas disciplinas. Esto ha producido que se implementen sistemas biométricos capaces de capturar y analizar datos de diferente índole. Entre los sistemas biométricos se puede mencionar aquellos que incluyen tecnologías como el *eye tracking*. Esta tecnología es capaz de registrar en dónde se fija la mirada del individuo, se permite estudiar qué es lo que observa e inclusive el tiempo de observación.

Dentro de la psicosemiótica esto da la posibilidad de registrar la interacción de una persona con un signo, lo cual permite desarrollar estudios relacionados con sistemas visuales. Distintos sectores muestran interés en estos estudios, como las empresas que buscan la satisfacción del usuario.

Asimismo, estas tecnologías pueden tener gran influencia en el estudio de la psicofísica, que relaciona la reacción del cuerpo humano ante un fenómeno físico con uno psicológico. Debido a esto se buscó desarrollar un sistema capaz de evaluar la percepción del usuario, con el fin de obtener datos pertinentes a estudios de sistemas visuales y psicológicos. Este sistema incluye la tecnología del *eye tracking* y es capaz de recabar datos de interés, que son analizados por medio de procesamiento de imágenes.

1. SISTEMA VISUAL HUMANO

1.1. Energía electromagnética radiante

La luz puede comportarse como una onda electromagnética o como una partícula. Según la teoría ondulatoria, la onda electromagnética, al propagarse, varía el campo eléctrico que al mismo tiempo produce una variación en el campo magnético. Si se considera el comportamiento como onda es posible describir diferentes componentes físicos como la longitud de onda o intensidad.

Es importante recordar que Isaac Newton fue uno de los primeros en demostrar que los objetos y superficies no poseen color, en su tratado *Opticks* describe claramente que los rayos no están coloreados.

Por lo tanto, el color es una experiencia psicológica producida por el sistema nervioso, tal y como se describe en *The rays are not coloured* (Los rayos no tienen color) “El color es en realidad una sensación producto de la persona; los colores no existen a menos que algún observador los perciba. E color no se produce ni siquiera en la sucesión de eventos que tiene lugar entre los receptores retinales y la corteza visual, sino sólo hasta que la conciencia del observador interpreta finalmente la información”[1].

Por lo tanto, aun cuando el componente físico principal es la longitud de onda, cabe mencionar que se han identificado 3 dimensiones principales: el matiz, la brillantez y la saturación.

[1] WRIGHT, William. *The rays are not coloured.* www.readcube.com/articles/10.1038/1981239a0. Consulta: 5 de febrero de 2017.

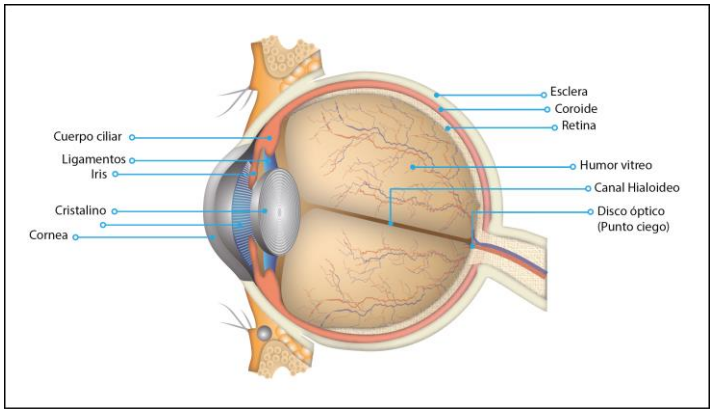
El color o más bien conocido como matiz, es la longitud de onda en sí y por medio de ella el cerebro puede hacer una interpretación de color. La brillantez es una dimensión que varía según la intensidad de la luz; mientras más alta es la intensidad, esta parece más blanca.

La saturación es la dimensión en donde a medida que se agreguen longitudes de onda a la longitud de onda original, esta reducirá la saturación y el color empezará a apreciarse más grisáceo.

1.2. Capas del globo ocular

En el ojo ocular se pueden distinguir 3 capas de tejido: una túnica externa donde se puede localizar la córnea, que es la que permite la entrada de los rayos de luz hacia el ojo; siguiente a este tejido se encuentra la esclera que es una capa externa que rodea y protege el ojo hasta el nervio óptico; una túnica media que se conoce más como úvea, que constituye el iris, y una túnica interna sensorial que forma la retina.

Figura 1. **Anatomía del ojo humano**



Fuente: *Designed by Freepik.* http://www.freepik.com. Consulta: 28 de junio de 2016.

1.2.1. Túnica externa

La túnica externa está constituida por la córnea y la esclera.

1.2.1.1. Córnea

La córnea es como una lente transparente que cubre al iris y al cristalino. Entre sus funciones se encuentran proteger al ojo del polvo, gérmenes o cualquier agente externo que pueda ser dañino al ojo; controlar el enfoque y la cantidad de luz que entra a él.

1.2.1.2. Esclera

La túnica externa continúa como esclera o esclerótica está constituida por un tejido duro, opaco y elástico. Ayuda a proteger y dar forma al ojo.

1.2.1.3. Túnica media

Es conocida como úvea y está conformada por: la coroides, el cuerpo ciliar y el iris.

1.2.1.4. Coroides

Es una membrana de color obscuro que se encuentra entre la retina y la esclerótica del ojo. Entre sus funciones principales se encuentran: nutrir la retina por medio de una gran cantidad de vasos sanguíneos; mantener una temperatura adecuada e impedir el paso de la luz por medio de su color obscuro.

1.2.1.5. Cuerpo ciliar

Está formado por un tejido grueso coroideo y se encuentra atrás del iris. Es responsable del humor acuso, el cual es un líquido claro que proporciona nutrientes al cristalino e iris.

1.2.1.6. Iris

Es una membrana de color que divide al ojo en dos compartimientos, en donde el compartimiento posterior conduce a la pupila. La pupila es la responsable de regular la cantidad de luz que entra al ojo. Detrás del iris se encuentra el cristalino.

1.2.2. Túnica interna

Más conocida como retina, es una capa que recubre las paredes internas del ojo. Su función es recoger, elaborar y transmitir sensaciones visuales.

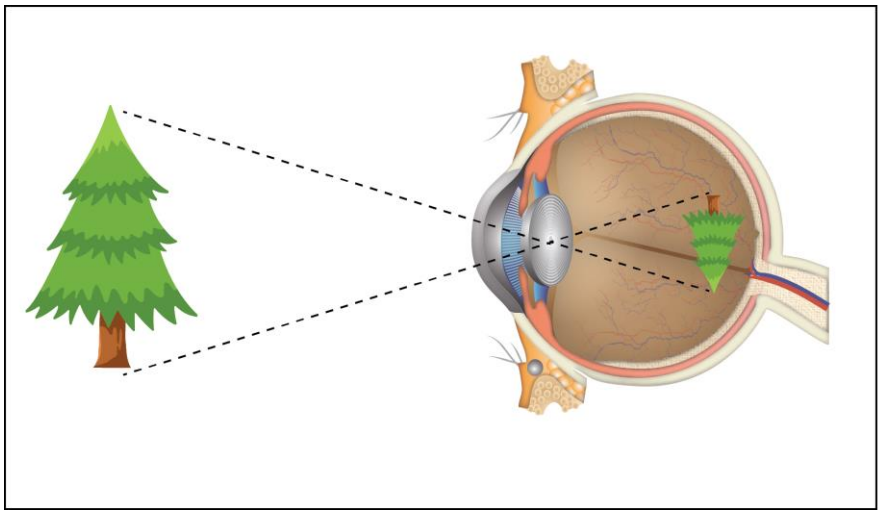
Utilizando instrumentos como el oftalmoscopio es posible observar la superficie interna de la retina a través de la pupila. La retina es la porción neural del ojo y es la parte del sistema nervioso central, está formada de neuronas interconectadas. Está compuesta por conos y bastones que son susceptibles a la luz.

1.3. Formación de imágenes en la retina

La córnea y el cristalino son los responsables principales de la refracción de la luz necesaria para la formación de imágenes sobre la retina.

La luz reflejada por un objeto debe atravesar la córnea y el cristalino para que pueda ser centrada en la retina. La córnea realiza el mayor enfoque de luz que entra, mientras el cristalino actúa como una fina lente que logra enfocar figuras a distintas distancias. Un factor importante a considerar es el índice de refracción del medio, porque debido a esto la imagen se puede ver borrosa y fuera de foco.

Figura 2. **Formación de imágenes en la retina**



Fuente: *Designed by Freepik.* http://www.freepik.com. Consulta: 28 de junio de 2016

1.4. Psicosemiótica y psicofísica

En términos generales la psicofísica es la parte de la psicología que estudia entre un físico y una repuesta psicológica. Mientras que la psicosemiótica estudia la relación entre un estímulo compuesto por un signo y una respuesta psicológica

1.4.1. Importancia de la percepción visual

El 80 % de la información que se capta es por medio de la vista. Por lo tanto, la vista es el sentido dominante; esto se puede comprobar al observar diferentes experimentos realizados, entre ellos el estudio de la estimulación visual[2].

Su estudio consistía en hacer que el usuario dirigiera su vista a través de un lente que distorsionaba los objetos, mientras palpaba con la mano un cuadro cubierto por un fino lienzo. Mientras esto, ocurría el usuario concordaba con que el cuadrado (objeto real) en realidad era un rectángulo (objeto percibido). Esto quiere decir que la imagen distorsionada influyó más en el usuario que la sensación táctil no alterada.

1.4.2. Semiótica

Todo proceso de comunicación está representado por signos que a su vez adquieren un significado. La semiótica se puede definir como el estudio del significado de estos signos.

1.4.3. Psicofísica

La psicofísica o psicología experimental es el estudio de la relación entre fenómenos físicos y mentales.

[2] *Teoría para el diseño visual*. http://psicosemioticavisual. blogspot.com/2011/08/la-contaminacion-visual-esta-en-la-mente.html. Consulta: 16 de enero de 2017.

1.4.3.1. Historia

Este término dio inicio entre 1851 y 1860, cuando Fechner, un médico de la Universidad de Leipzig, propuso el siguiente principio: “La intensidad de una sensación, se incrementa a lo largo del estímulo”[3], el cual se representa por medio de la siguiente ecuación:

$$S = k \log I \qquad [1]$$

La ley de Fechner dice que la sensación (S) es proporcional al logaritmo de la intensidad física del estímulo (I).

Sin embargo, estudios posteriores demostraron que su ley puede no ser tan certera como él la había descrito. Uno de esos estudios fue elaborado por S.S. Stevens, graduado en psicología de la universidad de Standford (*A.B. in psychology*), quien muestra que existe una relación más favorable entre la magnitud del estímulo con la magnitud de la sensación.

S.S. Stevens plantea que la relación se produce mediante una relación de ley de potencia, donde se describe que la sensación(S) crece en proporción a la intensidad física del estímulo (I) a una potencia.

$$S = k\ I^{b} \qquad [2]$$

Donde el coeficiente b es una constate utilizada para determinar la dimensión sensorial.

[3] *Teoría para el diseño visual*. http://psicosemioticavisual. blogspot.com/2011/08/la-contaminacion-visual-esta-en-la-mente.html. Consulta: 16 de enero de 2017.

1.4.4. Atención selectiva

La visión constantemente recibe cantidades enormes de información; sin embargo, es imposible que el receptor preste atención a cada una de ellas. Debido a que existe solo un límite de datos que el receptor pueda procesar, debe ser capaz de elegir los datos de su interés.

1.4.5. Atención dividida

Cuando existen situaciones que requieren procesar información múltiple, la atención se dirige a una sola fuente de estímulo desprecia el resto de los estímulos. Esto es muy sencillo de comprobar: se toma una moneda con la mano y estira el brazo totalmente, se debe concentrar la vista en la moneda durante unos segundos. Luego, sin quitar la vista de la moneda se debe hacer lo posible por ver los elementos alrededor y atrás de ella. Se podrá observar que la mayor parte de elementos capturados por el ojo alrededor de la moneda se miran borrosos.

Este sencillo experimento comprueba una gran verdad: la vista es limitada y no puede enfocarse en más de un objeto a la vez.

1.5. Matemática del procesamiento de imagen

Desde el punto de vista matemático, las imágenes están representadas por matrices, en donde lo que se desea es sumar dos matrices. Dentro de la representación matricial las coordenadas de una imagen representan la posición dentro de la matriz, mientras que la intensidad de la imagen es representada por los valores dados de la matriz.

1.5.1. Operaciones aritméticas

Si a la matriz de una imagen se le suman 100 pixeles, la imagen resultante será por lo general más clara, debido a que se modifican los valores de intensidad. Esto se puede ver claramente en el siguiente ejemplo, en donde se eligió una imagen con tamaño de 500 pixeles por 375 pixeles. Por medio del programa de Matlab se le sumó el valor de 100 a cada pixel.

Tabla I. **Ejemplo de suma**

Código utilizado en Matlab 2014	
>> imagen=imread('1.png');	Importar la imagen en sus valores matriciales
>> suma=imagen+100;	Sumar el valor de 100 en cada valor de la matriz
>> imshow(suma)	Mostrar la imagen

Fuente: elaboración propia.

Con el código correspondiente se observa que la imagen de la izquierda es más clara que la imagen original de la derecha.

Tabla II. **Suma de valor de 100**

	**+100=**	
Imagen original		Imagen sumada

Fuente: elaboración propia.

Por el contrario, si a la imagen se le restan 100 pixeles, se pensaría que la imagen debería ser más oscura en su totalidad; sin embargo, al realizar la operación se puede observar que esto no sucede del todo, debido a que existen algunas partes claras en la imagen.

Tabla III. **Resta**

Código utilizado en Matlab 2014	
>> imagen=imread('1.png');	Importar la imagen en sus valores matriciales
>> resta=imagen-100;	Restar el valor de 100 en cada valor de la matriz
>> imshow(resta)	Mostrar la imagen

Fuente: elaboración propia.

Tabla IV. **Resta del valor de 100**

	**-100=**	
Imagen original		Imagen restada

Fuente: elaboración propia.

Para que este procedimiento sea más claro se debe observar que los valores de la intensidad van a variar dependiendo del valor numérico de la matriz; cada pixel puede tener un valor entre 0 y 255.

Por lo tanto, si un pixel vale 190 y a este se le suma 100, el resultado sería 34. Esta operación se puede observar en la tabla V. Claramente la suma de 100 + 190 es igual a 290, que es representado por un numero de 9 bits; sin embargo, como los pixeles solo pueden contener de 0 a 255 (8 bits) el noveno bit, que es el más importante (identificado por el color rojo en la tabla V), se pierde para que pueda ser almacenado el valor. Esto da como resultado el número binario 00100010 (identificado por el color en celeste en la tabla V) que representa el número 34 en sistema decimal.

Entonces, en este caso, el color de ese pixel ha pasado de un valor de 290 a un valor de 34, que sería un color más claro.

Tabla V. **Suma de números binarios para imágenes**

Decimal	Binario								
190		1	0	1	1	1	1	1	0
+ 100	+	0	1	1	0	0	1	0	0
290	1	0	0	1	0	0	0	1	0
	34								

Fuente: elaboración propia.

En el caso de la resta, es importante recordar que los valores de la representación matricial de cada imagen siempre se consideran positivos; por lo tanto, al realizar una resta, usualmente se utiliza el complemento A2, en donde el bit más significativo indica el signo positivo o negativo, siendo negativo en el valor de 1.

Para aplicar el complemento A2 se toma el número 100 y para que este sea -100, se busca el primer 1 de derecha a izquierda, en donde los números siguientes a este se remplazan por su inverso; por lo tanto, el numero -100 estaría representado en sistema binario como 00011100. Luego de esto los

números en binario se suman, se obtiene así 11011010 que equivale al número 218 en sistema decimal y al número 90 pero sin signo.

En este caso se observa que el valor del pixel describe un aumento en su intensidad.

Tabla VI. **Aumento de intensidad**

Decimal	Binario								
190		1	0	1	1	1	1	1	0
-100		0	0	0	1	1	1	0	0
90		1	1	0	1	1	0	1	0
	218								

Fuente: elaboración propia.

1.5.2. Operaciones aritméticas con saturación

La saturación se da cuando se realiza una suma a los valores matriciales de una imagen. Cuando los valores de la matriz van en aumento y estos llegan a 255, este valor queda retenido como el más alto.

En el programa de Matlab se puede variar el rango de saturación. En el siguiente ejemplo se toman los pixeles de 0 a 50.

Tabla VII. **Saturación de 0 a 50 1**

Código utilizado en Matlab 2014	
>> imagen=imread('1.png');	Importar la imagen en sus valores matriciales
>> imshow(resta,[0 50])	Mostrar la imagen, saturación de pixels de 0 a 50.

Fuente: elaboración propia.

Tabla VIII. **Saturación de 0 a 50 2**

	F[0 50]	
Imagen original		Imagen saturada

Fuente: elaboración propia.

De la misma forma, al restar a los valores matriciales de una imagen, debido a su decremento pasará por números negativos hasta retenerse en 0.

Tabla IX. **Saturación de 200 a 255**

Código utilizado en Matlab 2014	
>> imagen=imread('1.png');	Importar la imagen en sus valores matriciales
>> imshow(resta,[200 255])	Mostrar la imagen, saturación de pixels de 200 a 255.

Fuente: elaboración propia.

Tabla X. **Comparación de original y saturada**

	F[200 255]	
Imagen original		Imagen saturada

Fuente: elaboración propia.

1.5.3. Operaciones lógicas

Las operaciones lógicas, así como las aritméticas; se emplean para operar los pixeles de las imágenes. Entre las operaciones principales se encuentran:

- AND
- OR
- NOT
- XOR

Dentro de las imágenes solo es posible reconocer el valor falso, que es cuando el pixel tiene un valor de 0, mientras que el valor verdadero es cualquier valor distinto de 0.

La operación lógica OR trabaja de manera que para que un valor sea verdadero, solo necesita que uno de los dos valores entrantes sea verdadero.

Por medio de la operación lógica NOT se obtiene el inverso. Con la operación lógica XOR se obtiene un valor verdadero cuando uno de los dos valores es verdadero; sin embargo, si ambos valores son verdaderos, el resultado al final será falso.

De todas las operaciones lógicas, una de las más utilizadas es la operación lógica AND, en donde un valor es verdadero cuando ambos valores son verdaderos. Esta operación es muy útil debido a que permite el uso de máscaras. Estas permiten operar solo un grupo específico de píxeles y resguarda el resto de la información.

En el siguiente ejemplo se realiza una operación AND representada por el símbolo “&” entre la imagen original (A) y la resta (B) obtenida en la tabla XI.

Tabla XI. **Operación lógica AND**

.

Código utilizado en Matlab 2014	
>> imagen=imread('1.png');	Importar la imagen en sus valores matriciales
>> resta=imagen-100;	Restar el valor de 100 en cada valor de la matriz
>> and= imagen & resta;	Operación lógica AND entre las dos imágenes
>> imshow(and)	Mostrar la imagen

Fuente: elaboración propia.

Tabla XII. **Descripción gráfica de operaciones aritméticas y lógicas**

	**A&B**	
Imagen original (A)		
		
Imagen restada (B)		Resultado A&B

Fuente: elaboración propia.

1.5.1. Convolución

Los filtros son algoritmos que permiten resaltar características de una imagen original, lo que da como resultado una nueva imagen. Por ejemplo, si se quiere resaltar ciertos contornos en una imagen, se aplica convolución.

En el caso de las imágenes digitales, el filtrado se aplica en el dominio del tiempo y de la frecuencia. Para trabajar en el tiempo se utiliza la convolución y la correlación. Mientras, en el dominio de la frecuencia se utiliza la transformada de Fourier.

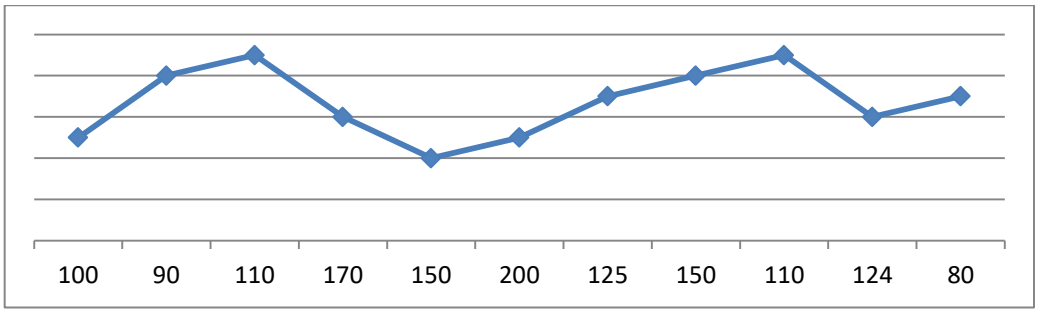
Para describir la convolución es importante recordar el concepto de derivada, el cual se describe como el incremento de una función respecto al incremento de una variable cuando esta tiende a 0.

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta(x)} \to f(x) - f(x-1) \qquad [3]$$

Es imprescindible recordar que en el tratamiento de señales discretas el incremento de la variable no puede ser tan pequeño; por lo tanto, la derivada se convierte en la diferencia entre el valor de la función menos la función evaluada en el punto anterior, como se muestra en la ecuación [3].

Un ejemplo de esto se puede observar en la siguiente gráfica. Para encontrar la derivada es necesario restar el dato actual menos el dato anterior; para ello se resta 100 menos 90 que daría como resultado 10.

Figura 3. **Derivada**



Fuente: elaboración propia.

Con el mismo procedimiento se obtiene la tabla XIII, en donde la última columna de la derecha representa la derivada. Por lo tanto, para encontrar la derivada se realiza la multiplicación de la función $f(x)$ por -1 y se le suma la

función $f(x-1)$. Al encontrar la derivada de la función se puede ver que se desplaza una función discreta sobre otra.

Tabla XIII. **Derivada**

$f(x-1)$	$f(x)$	D
100	90	10
90	110	-20
110	170	-60
170	150	20
150	200	-50
200	125	75
125	150	-25
150	110	40
110	124	-14
124	80	44

Fuente: elaboración propia.

Dentro la convolución se realiza las mismas operaciones. Como se observa en la ecuación 4, se busca encontrar una función $g(x)$ utilizando 2 funciones $h(x)$ y $f(x)$. Así como se realizó la derivada, se puede ver que la convolución es el desplazamiento de la función $h(x)$ a lo largo de la función $f(x)$. Luego se multiplican los desplazamientos $f(i)h(x-i)$ y posteriormente se suman todas las multiplicaciones.

$$g(x) = h(x) * f(x) = \sum_{i=-\infty}^{i=\infty} f(i)h(x-i) \qquad [4]$$

1.5.2. Convolución bidimensional

En el caso de la convolución para imágenes, estas se representan con la convolución bidimensional. Como se puede observar en la ecuación 5, ahora la fórmula está representada en dos dimensiones.

$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=-\infty}^{i=\infty} \sum_{j=-\infty}^{j=\infty} f(i,j)h(x-i,y-j) \qquad [5]$$

La función $h(x,y)$ no tiene un rango infinito y usualmente está representada por una matriz 3x3; debido a esto, la ecuación 5 se puede representar de dos maneras.

$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=0}^{i=2} \sum_{j=0}^{j=2} f(i,j)h(x-i,y-j) \qquad [6]$$

$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} f(i,j)h(x-i,y-j) \qquad [7]$$

Por lo tanto, una función será la imagen y la otra será una matriz bidimensional de 3x3 o 5x5 elementos. A esta operación en el dominio de la frecuencia se puede llamar filtro.

2. DISEÑO DE HARDWARE

2.1. Cámara

El elemento principal que será capaz de proveer información al sistema será la cámara. Es imprescindible cuidar la calidad de la imagen que esta proporcione. La calidad dependerá en gran medida del tipo de cámara y de sus propiedades. Entre sus propiedades se encuentran: el brillo, que es el uso de colores claros en vez de los colores obscuros; resolución, se refiere al alto y ancho de la imagen con unidad de medida en píxeles; contraste, es la relación entre los colores más claros y los más obscuros; saturación, es la intensidad de color que contiene la imagen.

La captura de una imagen a través de una cámara envuelve el proceso de conversión análogo – digital. La señal analógica pasa a trevés de un digitalizador que almacena pixeles individuales que posteriormente se visualizan y se almacenan.

2.1.1. Tipos de cámara

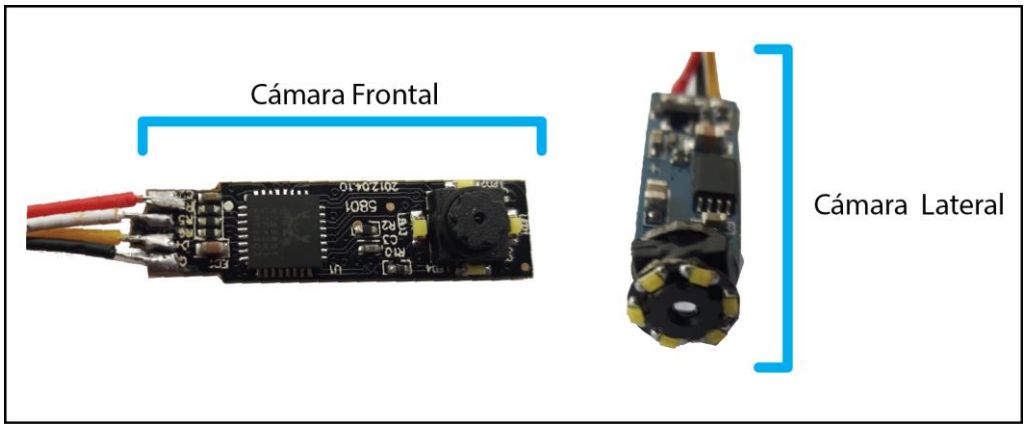
Existen diferentes tipos de cámara, en la actualidad las más utilizadas son las cámaras digitales. Debido a los requerimientos del proyecto es necesario enfocarse en el estudio de las cámaras digitales del menor tamaño posible. Un buen ejemplo de estas cámaras son las cámaras utilizadas para las endoscopías, debido a que son de un tamaño bastante reducido y la resolución de la imagen es bastante alta.

2.1.2. Cámaras utilizadas

Las seleccionadas para el proyecto son cámaras de endoscopio, livianas y pequeñas, lo que permite una perfecta adaptación al sistema portable de lentes. Así mismo, cuentan con una alta resolución que facilita el proceso y análisis de las imágenes.

Las cámaras seleccionadas se pueden observar en la figura 4. Se seleccionaron dos tipos diferentes de estructura, que se adecuan a la posición en que se colocará. La salida de las cámaras es USB, lo que permite una comunicación más fácil con el microprocesador.

Figura 4. **Cámaras de endoscopio**

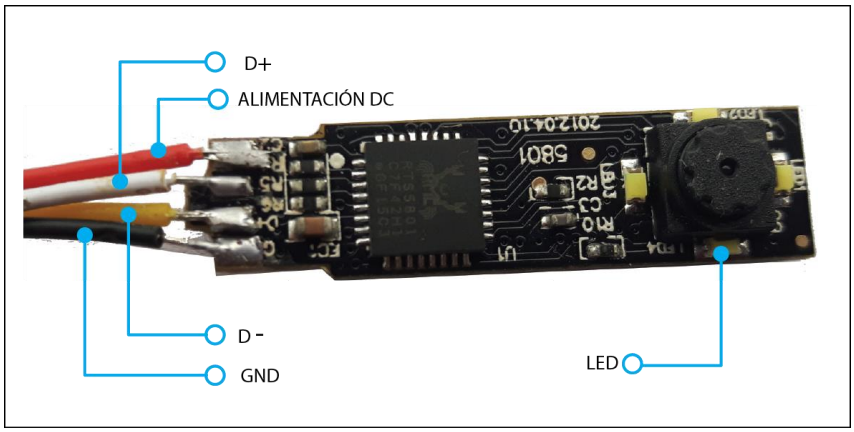


Fuente: elaboración propia, empleando Adobe Illustrator.

2.1.2.1. Conexiones de las cámaras

Las cámaras cuentan con cuatro cables; el cable rojo es para la alimentación de 5 voltios; el negro corresponde a tierra (GND); el blanco corresponde a Data + (D+) y el amarillo, a Data – (D–).

Figura 5. **Pines de las cámaras**



Fuente: elaboración propia, empleando Adobe Illustrator.

Los 4 cables se soldaron a una terminal USB tipo A macho, para establecer la comunicación con Raspberry Pi.

2.2. Tornillos

Los tornillos utilizados son de cabeza cilíndrica de 3mm diámetro, con un largo aproximado de 6mm. En la figura 6 se puede observar el tornillo y tuerca utilizados. Se recomienda que sea del menor tamaño posible; sin embargo, es prudente considerar que entre menor sea el tamaño, mayor será el costo.

Figura 6. **Tornillos y tuercas utilizadas**

Fuente: elaboración propia, empleando Adobe Illustrator.

2.3. Estructura del sistema

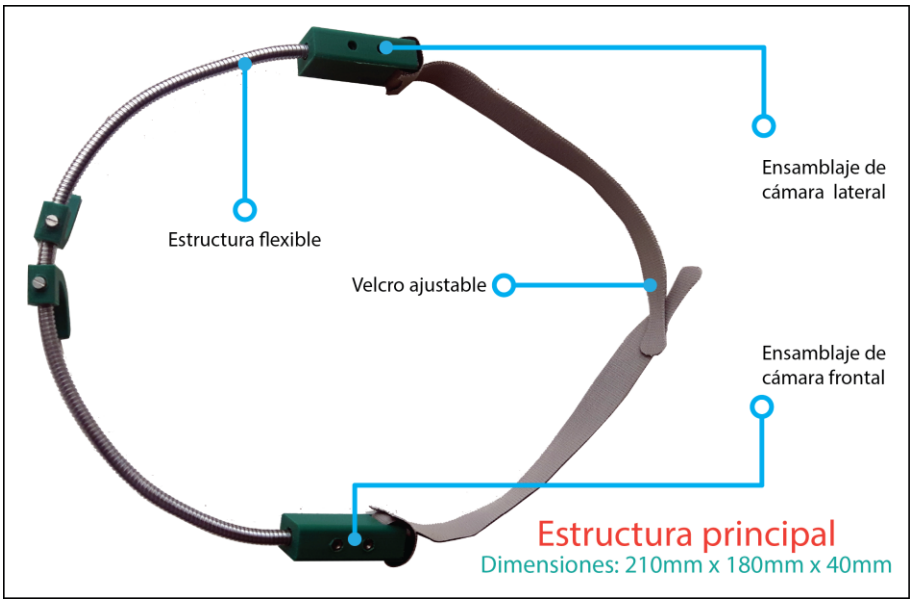
La estructura se muestra en la figura 6. Cuenta con una base parecida a lentes de plástico, seguida de pequeñas estructuras que se utilizan para sostener las cámaras; así mismo, cuenta con un mecanismo que permite mover la cámara dirigida al usuario.

La estructura fue modelada en 3D con el programa Autodesk Inventor. Posteriormente fue impresa en una impresora 3D cartesiana, utilizando PLA, que se seleccionó debido a su bajo costo y durabilidad. La estructura se ha dividido en 3 partes principales para su fácil comprensión y ensamblaje.

2.3.1. Parte 1 : estructura principal

La estructura principal está hecha de un tubo de metal flexible que permite el ajuste de la misma. Se seleccionó también un velcro especial que permite ajustar la estructura a la cabeza sin que esta se mueva. En la figura 7 se observa la estructura principal, en donde se ensamblarán las cámaras.

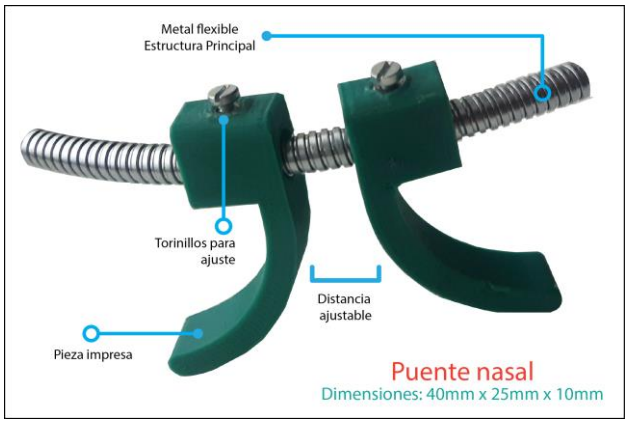
Figura 7. **Estructura principal**



Fuente: elaboración propia, empleando Adobe Illustrator.

La segunda parte, que se muestra en la figura 8, se utilizará para darle seguridad a los lentes y que estos no se caigan debido al peso de los dispositivos. Cuenta con tornillos ajustables que permiten al puente nasal expandirse o contraerse, dependiendo de la estructura del rostro.

Figura 8. **Puente nasal**

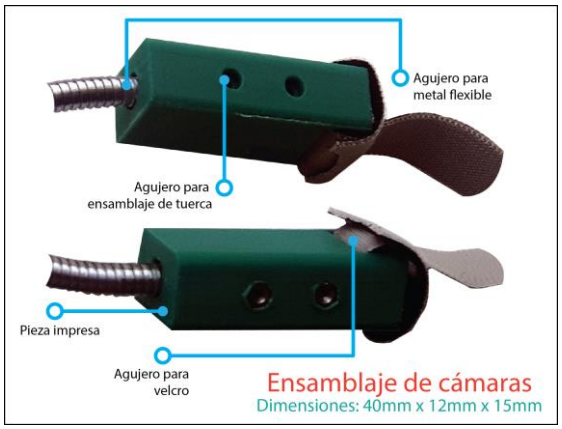


Fuente: elaboración propia, empleando Adobe Illustrator.

2.3.2. Ensamblaje para cámaras

Para montar las dos cámaras se creó un módulo rectangular que cuenta con dos tornillos que permiten el ajuste de las estructuras de cada cámara.

Figura 9. **Estructura para ensamblaje de cámaras**

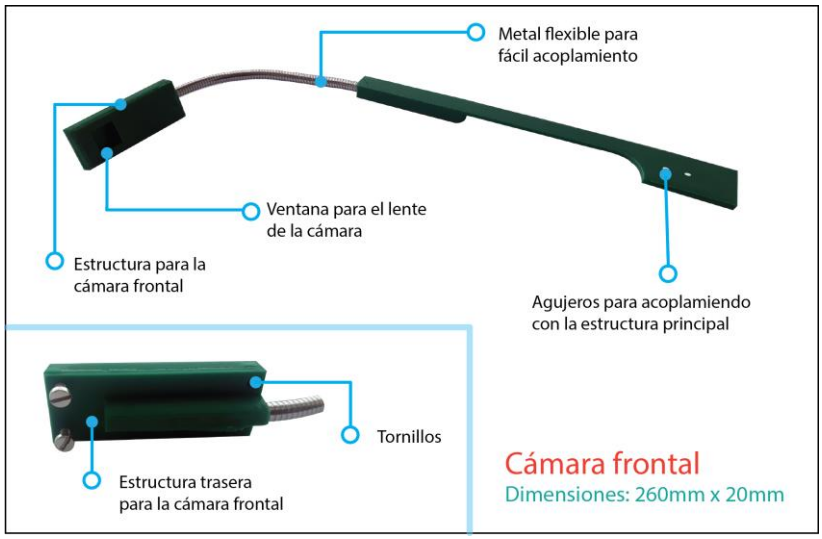


Fuente: elaboración propia, empleando Adobe Illustrator.

2.3.3. Cámara frontal

Está compuesta por una estructura de plástico que se une en uno de sus extremos a la estructura principal y en el otro extremo a una barra de metal flexible. Seguido de la barra de metal flexible se encuentra una estructura rectangular que permite colocar a la cámara frontal en su interior, encuentra asegurada por tornillos en su parte posterior.

Figura 10. **Estructura para la cámara frontal**

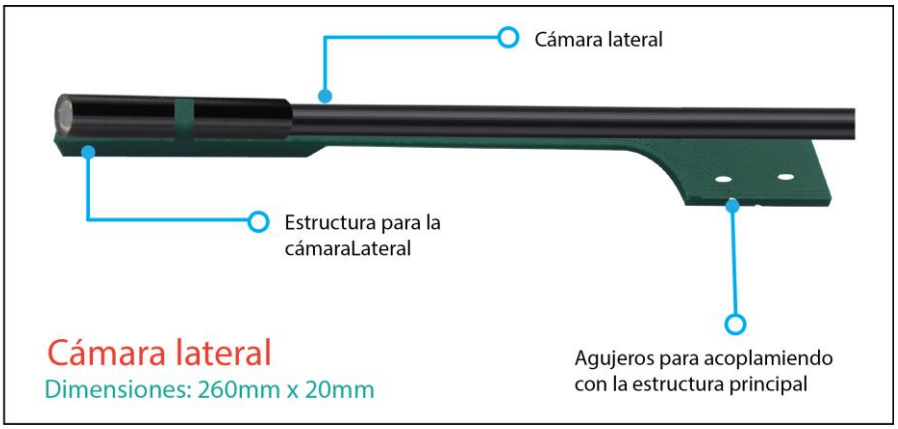


Fuente: elaboración propia, empleando Adobe Illustrator.

2.3.3.1. Cámara lateral

La estructura de la cámara lateral está compuesta únicamente por una pieza de plástico que ayuda a sostener la cámara y dirigir sus cables al dispositivo.

Figura 11. **Estructura para la cámara lateral**



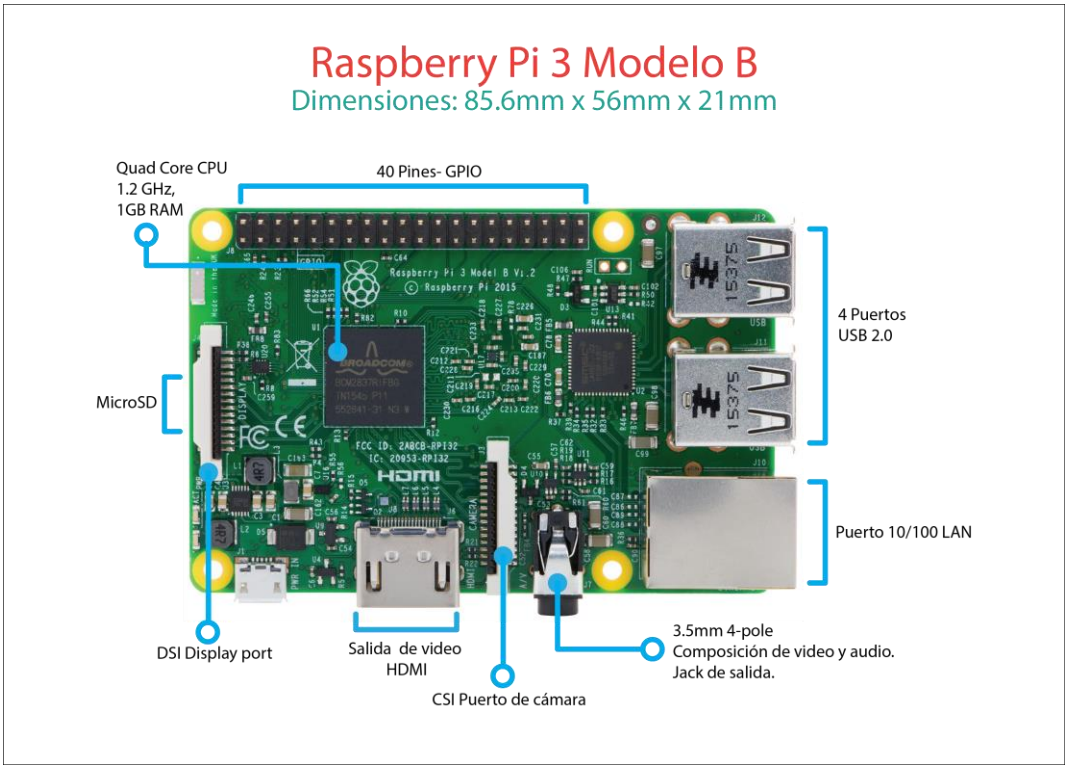
Fuente: elaboración propia, empleando Adobe Illustrator.

.

2.4. Raspberry Pi 3 Modelo B

Es la pieza central del sistema que permite el procesamiento de la imagen. La Raspberry Pi 3 Modelo B es una computadora pequeña que cuenta con un procesador Quad-Core y 1.20 GHz, RAM de 1GB, wi-fi y Bluetooth sin necesidad de adaptadores.

Figura 12. **Raspberry Pi 3 Modelo B**



Fuente: elaboración propia, empleando Adobe Illustrator.

2.5. Alimentación

La alimentación de la raspberry se establece por medio de una batería recargable, que incorpora puertos USB y una la capacidad 2 amperios.

Figura 13. **Fuente de alimentación portable**

Fuente: elaboración propia, empleando Adobe Illustrator.

2.5.1.1. Estructura final

En la Figura 14 se puede observar la unión de las partes anteriormente mencionadas, que conforman el sistema biométrico.

Figura 14. **Sistema biométrico**

Fuente: elaboración propia, empleando Adobe Illustrator.

3. DISEÑO DE SOFTWARE

3.1. Conexión remota al sistema

Para iniciar el sistema, el administrador debe acceder remotamente al dispositivo. Esto se puede realizar desde una computadora o un teléfono celular que tenga la tecnología adecuada.

Para ingresar remotamente al sistema se utilizó el programa Team Viewer. Entre las funciones de este programa se encuentran compartir dispositivos, controlar escritorios y transferir archivos entre ordenadores.

3.2. OpenCV

OpenCV (*Open Source Computer Vision Library*) traducida al español como Librería de visión por computadora gratuita, es una herramienta académica y comercial que utiliza diferentes lenguajes de programación como C++, C, Python y Java; así mismo, es soportada por diferentes interfaces como Windows, Linux, Mac OS, IOS y Android. Esta librería está diseñada para trabajar en tiempo real el tratamiento de imágenes y videos; incluye captura, procesamiento y análisis de los mismos.

En el presente proyecto se utilizó la librería OpenCV 2 junto con el lenguaje de programación de Python. Es posible dividir las funciones del programa en los siguientes incisos: análisis de la pupila, la captura de video y escritura de video.

3.2.1. Análisis de pupila

En un nivel muy básico, el proceso para detección de la pupila consiste en convertir la imagen a escala de grises, transformar la imagen a una imagen binaria (*Thresholding*), limpiar la imagen, encontrar el contorno más circular y por último, dibujar en la imagen.

- Inicialización: en la primera parte del programa se importan las librerías que se utilizarán: La librería *Numpy,* para cálculos de vectores y matrices, y *Opencv,* que permitirá análizar y procesar la imagen. Son más eficientes, comparadas con el soporte nativo de Python.

- Instanciar cámara: para instanciar la cámara se utiliza la función de captura de Opencv y se especifica el puerto del dispositivo de video en el que se conectará la cámara.

- Procesamiento: primero se transforma a escala de grises la imagen capturada por la cámara; luego se transforma la imagen en una imagen binaria (*Thresholding*) donde se especifica un parámetro numérico. Todo lo que sea más negro que ese parámetro se convertirá en negro y el resto será transformado en blanco.

Posteriormente se crea un ciclo en el que se recorre la imagen binaria y se buscan circunferencias o elipses dentro de ella. Se colocan clasificadores que permiten encontrar de una forma más efectiva el centro de la pupila. Las coordenadas del centro de la pupila son capturadas y almacenadas.

3.2.2. Captura de video

Para capturar el video de las cámaras dentro del entorno es importante verificar que existan las librerías necesarias para Python. Por tal motivo, se analizarán los siguientes parámetros:

- Salida de video: es el formato en el que se escribirá el video capturado. Existen varias extensiones, entre las más populares se encuentran; .avi, .mp4, .mov.

- Cámara raspberry pi: se debe de especificar el valor mayor a cero para acceder al módulo de la cámara de la *raspberry pi*.

- FPS: controla la cantidad de cuadros por segundo con la que se desea procesar y grabar el video.

- Códec: este elemento identifica al video, el formato de compresión y el formato de pixeles/color. Entre los codecs más populares se encuentran MJPEG, DIVX y H264.

La combinación del códec y el formato de salida es uno de los pasos más importantes. Existen docenas de combinaciones y la mayoría son impredecibles, muchas veces pueden o no funcionar entre sí.

La combinación recomendada es un códec MJPEG con una extensión de salida .avi la cual permite escribir el video reduciendo su peso.

3.3. Escritura de video

Para la escritura del video, luego realizar las inicializaciones necesarias, se utilizan funciones propias de Opencv para dibujar. En este sistema se utilizó un círculo. En la siguiente figura se puede observar un ejemplo del código utilizado; se indica el punto en donde se ubicará el centro del círculo seguido del radio y el color. Existen otros parámetros para esta función, por ejemplo: indicar el grosor de línea, que en este caso, es tres.

Figura 15. **Circunferencia en Opencv**

```
circle(img, Point(cols / 2, rows / 2), 250, Scalar(255,0,0), 3);
```

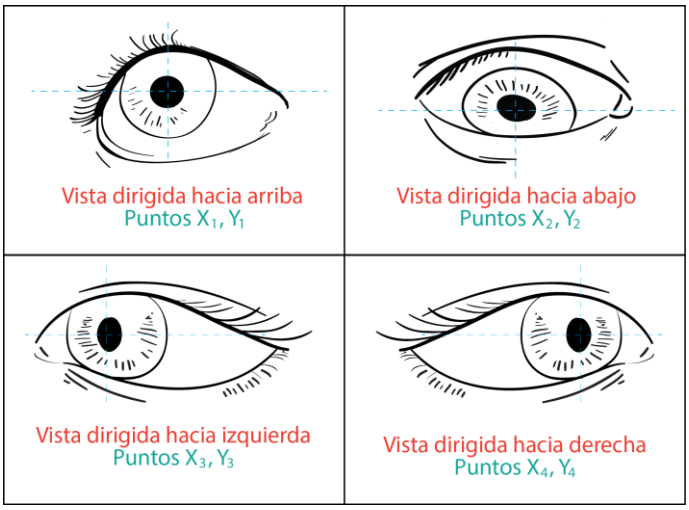
Fuente: elaboración propia, empleando Adobe Illustrator.

Las coordenadas del centro del círculo que será dibujado están dadas por el análisis de la pupila de la cámara frontal. Sin embargo, para ello se debe establecer una relación de proporción, que se abordará en los siguientes incisos.

3.4. Calibración y despliegue de video

Para realizar la calibración se le solicita al usuario portador del sistema que fije su vista hacia arriba, abajo, izquierda y derecha, para capturar cuatro imágenes. Posteriormente, por medio del análisis de pupila mencionado se encuentran cuatro pares de coordenadas.

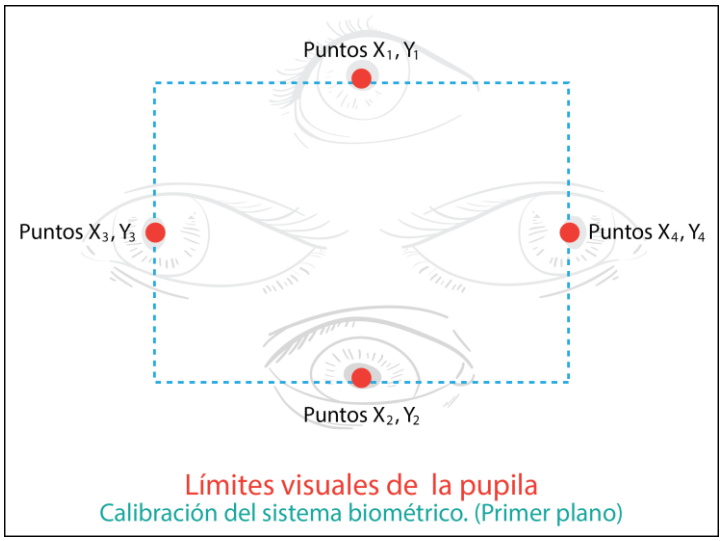
Figura 16. **Calibración del sistema mediante la pupila**



Fuente: elaboración propia, empleando Adobe Illustrator.

Al ubicar estos 4 pares de coordenadas se obtendrá un plano con los límites visuales de la pupila, como se muestra en la figura 17. Debido a que es necesario ubicar este plano dentro del plano de la cámara lateral, que cuenta con una resolución de 640 píxeles por 480 píxeles, es necesario realizar dos transformaciones geométricas: una de ellas será una traslación y la homotecia.

Figura 17. **Límites visuales de la pupila**



Fuente: elaboración propia, empleando Adobe Illustrator.

3.4.1. Traslación

Para la transformación de traslación se debe tomar el primer plano (límites visuales de la pupila) y ubicarlo en cero con respecto a la imagen captada por la segunda cámara. El eje Y es representado por la siguiente ecuación donde el valor de v_1 debe ser igual al inverso aditivo de Y_1.

$$v_1 = -Y_1$$

$$\begin{bmatrix} X_1{}' \\ Y_1{}' \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} + \begin{bmatrix} 0 \\ v_1 \end{bmatrix}$$

$$\begin{bmatrix} X_2{}' \\ Y_2{}' \end{bmatrix} = \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ v_1 \end{bmatrix}$$

El eje X se representa de la siguiente manera, donde v_2 debe ser igual al inverso de X_3

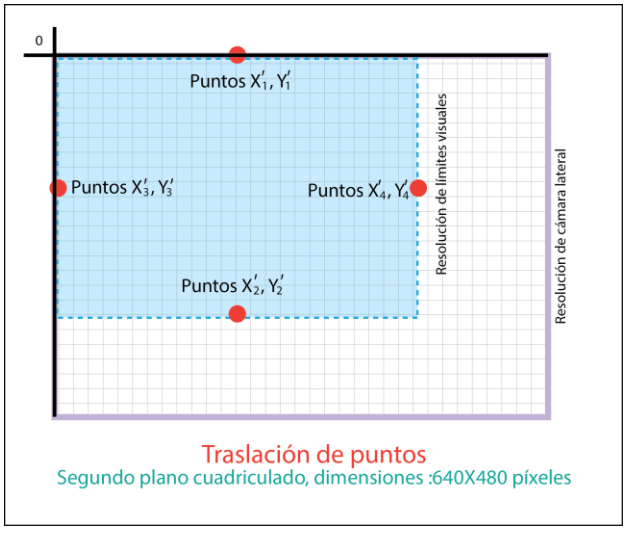
$$v_2 = -Y_2$$

$$\begin{bmatrix} X_3{}' \\ Y_3{}' \end{bmatrix} = \begin{bmatrix} X_3 \\ Y_3 \end{bmatrix} + \begin{bmatrix} 0 \\ v_2 \end{bmatrix}$$

$$\begin{bmatrix} X_4{}' \\ Y_4{}' \end{bmatrix} = \begin{bmatrix} X_4 \\ Y_4 \end{bmatrix} + \begin{bmatrix} 0 \\ v_2 \end{bmatrix}$$

Al realizar las operaciones anteriores se logra centrar en cero la esquina superior izquierda de la imagen, como se puede ver gráficamente en la figura 18.

Figura 18. **Traslación de primer plano**

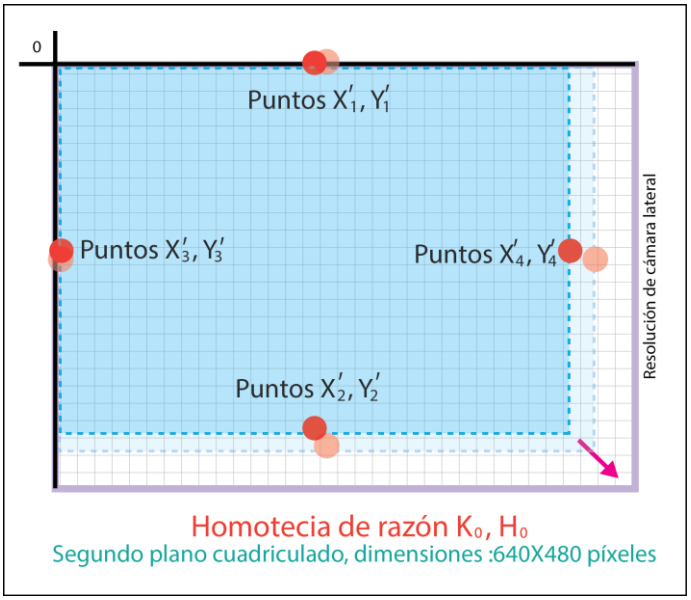


Fuente: elaboración propia, empleando Adobe Illustrator.

3.4.2. Homotecia

La siguiente es una transformación que cambia el tamaño del plano sin variar su forma. Como se muestra en la figura 15, el plano cambia de tamaño hasta llegar a la resolución de la cámara lateral.

Figura 19. **Homotecia**



Fuente: elaboración propia, empleando Adobe Illustrator.

Esta transformación se puede describir de la siguiente manera, en donde los puntos X y Y serán las coordenadas que se reciban de la cámara frontal y serán ampliados mediante la multiplicación de los factores K_0 y H_0 .

$$\begin{bmatrix} C_x \\ C_y \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} \cdot \begin{bmatrix} K_0 & 0 \\ 0 & H_0 \end{bmatrix}$$

$$K_0 = \frac{640}{X_4'}$$

$$H_0 = \frac{480}{X_2'}$$

Por último las coordenadas encontradas C_x y C_y se dibujaran encima del video capturado por la cámara lateral.

3.5. Programación con hilos

La programación con hilos es ejecutar procesos independientes al mismo tiempo, compartiendo los recursos en el sistema.

En el desarrollo del programa se utilizaron dos hilos, uno para tomar fotografías de manera frecuente del ojo y analizar la imagen posteriormente. El otro hilo se utilizó para la segunda cámara, que es encargada de capturar el video en tiempo real, y escribir para encima del video hacia donde se dirige la mirada de la pupila.

4. IMPLEMENTACIÓN DEL SISTEMA

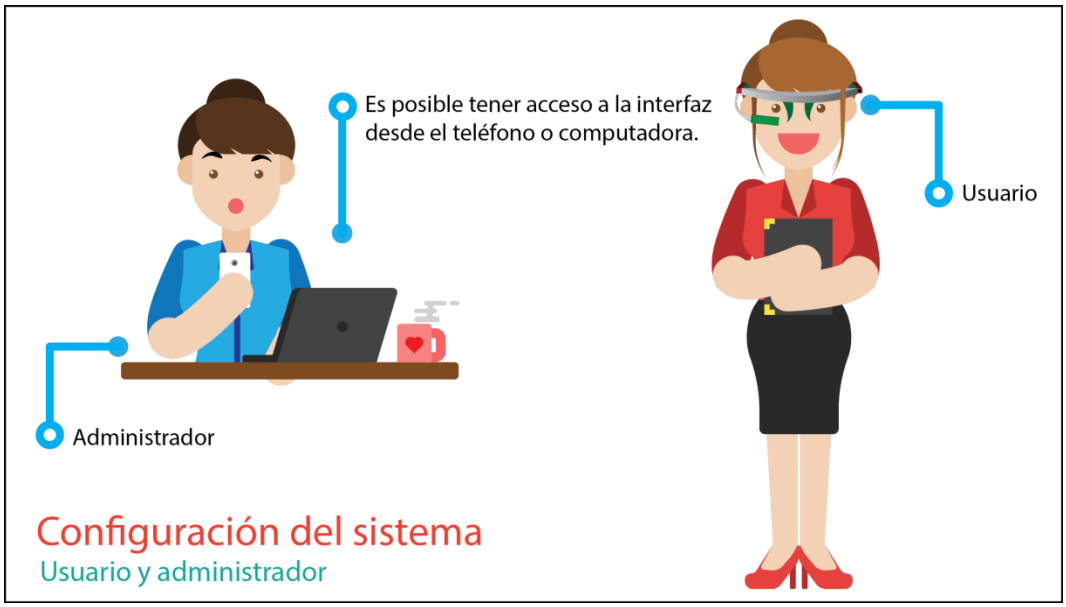
4.1. Configuración del sistema

El desarrollo será llevado a cabo de la siguiente manera: como se observa en la figura 20, el administrador accede remotamente al dispositivo que da inicio a la interfaz gráfica, en donde encontrará la primera parte de calibración.

Para la calibración se le debe solicitar al usuario que sin mover su rostro dirija su mirada hacia arriba. Luego, el administrador presiona el botón que corresponde y captura una imagen del ojo del usuario. Este procedimiento continúa hasta obtener los cuatro puntos correspondientes.

Luego de terminar la calibración, el administrador empieza el análisis de la pupila al oprimir el botón de iniciar que se encuentra en la parte de abajo de la calibración. Cuando se inicia el análisis, el sistema automáticamente empieza a grabar el video dentro del dispositivo. Por último, al terminar, el administrador puede parar el video al pulsar el botón de detener.

Figura 20. **Configuración del sistema**



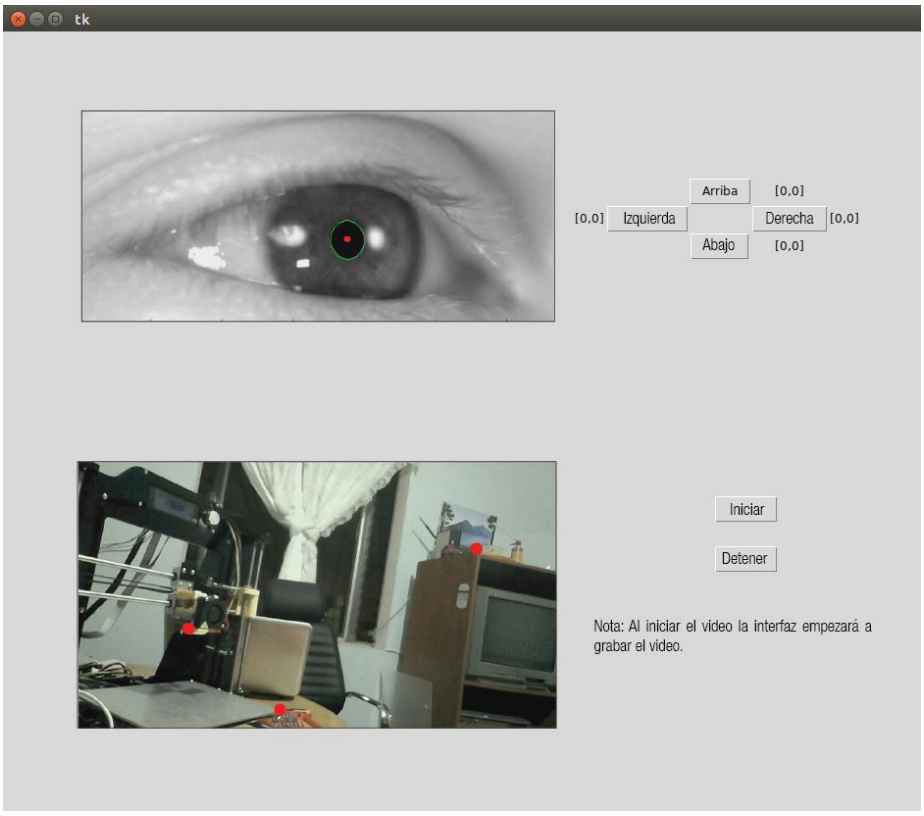
Fuente: *Designed by Freepik.* http://www.freepik.com. Consulta: 28 de junio de 2017

4.2. Interfaz para el usuario

Para la elaboración de la interfaz gráfica del usuario se utilizó la herramienta TkInter.

En la figura 21 se muestra la interfaz creada para el administrador. En la parte de arriba se encuentra la calibración y en la parte abajo, el video en vivo que muestra el recorrido de la pupila que se obtuvo del procesamiento de la imagen.

Figura 21. **Interfaz gráfica**



Fuente: elaboración propia, empleando Adobe Illustrator.

4.3. Presupuesto

El costo total del proyecto fue de mil novecientos quetzales que es aproximadamente un doceavo de lo que cuesta un modelo de una marca estándar.

Tabla XIV. **Presupuesto**

Material	Costo
Cámara lateral	Q 250,00
Cámara frontal	Q 350,00
Raspberry Pi 3 B	Q 450,00
Estructura	Q 200,00
Impresiones 3D	Q 300,00
Batería	Q 300,00
Cables y componentes	Q 50,00
Total	Q 1 900,00

Fuente: elaboración propia.

CONCLUSIONES

1. Se implementó un sistema biométrico que es capaz de grabar la interacción ocular del usuario portador con su entorno físico.

2. Se implementó un sistema de bajo costo con una estructura que contiene una cámara frontal, cámara lateral, una batería, una estructura impresa en 3D y una Raspberry Pi 3B como procesador.

3. Se capturaron datos de las coordenadas oculares, los cuales pueden ser desplegados en un video, almacenados y utilizados para investigaciones en los sistemas visuales.

4. Se almacenaron y desplegaron los datos por medio de una interfaz gráfica amigable al usuario, que permite grabar el recorrido visual en un formato de fácil manejo.

RECOMENDACIONES

1. Utilizar las librerías más actualizadas de OpenCv, facilitar la elaboración del software.

2. Tomar en cuenta diversos proyectos de código abierto que pueden mejorar el análisis de la pupila.

3. Con este sistema biométrico es posible desarrollar más algoritmos que permitan cuantificar, de una manera apropiada, la interacción del usuario con estímulos físicos.

4. Integrar clasificadores que permitan una mejor precisión en el análisis y procesamiento de la pupila.

5. Estudiar la posibilidad de integrar el sistema con la encefalografía para estudios más completos.

BIBLIOGRAFÍA

1. *Aplicación para acceso remoto*. [en línea]. <https://www.teamviewer.com/es/>. [Consulta: 20 de febrero de 2017].

2. *Breve introducción a TkInter*. [en línea]. <http://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html>. [Consulta: 20 de marzo de 2017].

3. *Detección básica de pupila.* [en línea]. <http://blog.mirosval.sk/master-thesis/english/2014/01/29/pupil-detection.html> [Consulta: 5 de abril de 2017].

4. *Documentación de Opencv*. [en línea] <http://opencv.org/>. [Consulta: 30 de Junio de 2017].

5. DUCHOWSKI, Ariel. *Eye-tracking methodology: Theory and practice*. Inglaterra: Springer, 2003. 251 p.

6. GRAUE WIECHERS, Enrique. *Oftamología*. 3a ed. México: McGraw-Hill. 2009. 884 p.

7. MATHOT, Dalmaijer y STIGCHEL, Vander. *PyGaze, the open-source toolbox for eye tracking*. [en línea]. <http: ://www.pygaze.org/contributores>. [Consulta: 25 de febrero de 2017].

8. POOLE, David. *Algebra lineal una introducción moderna*. 2a ed. España: Ediciones Paraninfo, 2010. 472 p.

9. PURVES, Agustine; FITZPATRICK, Hall. *LaMantia, White, Neurociencia*. 5a ed. México: Médica panamericana. 2016. 759 p.

10. *Teoría para el diseño visual*. [en línea]. <http://psicosemioticavisual. blogspot.com/2011/08/la-contaminacion-visual-esta-en-la-men te . html>. [Consulta: 16 de enero de 2017].

11. UMBAUGH, S. *Image enhancement through gray-scale modification. Computer Vision and Image Processing*. London: Pretince-Hall. 2009. 100 p.

12. *Visión por computadora*. [en línea]. <http://www.pyimagesearch.com/ category/image-processing/>. [Consulta: 2 de febrero de 2017].

13. WRIGHT, William. *The rays are not coloured.* [en línea] <www.readcube.com/articles/10.1038/1981239a0>. [Consulta: 5 de febrero de 2017].

APÉNDICE

Apéndice1. **Código utilizado**

```
# CamTracker GUI test

from camtracker import Setup

# initialize setup
setup = Setup()

# run GUI
tracker = setup.start()
#_____________________________________________________________

# Webcam EyeTracker classes
# Edwin S. Dalmaijer
# version original 0.1, 12-10-2013
# Modificación:  0.2, 20-07-2013

# in DEBUG mode, images of the calibration are saved as strings in a textfile,
# after the calibration, the textfile will be read and PNG images will be
# produced based on the content of the file
DEBUG = False
BUFFSEP = 'edwinisdebeste'
```

Continuación del apéndice 1.

```
        # # # # # #
        # imports #

        import os.path
        import numpy as np
        import cv2
        import threading

        # Try to import VideoCapture Library
        # Requires VideoCapture & PIL libraries
        vcAvailable = False
        import imp
        try:
          imp.find_module('VideoCapture')
          vcAvailable = True
          import VideoCapture
        except ImportError:
          print "VideoCapture module not available"

        try:
          import pygame
          import pygame.camera
          pygame.init()
          pygame.camera.init()
        except:
          raise Exception("Error in camtracker: PyGame could not be imported and
initialized! :(")
```

Continuación del apéndice 1.

```
    # # # # #
    # functions

    def available_devices():

      """Returns a list of available device names or numbers; each name or
      number can be used to pass as Setup's device keyword argument

      arguments
      None

      keyword arguments
      None

      returns
      devlist        --      a list of device names or numbers, e.g.
                                  ['/dev/video0','/dev/video1'] or [0,1]
      """

      return pygame.camera.list_cameras()


    # # # # #
    # classes

    class Setup:
```

Continuación del apéndice 1.

```
    def video(self):

        global a, b
        a=0
        b=0
        cap = cv2.VideoCapture(1)
        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
        out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

        while (True):
            a = self.settings['pupilpos'][0]
            b = self.settings['pupilpos'][1]
            # Captura de frame por frame
            ret, frame = cap.read()

            # Operaciones en el Frame, (Centro), Radio,
            cv2.circle(frame, (a, b), 10, (0, 0, 255), -1)
            #out.write(frame)
            # Desplegar los resultados

            cv2.imshow('frame', frame)
            # out.write(frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Salida con "q", destruir las ventanas
        cap.release()
```

Continuación del apéndice 1.

```
        cv2.destroyAllWindows()

    """The Setup class provides means to calibrate your webcam to function
    as an eye tracker"""
        def __init__(self, device='/dev/video2', camres=(640,480),
disptype='window', dispres=(1024,768), display=None):

        """Initializes a Setup instance

        arguments
        None

        keyword arguments
        device      --  a string or an integer, indicating either
                        device name (e.g. '/dev/video0'), or a
                        device number (e.g. 0); None can be
passed
                        too, in this case Setup will autodetect a
                        useable device (default = None)
        camres      --  the resolution of the webcam, e.g.
                        (640,480) (default = (640,480))
        disptype    --  a string indicating what kind of
                        calibration display should be presented;
                        choose from 'window' (PyGame
windowed),
                        'fullscreen' (PyGame fullscreen)
                        (default = 'window')
```

Continuación del apéndice 1.

```
        dispres         --  the resolution of the display, e.g.
                            (1280,1024) (default = 1024,768)
        display         --  pass None to let the Setup create its
own
                            display, otherwise pass a display that
                            matches the disptype you provided
(under
                            the disptype argument) to let the Setup
use
                            that display; example: set disptype to
                            'fullscreen', then pass a
                            pygame.surface.Surface instance that is
                            returned by pygame.display.set_mode:
                            calibration will then be presented on the
                            passed pygame.surface.Surface
instance
        """


        # DEBUG #
        if DEBUG:
                self.savefile = open('data/savefile.txt','w')
        # # # # #

        # create new Display if none was passed, or use the provided
display
        if display == None:
```

Continuación del apéndice 1.

```
                if disptype == 'window':
                        self.disp = pygame.display.set_mode(dispres,
pygame.RESIZABLE)
                elif disptype == 'fullscreen':
                        self.disp = pygame.display.set_mode(dispres,
pygame.FULLSCREEN|pygame.HWSURFACE|pygame.DOUBLEBUF)
                else:
                        raise Exception("Error in camtracker.Setup.__init__:
disptype '%s' was not recognized; please use 'window', 'fullscreen'")
        # if a display was specified, use that
        else:
                self.disp = display
                dispres = self.disp.get_size()

        # select a device if none was selected
        if device == None:
                available = available_devices()
                if available == []:
                        raise Exception("Error in camtracker.Setup.__init__:
no available camera devices found (did you forget to plug it in?)")
                else:
                        device = available[0]

        # create new camera
        self.tracker = CamEyeTracker(device=device, camres=camres)

        # find font: first look in directory, if that fails we fall back to default
```

Continuación del apéndice 1.

```
            try:
                fontname = os.path.join(os.path.split(os.path.abspath(__file__))[0],'resources','roboto_regular-webfont.ttf')
            except:
                fontname = pygame.font.get_default_font()
                print("WARNING: camtracker.Setup.__init__: could not find 'roboto_regular-webfont.ttf' in the resources directory!")
            # create a Font instance
            self.font = pygame.font.Font(fontname, 24)
            self.sfont = pygame.font.Font(fontname, 12)

            # set some properties
            self.disptype = disptype
            self.dispsize = dispres
            self.fgc = (255,255,255)
            self.bgc = (0,0,0)

            # fill display with background colour
            self.disp.fill(self.bgc)

            # set some more properties
            self.img = pygame.surface.Surface(self.tracker.get_size())    # empty surface, gets filled out with camera images
            self.settings = {'pupilcol':(0,0,0), \
                             'threshold':100, \
                             'nonthresholdcol':(100,100,255,255), \
```

Continuación del apéndice 1.

```
                                    'pupilpos': (camres[0]/2,camres[1]/2), \
                                    'pupilrect':pygame.Rect(camres[0]/2-
50,camres[1]/2-25,100,50), \
                                    'pupilbounds': [0,0,0,0], \
                                    '':None
                                    }
                # variables that hold calibration for the eye
                self.array_up = (0,0)
                self.array_down = (0,0)
                self.array_left = (0,0)
                self.array_right = (0,0)
                hilo2 = threading.Thread(target=self.video)
                hilo2.start()

        def start(self):

                """Starts running the GUI

                arguments
                None

                keyword arguments
                None

                returns
                None
                """
```

Continuación del apéndice 1.

```
            # show welcoming screen (loading...)
            self.show_welcome(loading=True)

            # DEBUG #
            if DEBUG:

        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
            # # # # #

            # create GUI
            self.setup_GUI()

            # replace 'loading' on welcoming screen with 'press any key to
start'
            self.show_welcome(loading=False)



            # DEBUG #
            if DEBUG:

        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
            # # # # #

            # wait for keypress
            noinput = True
            while noinput:
```

Continuación del apéndice 1.

```
                    for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN:
                            noinput = False

            # show welcoming screen (and we're loading again)
            self.show_welcome(loading=True)

            # DEBUG #
            if DEBUG:

        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
            # # # # #

            # draw general GUI (no stage information yet)
            self.draw_stage(stagenr=None)

            # DEBUG #
            if DEBUG:

        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
            # # # # #

            # mouse visibility
            pygame.mouse.set_visible(True)

            # start setup (should return a CamEyeTracker instance)
            tracker = self.run_GUI()
```

Continuación del apéndice 1.

```
        return tracker


    def show_welcome(self, loading=False):

        """Shows a welcoming screen with package and author information,
        either depecting "Loading, please wait..." or "Press any key to
        start", depending on the loading argument

        arguments
        None

        keyword arguments
        loading         --  Boolean indicating whether welcoming screen
                            should say "Loading, please wait..." or
                            "Press any key to start"

        returns
        None            --  directly draws on self.disp
        """

        # welcome text
        welcometext = \
        """Welcome to the Webcam EyeTracker calibration interface!
```

Continuación del apéndice 1.

```
        author: Edwin Dalmaijer
        version: 0.1 (12-10-2013)



        """

        # reset display
        self.disp.fill(self.bgc)

        # loading message
        if loading:
                welcometext += "Loading, please wait..."
        else:
                welcometext += "Press any key to start!"

        # remove tabs from text
        welcometext = welcometext.replace("\t","")

        # draw lines on display
        x = self.dispsize[0]/2; y = self.dispsize[0]/2
        lines = welcometext.split("\n")
        nlines = len(lines)
        for lnr in range(nlines):
                # render text
                linesize = self.font.size(lines[lnr])
```

Continuación del apéndice 1.

```
                    rendered = self.font.render(lines[lnr], True, self.fgc) #
Font.render(text, antialias, color, background=None)
                    # position
                    pos = (x-linesize[0]/2, y + (lnr - nlines/2)*linesize[1])
                    # draw to disp
                    self.disp.blit(rendered, pos)

            # update!
            pygame.display.flip()


    def setup_GUI(self):

            """Sets up a GUI interface within a PyGame Surface

            arguments
            None

            keyword arguments
            None

            returns
            None                --      returns nothing, but draws on self.disp
and
                                        sets self.guisurface
            """
```

Continuación del apéndice 1.

```
        # directory
        resdir = os.path.join(os.path.split(os.path.abspath(__file__))[0],'resources')
        if not os.path.exists(resdir):
            raise Exception("Error in camtracker.Setup.setup_GUI: could not find 'resources' directory to access button images; was it relocated or renamed, or is the installation of camtracker incorrect?")

        # find image paths
        imgpaths = {}
        buttnames = ['1','2','3','up','down','t','space','r','escape']
        buttstates = ['active','inactive']
        for bn in buttnames:
            imgpaths[bn] = {}
            for bs in buttstates:
                filename = "%s_%s.png" % (bn,bs)
                imgpaths[bn][bs] = os.path.join(resdir,filename)
                if not os.path.isfile(imgpaths[bn][bs]):
                    print("WARNING: image file '%s' was not found in resources!" % filename)
                    imgpaths[bn][bs] = os.path.join(resdir,"blank_%s.png" % bs)

        # image positions (image CENTERS!)
        buttsize = (50,50)
        camres = self.tracker.get_size()
        buttpos = {}
```

Continuación del apéndice 1.

```
            y = self.dispsize[1]/2 + int(camres[1]*0.6)
            buttpos['1'] = int(self.dispsize[0]*(2/6.0) - buttsize[0]/2), y
            buttpos['2'] = int(self.dispsize[0]*(3/6.0) - buttsize[0]/2), y
            buttpos['3'] = int(self.dispsize[0]*(4/6.0) - buttsize[0]/2), y
            buttpos['space'] = int(self.dispsize[0]*(5/6.0) - buttsize[0]/2), y

            leftx = self.dispsize[0]/2 - (camres[0]/2 + buttsize[0]) # center of the
buttons on the right
            rightx = self.dispsize[0]/2 + camres[0]/2 + buttsize[0] # center of the
buttons on the left
            buttpos['up'] = rightx, self.dispsize[1]/2-buttsize[1] # above
snapshot half, to the right
            buttpos['down'] = rightx, self.dispsize[1]/2+buttsize[1] # below
snapshot half, to the right
            buttpos['t'] = leftx, self.dispsize[1]/2+camres[1]/2-buttsize[1]/2 #
same level as snapshot bottom, to the left
            buttpos['r'] = leftx, self.dispsize[1]/2 # halfway snapshot (==halfway
display), to the left
            buttpos['escape'] = buttsize[0], buttsize[1] # top left

            # new dict for button properties (image, position, and rect)
            self.buttons = {}
            # loop through button names
            for bn in imgpaths.keys():
                # new dict for this button name
                self.buttons[bn] = {}
                # recalculate position
```

Continuación del apéndice 1.

```
                buttpos[bn] = buttpos[bn][0]-buttsize[0]/2, buttpos[bn][1]-
buttsize[1]/2
                # loop through button states
                for bs in imgpaths[bn].keys():
                        # new dict for this button name and this button state
                        self.buttons[bn][bs] = {}
                        # load button image
                        self.buttons[bn][bs]['img'] =
pygame.image.load(imgpaths[bn][bs])
                        # save position and rect
                        self.buttons[bn][bs]['pos'] = buttpos[bn]
                        self.buttons[bn][bs]['rect'] = buttpos[bn][0],
buttpos[bn][1], buttsize[0], buttsize[1]

        # save buttsize
        self.buttsize = buttsize


    def draw_button(self, image, pos):

        """Draws a button on the display

        arguments
        image       --  a pygame.surface.Surface instance,
depicting
                        a button
        pos         --  a (x,y) position coordinate, indicating the
```

Continuación del apéndice 1.

```
                                        top left corner of the button

            keyword arguments
            None

            returns
            None            --      directly draws on self.disp
            """

            self.disp.blit(image, pos)


    def draw_stage(self, stagenr=None):

            """Draws the GUI window for the passed stage nr

            arguments
            None

            keyword arguments
            stagenr         --      None for only the basic buttons, or a
stage
                                    number for the basic buttons, as well as
the
                                    stage specific buttons

            returns
```

Continuación del apéndice 1.

```
                None                    --      directly draws on self.disp
                """

                # clear display
                self.disp.fill(self.bgc)

                # universal buttons
                buttonstodraw = ['1','2','3','space','escape','t','r']
                activetodraw = []

                # stage specific buttons
                if stagenr == 1:
                        title = "set pupil detection threshold"
                        buttonstodraw.extend(['up','down'])
                        activetodraw.extend(['1'])
                elif stagenr == 2:
                        title = "select pupil and set pupil detection bounds"
                        buttonstodraw.extend(['up','down'])
                        activetodraw.extend(['2'])
                elif stagenr == 3:
                        title = "confirmation"
                        buttonstodraw.extend(['up','down'])
                        activetodraw.extend(['3'])
                else:
                        title = "loading, please wait..."

                # draw inactive buttons
```

Continuación del apéndice 1.

```
            for buttname in buttonstodraw:

        self.draw_button(self.buttons[buttname]['inactive']['img'],self.buttons[buttn
ame]['inactive']['pos'])

            # draw active buttons
            for buttname in activetodraw:

        self.draw_button(self.buttons[buttname]['active']['img'],self.buttons[buttna
me]['active']['pos'])

            # draw title
            titsize = self.font.size(title) # author note: LOL, 'titsize'!
            titpos      =      self.dispsize[0]/2-titsize[0]/2,      self.dispsize[1]/2-
(self.tracker.get_size()[1]/2+titsize[1])
            titsurf = self.font.render(title, True, self.fgc)
            self.disp.blit(titsurf,titpos)


        def run_GUI(self):

            """Perform a setup to set all settings using a GUI

            arguments
            None

            keyword arguments
```

Continuación del apéndice 1.

```
        None

        returns
        None            --      returns nothing, but does fill in
                                the self.settings dict
        """

        # # # # #
        # variables

        # general
        stage = 1 # stage is updated by handle_input functions

        # stage specific
        stagevars = {}

        stagevars[0] = {}
        stagevars[0]['show_threshimg'] = False # False for showing snapshots, True for showing thresholded snapshots
        stagevars[0]['use_prect'] = True # False for no pupil search limits, True for pupul rect

        stagevars[1] =  {}
        stagevars[1]['thresholdchange'] = None # None, 'up', or 'down'

        stagevars[2] = {}
```

Continuación del apéndice 1.

```
            stagevars[2]['clickpos'] = 0,0 # becomes a (x,y) tuple, indicating click position within the webcam's snapshots (to determine pupil rect)
            stagevars[2]['prectsize'] = 100,50 # pupilrectsize
            stagevars[2]['prect'] = pygame.Rect(stagevars[2]['clickpos'][0],stagevars[2]['clickpos'][1],stagevars[2]['prectsize'][0],stagevars[2]['prectsize'][1]) # rect around pupil, in which the pupil is expected to be
            stagevars[2]['vprectchange'] = None  # None, 'up', or 'down'
            stagevars[2]['hprectchange'] = None  # None, 'right', or 'left'

            stagevars[3] = {}
            stagevars[3]['confirmed'] = False

            # set Booleans
            running = True                  # turns False upon quiting the GUI

            # set image variables
            imgsize = self.img.get_size()
            blitpos = (self.dispsize[0]/2-imgsize[0]/2, self.dispsize[1]/2-imgsize[1]/2)

            # # # # #
            # run GUI
            while running:

                # # # # #
```

Continuación del apéndice 1.

```
                # general

                # draw stage
                self.draw_stage(stagenr=stage)

                # get new snapshot, thresholded image, and pupil measures
(only use pupil bounding rect after stage 1)
                useprect = stagevars[0]['use_prect'] and stage > 1
                self.img, self.thresholded, pupilpos, pupilsize, pupilbounds =
self.tracker.give_me_all(pupilrect=useprect)

                # update settings
                self.settings = self.tracker.settings

                # check if the thresholded image button is active
                if stagevars[0]['show_threshimg']:
                        # draw active button
                        self.draw_button(self.buttons['t']['active']['img'],
self.buttons['t']['active']['pos'])
                # if threshold button is not active, draw inactive button
                else:
                        self.draw_button(self.buttons['t']['inactive']['img'],
self.buttons['t']['inactive']['pos'])

                # check if the thresholded image button is active
                if stagevars[0]['use_prect']:
                        # draw active button
```

Continuación del apéndice 1.

```
                    self.draw_button(self.buttons['r']['active']['img'],
self.buttons['r']['active']['pos'])
                # if threshold button is not active, draw inactive button
                else:
                    self.draw_button(self.buttons['r']['inactive']['img'],
self.buttons['r']['inactive']['pos'])

                # check for input
                inp, inptype = self.check_input()

                # handle input, according to the stage (this changes the
stagevars!)
                stage, stagevars = self.handle_input(inptype, inp, stage,
stagevars)

                # # # # #
                # stage specific

                # stage 1: setting pupil threshold
                if stage == 1:
                    # set camera threshold
                    if stagevars[1]['thresholdchange'] != None:
                        if stagevars[1]['thresholdchange'] == 'up' and
self.settings['threshold'] < 255:
                            self.settings['threshold'] += 1
                        elif stagevars[1]['thresholdchange'] == 'down'
and self.settings['threshold'] > 0:
```

Continuación del apéndice 1.

```
                                        self.settings['threshold'] -= 1
                                    stagevars[1]['thresholdchange'] = None


                        # stage 2: select eye by clicking on it
                        if stage == 2:
                            # check if input is a mouse click
                            if type(inp) in [tuple,list]:
                                # check if mouse position is in image
                                mpos = pygame.mouse.get_pos()
                                hposok = mpos[0] > blitpos[0] and mpos[0] <
blitpos[0]+imgsize[0]
                                vposok = mpos[1] > blitpos[1] and mpos[1] <
blitpos[1]+imgsize[1]
                                if hposok and vposok:
                                    # set pupil position
                                    stagevars[2]['clickpos'] = inp[0]-
blitpos[0], inp[1]-blitpos[1]
                                    self.settings['pupilpos'] =
stagevars[2]['clickpos'][:]
                                    # set pupil rect
                                    x = stagevars[2]['clickpos'][0] -
stagevars[2]['prectsize'][0]/2
                                    y = stagevars[2]['clickpos'][1] -
stagevars[2]['prectsize'][1]/2
                                    stagevars[2]['prect'] =
pygame.Rect(x,y,stagevars[2]['prectsize'][0],stagevars[2]['prectsize'][1])
```

Continuación del apéndice 1.

```
                                        self.settings['pupilrect']                    =
stagevars[2]['prect']

                                # if input was a key or button press
                                elif             stagevars[2]['vprectchange']              or
stagevars[2]['hprectchange']:
                                        # change pupil rect size
                                        if stagevars[2]['vprectchange'] != None:
                                                if stagevars[2]['vprectchange'] == 'up':
                                                        stagevars[2]['prectsize']             =
stagevars[2]['prectsize'][0], stagevars[2]['prectsize'][1] + 1
                                                elif     stagevars[2]['vprectchange']     ==
'down':
                                                        stagevars[2]['prectsize']             =
stagevars[2]['prectsize'][0], stagevars[2]['prectsize'][1] - 1
                                                stagevars[2]['vprectchange'] = None
                                        if stagevars[2]['hprectchange'] != None:
                                                if stagevars[2]['hprectchange'] == 'right':
                                                        stagevars[2]['prectsize']             =
stagevars[2]['prectsize'][0] + 1, stagevars[2]['prectsize'][1]
                                                elif stagevars[2]['hprectchange'] == 'left':
                                                        stagevars[2]['prectsize']             =
stagevars[2]['prectsize'][0] - 1, stagevars[2]['prectsize'][1]
                                                stagevars[2]['hprectchange'] = None
                                        # set pupil rect
                                        x = self.settings['pupilrect'][0]
                                        y = self.settings['pupilrect'][1]
```

Continuación del apéndice 1.

```
                                    stagevars[2]['prect']                                    =
pygame.Rect(x,y,stagevars[2]['prectsize'][0],stagevars[2]['prectsize'][1])
                                    self.settings['pupilrect'] = stagevars[2]['prect']

                            # draw pupil rect
                            pygame.draw.rect(self.img,                    (0,0,255),
self.settings['pupilrect'], 2)
                            pygame.draw.rect(self.thresholded,            (0,0,255),
self.settings['pupilrect'], 2)

                        # stage 3: confirmation
                        if stage == 3:
                            # set camera threshold
                            if stagevars[1]['thresholdchange'] != None:
                                if  stagevars[1]['thresholdchange']  ==  'up'  and
self.settings['threshold'] < 255:
                                    self.settings['threshold'] += 1
                                elif  stagevars[1]['thresholdchange']  ==  'down'
and self.settings['threshold'] > 0:
                                    self.settings['threshold'] -= 1
                                stagevars[1]['thresholdchange'] = None
                            # draw pupil center and pupilbounds in image
                            try:                            pygame.draw.rect(self.img,
(0,255,0),pupilbounds,1);                    pygame.draw.rect(self.thresholded,
(0,255,0),pupilbounds,1)
                            except: print("pupilbounds=%s" % pupilbounds)
```

Continuación del apéndice 1.

```
                    try:                    pygame.draw.circle(self.img,
(255,0,0),pupilpos,3,0);                    pygame.draw.circle(self.thresholded,
(255,0,0),pupilpos,3,0)
                    except: print("pupilpos=%s" % pupilpos)
                    # is settings are confirmed, stop running
                    if stagevars[3]['confirmed']:
                        running = False

                #print self.settings['pupilpos']



                # draw values
                starty = self.dispsize[1]/2 - imgsize[1]/2
                vtx = self.dispsize[0]/2 - imgsize[0]/2 - 10 # 10 isa
                vals = ['pupil colour',str(self.settings['pupilcol']), 'threshold',
str(self.settings['threshold']), 'pupil position', str(self.settings['pupilpos']), 'pupil
rect', str(self.settings['pupilrect'])]
                for i in range(len(vals)):
                    # draw title
                    tsize = self.sfont.size(vals[i])
                    tpos = vtx-tsize[0], starty+i*20
                    tsurf = self.sfont.render(vals[i], True, self.fgc)
                    self.disp.blit(tsurf,tpos)

                # draw new image
                if stagevars[0]['show_threshimg']:
                    self.disp.blit(self.thresholded, blitpos)
```

Continuación del apéndice 1.

```
                else:
                        self.disp.blit(self.img, blitpos)

                # update display
                pygame.display.flip()

                # apply settings
                self.tracker.settings = self.settings

                # DEBUG #
                if DEBUG:

        self.savefile.write(pygame.image.tostring(self.disp,'RGB')+BUFFSEP)
                # # # # #

            # DEBUG #
            if DEBUG:
                # close savefile
                self.savefile.close()
                # message
                print("processing images...")
                # open savefile
                savefile = open('data/savefile.txt','r')
                # read ALL contents in once, then close file again
                raw = savefile.read()
                savefile.close()
```

Continuación del apéndice 1.

```
                    # split based on newlines (this leaves one empty entry, because of the final newline)
                    raw = raw.split(BUFFSEP)
                    # process strings and save image
                    for framenr in range(len(raw)-1):
                        img = pygame.image.fromstring(raw[framenr],self.dispsize,'RGB')
                        pygame.image.save(img,'data/frame%d.png' % framenr)
                # # # # #

                return self.tracker


        def check_input(self):

                """Checks if there is any keyboard or mouse input, then returns
                input (keyname or clickposition) and inptype ('mouseclick' or
                'keypress') or None, None when no input is registered

                arguments
                None

                keyword arguments
                None

                returns
                inp, inptype   --       inp is a keyname (string) when a key has
```

Continuación del apéndice 1.

```
                                        been pressed, or a click position
                                        ((x,y) tuple) when a mouse button
has
                                        been pressed
                                    inptype is a string, either 'mouseclick',
                                        or 'keypress'
                                    if no input is registered, returnvalues
                                    will be None, None
        """

        # if nothing happens, None should be returned
        inp = None
        inptype = None

        # check events in queue
        for event in pygame.event.get():
            # mouseclicks
            if event.type == pygame.MOUSEBUTTONDOWN:
                inp = pygame.mouse.get_pos()
                inptype = 'mouseclick'
            # keypresses
            elif event.type == pygame.KEYDOWN:
                inp = pygame.key.name(event.key)
                inptype = 'keypress'

        return inp, inptype
```

Continuación del apéndice 1.

```
        def handle_input(self, inptype, inp, stage, stagevars):

                """Checks the input, compares this with what is possible in the
                current stage, then returns adjusted stage and adjusted stage
                variables

                arguments
                inptype         --      string indicating input type, should be
                                        either 'mouseclick' or 'keypress'
                inp             --      input, should be either
                """

                # # # # #
                # mouseclicks to keypress value

                if inptype == 'mouseclick':

                        # click position
                        pos = inp[:]

                        # loop through buttons
                        for bn in self.buttons.keys():
                                # check if click position is on a button
                                r = self.buttons[bn]['inactive']['rect']
                                if pos[0] > r[0] and pos[0] < r[0]+r[2] and pos[1] > r[1]
and pos[1] < r[1]+r[3]:
                                        # change input to button name
```

Continuación del apéndice 1.

```
                                inp = bn
                                # break from loop (we don't want to loop through all the other buttons once we've found the clicked one)
                                break

        # # # # #
        # keypress (or simulated keypress) handling

        # stage 1
        if stage == 1:
                # up should increase threshold, down should decrease threshold
                if inp in ['up','down']:
                        stagevars[1]['thresholdchange'] = inp

        # stage 2
        elif stage == 2:
                # up should increase pupil rect size, down should decrease pupil rect size
                if inp in ['up','down']:
                        stagevars[2]['vprectchange'] = inp
                elif inp in ['left','right']:
                        stagevars[2]['hprectchange'] = inp

        # stage 3
        elif stage == 3:
```

Continuación del apéndice 1.

```
                # up should increase threshold, down should decrease
threshold
                if inp in ['up','down']:
                        stagevars[1]['thresholdchange'] = inp
                # space should confirm settings
                if inp == 'space':
                        stagevars[3]['confirmed'] = True
                if inp == 'w':
                        self.array_up = self.settings['pupilpos']
                        print self.array_up
                if inp == 's':
                        self.array_down = self.settings['pupilpos']
                if inp == 'a':
                        self.array_left = self.settings['pupilpos']
                if inp == 'd':
                        self.array_right = self.settings['pupilpos']

        # space should move to next stage (but not in stage 3)
        if inp == 'space' and stage < 3:
                stage += 1

        # number keys should make the stage jump to that number
        if inp in ['1','2','3']:
                stage = int(inp)

        # T should toggle between displays
        if inp == 't':
```

Continuación del apéndice 1.

```
                    if stagevars[0]['show_threshimg']:
                        stagevars[0]['show_threshimg'] = False
                    else:
                        stagevars[0]['show_threshimg'] = True

                # R should toggle between using pupil rect or not
                if inp == 'r':
                    if stagevars[0]['use_prect']:
                        stagevars[0]['use_prect'] = False
                    else:
                        stagevars[0]['use_prect'] = True

                # escape should close down
                if inp == 'escape':
                    pygame.display.quit()
                    raise Exception("camtracker.Setup: Escape was pressed")

                # return the changed variables
                return stage, stagevars


        class CamEyeTracker:

          """The CamEyeTracker class uses your webcam as an eye tracker"""

          def __init__(self, device=None, camres=(640,480)):
```

Continuación del apéndice 1.

```
        """Initializes a CamEyeTracker instance

        arguments
        None

        keyword arguments
        device      --      a string or an integer, indicating either
                            device name (e.g. '/dev/video0'), or a
                            device number (e.g. 0); None can be passed
                            too, in this case Setup will autodetect a
                            useable device (default = None)
        camres      --      the resolution of the webcam, e.g.
                            (640,480) (default = (640,480))
        """

        global vcAvailable
        if vcAvailable == False:
                # select a device if none was selected
                if device == None:
                        available = available_devices()
                        if available == []:
                                raise Exception("Error in camtracker.CamEyeTracker.__init__: no available camera devices found (did you forget to plug it in?)")
                        else:
                                device = available[0]
```

Continuación del apéndice 1.

```
                # start the webcam
                self.cam = pygame.camera.Camera(device, camres, 'RGB')
                self.cam.start()
        else:
                self.cam = VideoCapture.Device()

        # get the webcam resolution (get_size not available on all systems)
        try:
                self.camres = self.cam.get_size()
        except:
                self.camres = camres

        # default settings
        self.settings = {'pupilcol':(0,0,0), \
                                'threshold':100, \
                                'nonthresholdcol':(100,100,255,255), \
                                'pupilpos':(-1,-1), \
                                'pupilrect':pygame.Rect(self.camres[0]/2-
50,self.camres[1]/2-25,100,50), \
                                'pupilbounds': [0,0,0,0], \
                                '':None
                                }


    def get_size(self):

        """Returns a (w,h) tuple of the image size
```

Continuación del apéndice 1.

```
        arguments
        None

        keyword arguments
        None

        returns
        imgsize         --      a (width,height) tuple indicating the size
                                of the images produced by the webcam
        """
        return self.camres


    def get_snapshot(self):

        """Returns a snapshot, without doing any any processing

        arguments
        None

        keyword arguments
        None

        returns
        snapshot        --      a pygame.surface.Surface instance,
                                containing a snapshot taken with the
webcam
```

Continuación del apéndice 1.

```
        """
        global vcAvailable
        if vcAvailable:
            image = self.cam.getImage()
            mode = image.mode
            size = image.size
            data = image.tostring()
            return pygame.image.fromstring(data, size, mode)
        else:
            return self.cam.get_image()


    def threshold_image(self, image):

        """Applies a threshold to an image and returns the thresholded
        image

        arguments
        image           --      the image that should be thresholded, a
                                pygame.surface.Surface instance

        returns
        thresholded     --      the thresholded image,
                                a pygame.surface.Surface instance
        """

        # surface to apply threshold to surface
```

Continuación del apéndice 1.

```
            thimg = pygame.surface.Surface(self.get_size(), 0, image)

            # perform thresholding
            th                                                              =
(self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
            pygame.transform.threshold(thimg,  image,  self.settings['pupilcol'],
th, self.settings['nonthresholdcol'], 1)

            return thimg


    def find_pupil(self, thresholded, pupilrect=False):

            """Get the pupil center, bounds, and size, based on the
thresholded
            image; please note that the pupil bounds and size are very
            arbitrary: they provide information on the pupil within the
            thresholded image, meaning that they would appear larger if the
            camera is placed closer towards a subject, even though the
            subject's pupil does not dilate

            arguments
            thresholded        --      a pygame.surface.Surface instance, as
                                       returned by threshold_image

            keyword arguments
```

Continuación del apéndice 1.

```
        pupilrect       --  a Boolean indicating whether pupil searching
                            rect should be applied or not
                            (default = False)

        returns
        pupilcenter, pupilsize, pupilbounds
                        --  pupilcenter is an (x,y) position tuple that
                            gives the pupil center with regards to the
                            image (where the top left is (0,0))
                            pupilsize is the amount of pixels that are
                            considered to be part of the pupil in the
                            thresholded image; when no pupilbounds can
                            be found, this will return (-1,-1)
                            pupilbounds is a (x,y,width,height) tuple,
                            specifying the size of the largest square
                            in which the pupil would fit
        """


        # cut out pupilrect (but only if pupil bounding rect option is on)
        if pupilrect:
            # pupil rect boundaries
            rectbounds = pygame.Rect(self.settings['pupilrect'])
            # correct rect edges that go beyond image boundaries
```

Continuación del apéndice 1.

```
                    if self.settings['pupilrect'].left < 0:
                        rectbounds.left = 0
                    if self.settings['pupilrect'].right > self.camres[0]:
                        rectbounds.right = self.camres[0]
                    if self.settings['pupilrect'].top < 0:
                        rectbounds.top = 0
                    if self.settings['pupilrect'].bottom > self.camres[1]:
                        rectbounds.bottom = self.camres[1]
                    # cut rect out of image
                    thresholded = thresholded.subsurface(rectbounds)
                    ox, oy = thresholded.get_offset()

            # find potential pupil areas based on threshold
            th = (self.settings['threshold'],self.settings['threshold'],self.settings['threshold'])
            mask = pygame.mask.from_threshold(thresholded, self.settings['pupilcol'], th)

            # get largest connected area within mask (which should be the pupil)
            pupil = mask.connected_component()

            # get pupil center
            pupilcenter = pupil.centroid()

            # if we can only look within a rect around the pupil, do so
            if pupilrect:
```

Continuación del apéndice 1.

```
                    # compensate for subsurface offset
                    pupilcenter = pupilcenter[0]+ox, pupilcenter[1]+oy
                    # check if the pupil position is within the rect
                    if (self.settings['pupilrect'].left < pupilcenter[0] < self.settings['pupilrect'].right) and (self.settings['pupilrect'].top < pupilcenter[1] < self.settings['pupilrect'].bottom):
                        # set new pupil and rect position
                        self.settings['pupilpos'] = pupilcenter
                        x = pupilcenter[0] - self.settings['pupilrect'][2]/2
                        y = pupilcenter[1] - self.settings['pupilrect'][3]/2
                        self.settings['pupilrect'] = pygame.Rect(x,y,self.settings['pupilrect'][2],self.settings['pupilrect'][3])
                    # if the pupil is outside of the rect, return missing
                    else:
                        self.settings['pupilpos'] = (-1,-1)
                else:
                    self.settings['pupilpos'] = pupilcenter

                # get pupil bounds (sometimes failes, hence try-except)
                try:
                    self.settings['pupilbounds'] = pupil.get_bounding_rects()[0]
                    # if we're using a pupil rect, compensate offset
                    if pupilrect:
                        self.settings['pupilbounds'].left += ox
                        self.settings['pupilbounds'].top += oy
                except:
                    # if it fails, we simply use the old rect
```

Continuación del apéndice 1.

```
                pass

        return self.settings['pupilpos'], pupil.count(),
self.settings['pupilbounds']


    def give_me_all(self, pupilrect=False):

        """Returns snapshot, thresholded image, pupil position, pupil area,
        and pupil bounds

        arguments
        None

        keyword arguments
        pupilrect -- a Boolean indicating whether pupil
searching
                     rect should be applied or not
                     (default = False)

        returns
        snapshot, thresholded, pupilcenter, pupilbounds, pupilsize
            snapshot -- a pygame.surface.Surface instance,
                        containing a snapshot taken with the
webcam
            thresholded -- the thresholded image,
                           a pygame.surface.Surface instance
```

Continuación del apéndice 1.

```
                pupilcenter   --    pupilcenter is an (x,y) position tuple that
                                    gives the pupil center with regards to
the
                                    image (where the top left is (0,0))
                pupilsize     --    pupilsize is the amount of pixels that are
                                    considered to be part of the pupil in the
                                    thresholded image; when no
pupilbounds can
                                    be found, this will return (-1,-1)
                pupilbounds   --    pupilbounds is a (x,y,width,height) tuple,
                                    specifying the size of the largest square
                                    in which the pupil would fit
            """

            img = self.get_snapshot()
            thimg = self.threshold_image(img)
            ppos, parea, pbounds = self.find_pupil(thimg, pupilrect)

            return img, thimg, ppos, parea, pbounds
        def close(self):

            """Shuts down connection to the webcam and closes logfile

            arguments
            None

            keyword arguments
```

Continuación del apéndice 1.

```
        None

        returns
        None
        """

        # close camera
        self.cam.stop()
```

Fuente: elaboración propia.