PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito parcial para obtener el Título de INGENIERO DE SISTEMAS

CREACIÓN DE UNA HERRAMIENTA QUE PERMITA MOVER EL CURSOR DE UN COMPUTADOR A PARTIR DEL MOVIMIENTO OCULAR, UTILIZANDO TÉCNICAS DE VISIÓN ARTIFICIAL

Por

Br. Francisco A. Justo T.
Tutor: Iñaki Aguirre Gil, Doctor

Abril 2009

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Creación de una herramienta que permita mover el cursor de un computador, a partir del movimiento ocular, utilizando técnicas de visión artificial

Br. Francisco A. Justo T.

Proyecto de Grado — Sistemas de Control, 81 páginas

Resumen: El estudio propuesto está dirigido a crear una herramienta que permite mover el cursor de un computador, mediante el movimiento ocular, utilizando técnicas de visión artificial. La importancia de esta investigación radica en mejorar la calidad de vida de pacientes con tetraplejia, proporcionándoles la posibilidad de interactuar con un computador, y así realizar diversas actividades como acceder a Internet, manejar una gran variedad de programas para distintas tareas y, en general, realizar cualquier operación que pueda efectuarse utilizando un ratón típico de un computador personal. La herramienta creada trae grandes beneficios a personas con capacidades reducidas, ofreciéndoles la oportunidad de interactuar con el mundo exterior y, de esta manera, elevar su autoestima proporcionándoles la máxima funcionalidad posible a pesar de la lesión que hayan sufrido. El estudio realizado afirma la gran utilidad que tienen las técnicas de visión artificial, en el desarrollo de herramientas que pueden mejorar la calidad de vida de un ser humano. Se destaca que el sistema desarrollado satisface los requerimientos planteados, lo cual constituye un éxito desde el punto de vista de su bajo costo y su gran potencial a réplica.

Palabras clave: Visión artificial, Cursor del computador, Tetraplejia

Este proyecto está dedicado con el amor y el afecto más profundo

a mi madre, Angélica Torres,

a mi padre, Rubén D. Justo,

y a mis hermanos Rubén, José y Rosangélica,

quienes me inspiraron y motivaron a lograr esta meta.

Los Adoro.

Índice

Índice iv
Índice de Tablas vi
Índice de Figuras vii
Agradecimientos ix
Capítulo 1 Introducción 1
1.1 Planteamiento y justificación del problema 2
1.2 Objetivos 3
1.2.1 Objetivo general 3
1.2.2 Objetivos específicos 3
1.3 Estructura del documento 3
Capítulo 2 Estado del arte 5
2.1 Antecedentes 5
2.2 El ojo humano 9
2.3 Visión Artificial 11
2.3.1 Formación de imágenes 12
2.3.2 Procesamiento de imágenes 13
2.3.2.1 Segmentación de imágenes 13
2.3.2.2 Morfología 15
2.3.2.3 Procesamiento de histograma 17
2.4 La cámara web como sensor 17
Capítulo 3 Diseño del prototipo 20
3.1 Creación del periférico alternativo 21
3.1.1 Modificaciones de la cámara 21

3.1.2 Iluminación de la imagen 25
3.1.3 Ensamblaje del periférico 27
3.2 Metodología de las funciones de visión artificial 28
3.2.1 Funciones de morfología binaria 28
3.2.2 Funciones de umbralización 30
3.2.3 Funciones de segmentación 31
3.2.4 Funciones de histogramas 32
3.2.5 Funciones para invertir y convertir imágenes en escala de grises 35
Capítulo 4 Implementación de la herramienta y análisis de resultados 36
4.1 Definición de funciones y procedimientos 37
4.1.1 Funciones y procedimientos de *OpenCV* 37
4.1.2 Funciones y procedimientos de API 45
4.1.3 Funciones y procedimientos creados 48
4.2 Tratamiento de la imagen para la obtención de parámetros óptimos 58
4.3 Proceso paso a paso de la ejecución de la herramienta 68
4.4 Representación del algoritmo mediante Redes de Petri 71
4.5 Resultados de las pruebas 77
4.6 Factibilidad económica de la herramienta 78
Capítulo 5 Conclusiones y recomendaciones 80
5.1 Conclusiones 80
5.2 Recomendaciones 81
Referencias bibliográficas 82
ANEXOS 85

Índice de Tablas

Tabla 4.1: Descripción de los estados de la Red de Petri 73
Tabla 4.2: Descripción de las transiciones de la Red de Petri 75
Tabla 4.3: Costos de herramientas similares 79

Índice de Figuras

Figura 2.1: Vista frontal del ojo..... 9
Figura 2.2: Sección del ojo humano.....10
Figura 2.3: Representación matricial de una imagen a escala de grises 13
Figura 2.4: Ejemplo de aplicación de la técnica segmentación dada una semilla 15
Figura 2.5: Ejemplo de proceso de erosión binaria.....16
Figura 2.6: Imagen en la cual se observa el proceso de dilatación binaria 17
Figura 3.1: Cámara Logitech QuickCam.....22
Figura 3.2: Apertura de la cámara.....23
Figura 3.3: Ubicación de lente de la cámara.....23
Figura 3.4: Ubicación del sensor de luz y lente que contiene filtro 23
Figura 3.5: Extracción del filtro de luz infrarroja 24
Figura 3.6: Colocación del negativo fotográfico y armado de la cámara.....24
Figura 3.7: Cámara antes y después de la modificación.....25
Figura 3.8: Materiales utilizados para la creación del dispositivo de iluminación 26
Figura 3.9: Esquema de conexión de los componentes del dispositivo de iluminación ..26
Figura 3.10: Periférico alternativo visual 27
Figura 4.1: Diagrama de componentes de las dependencias de funciones API Win32 ...47
Figura 4.2: Imagen original obtenida de la cámara dadas las modificaciones.....58
Figura 4.3: Imagen donde se observa la región a ser tratada.....58
Figura 4.4: Variación del parámetro up en algoritmo de segmentación 59
Figura 4.5: Imagen segmentada e imagen binaria de la zona de interés.....60
Figura 4.6: Morfología de la zona de interés variando el parámetro n.....61
Figura 4.7: Resultado al encontrar centro y pintar la cruz en imagen binaria 62

Figura 4.8: Resultado al pintar centroide con una cruz en imagen del ojo 62
Figura 4.9: Cuadro rojo que delimita la zona de movimiento 63
Figura 4.10: Ejemplo de la ventana de calibración capturando ojo abierto 63
Figura 4.11: Ejemplo de la ventana de calibración capturando ojo cerrado.....64
Figura 4.12: Selección de la región para hacer el procesamiento ojo abierto 65
Figura 4.13: Selección de la región para hacer el procesamiento ojo cerrado 65
Figura 4.14: Resta entre imágenes ojo cerrado y ojo abierto 66
Figura 4.15: Resta entre dos imágenes ojo cerrado 66
Figura 4.16: Variaciones del umbral en la binarización de la figura 4.14.....67
Figura 4.17: Imagen final de adquisición en tiempo real 68
Figura 4.18: Movimiento del ojo hacia la izquierda e indicación del sentido con una flecha amarilla.....69
Figura 4.19: Movimiento del ojo hacia la derecha e indicación del sentido con una flecha verde.....69
Figura 4.20: Movimiento del ojo hacia arriba e indicación del sentido con una flecha anaranjada.....70
Figura 4.21: Movimiento del ojo hacia abajo e indicación del sentido con una flecha azul 70
Figura 4.22: Cerrado del ojo e indicación de la ejecución del clic y el doble clic.....70
Figura 4.23: Representación del algoritmo mediante una Red de Petri 76

Agradecimientos

En estas líneas deseo agradecer de corazón a todos los que influyeron en este estudio y, de alguna manera, permitieron que se hiciera posible. Antes que todo doy gracias a Dios Todopoderoso, por permitirme el alcance de esta meta, bendiciéndome con vida y salud.

A mi madre Angélica Torres, por su apoyo incondicional y motivación. Por sus enseñanzas continuas, mamá te amo mucho.

A mi padre Rubén Darío Justo, quien por sus consejos, enseñanzas y ejemplo a seguir, me motivaron y aumentaron las ganas de seguir adelante, papá te amo mucho.

En especial a Liz Aranguren, por haber estado a mi lado en los momentos más difíciles y más agradables vinculados con el progreso de esta investigación y por su ayuda incondicional en la culminación de éste.

Al profesor Iñaki Aguirre por haberme dado el privilegio de llevar a cabo esta grata investigación, por brindarme sus conocimientos y guiarme en el transcurso de su realización.

A mis hermanos Rubén, José y Rosangélica, por motivarme día a día para lograr esta meta, por interesarse en el desarrollo de este estudio y creer en mí.

A mis amigos y colegas cultivados en el transcurso de mi carrera, Fernando, Maryuri y Victor, por hacer más agradable esta trayectoria.

Capítulo 1

Introducción

En la actualidad, el acceso al mundo computacional y virtual se ha hecho prácticamente indispensable para la ejecución de tareas del día a día. La tecnología informática ha alcanzado diferentes niveles creando consigo un universo de aplicaciones que permiten hacer procesos más eficientes, comunicaciones más sencillas y, entre tantos aspectos más, dan acceso al conocimiento y entretenimiento.

El manejo de un computador tradicional requiere para el uso de los dispositivos de acceso (como el teclado o el ratón) la capacidad física del movimiento de, por lo menos, una de las extremidades del ser humano, con lo cual se limita el acceso a personas tetrapléjicas. La *tetraplejia* es una enfermedad causada por daños en la médula espinal que impide la movilidad de las cuatros extremidades del ser humano, reduciendo la capacidad de la persona al dejarla inhabilitada para realizar actividades del día a día y ocasionando grandes traumas psicológicos a las personas que llegan a padecer de dicha enfermedad (Health System, 2008).

En este sentido, la disposición de una herramienta que permita controlar el cursor a través del movimiento ocular es, sin duda alguna, un gran avance tecnológico que permite ampliar el tipo de usuarios. Esto hace posible que con recursos relativamente económicos y sin la necesidad de someter al usuario a una intervención médica, cualquier persona discapacitada pueda interactuar con el computador y con esto tener acceso a tareas que le asegurarán un mejoramiento en su calidad de vida. Entre la variedad de

oportunidades que brinda la herramienta está la comunicación, entretenimiento, acceso a diversidad de información, estudio o trabajo, dependiendo de las condiciones del usuario.

1.1 Planteamiento y justificación del problema

En el transcurso de los años se han venido desarrollando diferentes herramientas que permiten a los tetrapléjicos realizar algunas actividades y, con esto, mejorar su calidad de vida y elevar su autoestima. Sin embargo, algunas de estas herramientas requieren intervención de médicos especialistas, tales como los casos explicados por Sample (2005) y Scott (2006), en los cuales los discapacitados deben someterse a *intervenciones quirúrgicas*. Por otro lado, se encontraron usos de la visión artificial para desarrollar sistemas que pueden ser utilizados mediante un ordenador, pero presentan ciertas desventajas como *costos elevados* y, en algunos casos, la necesidad de que el usuario tenga *control de movimiento* voluntario de su cabeza. Además, varios de los sistemas resultan complejos, ya que requieren de *dispositivos adicionales* para detectar el movimiento de los ojos.

En atención a lo antes expuesto surgió la inquietud por realizar un estudio utilizando técnicas de visión artificial para proporcionar a pacientes tetrapléjicos la posibilidad de realizar tareas computacionales, a través de imágenes del movimiento de sus ojos capturadas por medio de una cámara web. Lo que se pretende con dicho estudio es crear una herramienta computacional que permita, detectar cambios del movimiento del ojo para identificar las intenciones del usuario y, con esto, lograr mover el cursor en las diferentes direcciones deseadas.

La herramienta creada brindará a las personas con *tetraplejia* la oportunidad de interactuar con un computador para realizar diversas actividades como: acceder a Internet, manejar una gran variedad de programas para distintas tareas y cualquier

operación que pueda realizarse utilizando un ratón tradicional de un computador personal. Se hace notorio el gran beneficio que esto trae consigo, ya que les ofrecerá a personas discapacitadas la posibilidad de interactuar con el mundo exterior, ofreciéndoles la máxima funcionalidad posible a pesar de la lesión que hayan sufrido.

1.2 Objetivos

1.2.1 Objetivo general

Crear una herramienta que permita mover el cursor de un computador a partir del movimiento ocular, utilizando técnicas de visión artificial.

1.2.2 Objetivos específicos

- Realizar una revisión bibliográfica sobre visión artificial y estudios previos relacionados con dicha técnica.
- Estudiar el funcionamiento básico de un ratón tradicional para computadoras.
- Desarrollar la herramienta mediante un lenguaje de programación apropiado al ámbito de estudio.
- Probar la herramienta que permite mover el cursor de un computador utilizando técnicas de visión artificial.

1.3 Estructura del documento

En este documento se presentan cinco capítulos, incluyendo esta introducción. En el capítulo 2 se definen algunos términos relacionados con visión artificial y varias técnicas para el procesamiento de imágenes. El capítulo 3 explica la parte experimental inicial, es

decir el diseño del prototipo. Luego, en el capítulo 4 procedimiento de la implementación de la herramienta y, a su vez, el análisis de los resultados obtenidos. Y, por último, en el capítulo 5 se presentan las conclusiones y recomendaciones de la herramienta creada.

Capítulo 2

Estado del arte

Este capítulo contiene información sobre la investigación de productos similares al que se expone en este proyecto. Se incluyen algunos conceptos básicos asociados al estudio, como los relacionados con el ojo humano y sus partes, visión artificial y algunas técnicas de procesamiento de imágenes. Sumado a esto, se justifica la selección de la cámara como sensor.

2.1 Antecedentes

Antes de comenzar el estudio referente al uso de la visión artificial para mover el cursor de un computador mediante el movimiento ocular, resulta necesario realizar una investigación exhaustiva con el fin de conocer algunas publicaciones o desarrollos de productos similares que se han hecho en el mismo ámbito. Algunas de las investigaciones que se mencionan a continuación sirvieron de base para el desarrollo de la nueva herramienta.

Tchalenko (2000) encabezó un proyecto denominado *Drawing and Cognition*, del cual forma parte un sistema al cual llamaron *Eye Mouse*. El sistema fue creado con dos propósitos principales: el primero, que sirviera como base de investigación de la relación del movimiento ocular voluntario con el dibujo y, el segundo, ayudar a personas con discapacidad a controlar un ordenador mediante el movimiento ocular. El mencionado

Eye Mouse consiste en una cámara y dos sensores infrarrojos atados al monitor de un computador personal para grabar los movimientos del ojo. Luego, un programa se encarga de procesar la información y lograr mover el cursor, hacer clic o doble clic, según sea el deseo del usuario. De acuerdo a la Real Academia Española (2001) un clic es la pulsación que se hace en alguno de los botones del ratón de un ordenador. Este sistema fue desarrollado en la Universidad de Camberwell; actualmente tiene un costo de aproximadamente 2800€.

Ward y MacKay (2002), de la Universidad de Cambridge, publicaron en la revista *Nature* un artículo sobre el desarrollo de un sistema para ordenador que escribe siguiendo las indicaciones de los ojos. Dichos autores mencionan que mediante un ordenador, dotado de una cámara especializada, interpreta el lenguaje de los ojos y es capaz de componer un texto a partir de las indicaciones de la mirada. Esta técnica requiere el uso de dos emisores de infrarrojos, los cuales se reflejan en las pupilas y, luego, los rayos reflejados son observados por la cámara para interpretar la información. En el caso del proyecto realizado se requirió únicamente de una videocámara para la adquisición de imágenes de los movimientos del ojo con algunas pequeñas adaptaciones. Además, es importante destacar que, de acuerdo a algunos médicos especialistas, la luz infrarroja no causa daños en la vista de las personas.

El Sistema "*I4Control*" (2004) creado en el Departamento de Cibernética de la Universidad Técnica Checa, permite mediante un periférico computacional, controlar el cursor de un ratón tradicional mediante el movimiento de los ojos o la cabeza. La esencia del sistema es una mini cámara en blanco y negro que es colocada en una montura de lentes para observar el punto de interés del usuario. Luego, de acuerdo a la posición del ojo, el cursor se mueve hacia la dirección deseada o se detiene. El clic y el doble clic se logran manteniendo el ojo cerrado por tiempos previamente establecidos. Este sistema tiene como ventajas que es de fácil instalación y manipulación. Sin embargo, posee grandes desventajas como su alto costo de aproximadamente 1600 Euros y, además,

requiere la instalación de una unidad de control, la cual es un periférico especial que hace posible su uso.

Sample (2005) explica el uso de un *circuito integrado* en el cerebro que permite mover un brazo robótico para pacientes tetrapléjicos. Sin embargo, para ayudar a personas discapacitadas a ser más independientes mediante este tipo de implante cerebral, el paciente debe someterse a una operación en la cual se le colocan electrodos sobre la corteza motora sensorial. Con esto, los pacientes se ven expuestos a dolores y riesgos asociados con intervenciones quirúrgicas.

Scott (2006) en su artículo *Neuroscience: Converting thoughts into actions,* de la revista *Nature,* comenta sobre los avances de la ciencia respecto a tecnología implantada en cerebros de parapléjicos con el fin de permitirles comunicarse e interactuar con el mundo exterior. En dicho artículo se comenta sobre cómo, a través de una prótesis cerebral, un paciente tetrapléjico logra con el pensamiento mover el cursor de un ordenador, subir el volumen a un televisor, revisar su correo electrónico, entre otras cosas. Este antecedente reafirma el hecho de que varios estudios previos con el mismo fin al que se presenta, necesitaban de la participación de médicos especialistas y, con esto, altos costos y la exigencia de que los pacientes se sometan a intervenciones quirúrgicas.

Palleja y Rubion (2007) del grupo de robótica de la Universidad de Lleida crearon una herramienta para personas con discapacidades motrices, la cual dieron a conocer con el nombre de *HeadMouse*. Dicha herramienta permite controlar el ratón con la cabeza mediante dos alternativas que se diferencian en el tipo de movimiento que debe tener la misma. La primera de ellas, conocida como *HeadMouse* 1, se refiere al control del ratón a través de movimientos relativos de la cabeza. La segunda, conocida como *HeadMouse* 2, es aquella en donde el movimiento del ratón sigue los movimientos absolutos de la cabeza. Para ejecutar la aplicación se necesita cómo mínimo una computadora Pentium IV con procesador 2GHz con sistema operativo Windows 98 o superior y, además, la disposición de una cámara *web* que soporte la resolución de 640x480. Esta herramienta

tiene grandes ventajas entre las cuales se destacan principalmente la disponibilidad para descargarla de forma gratuita a través de Internet y los requerimientos básicos para poder utilizarla. Sin embargo, tiene la limitante de que el usuario debe estar en capacidad de mover su cabeza para poder hacer uso de dicha herramienta.

La Asociación de Esclerosis Lateral Amiotrófica (2007) con el apoyo de la Fundación Caixa y la ayuda de un experto, logró desarrollar un sistema denominado Iriscom, con la finalidad de brindar a personas que no poseen control sobre ninguna parte de su cuerpo, la oportunidad de comunicarse a través del iris. Este producto utiliza una tecnología denominada videoculografía, la cual permite al usuario mover el puntero del ratón mediante el movimiento ocular. Dicho sistema se compone de una cámara y dos emisores de luz infrarroja que son acoplados a un ordenador personal. La cámara toma la imagen del ojo del usuario y el reflejo que los dos emisores provocan sobre el iris. Luego, una aplicación específica interpreta esta imagen, calcula hacia donde se encuentra mirando el ojo del usuario y convierte esa posición en una coordenada para el ratón. El Iriscom se puede conseguir en dos versiones QG2 y QG3, con valores de 6800€ y 7800€, respectivamente. La diferencia entre ambos radica en que el QG3 es más compacto, lo que lo hace más fácil de trasportar. Los requisitos mínimos que se necesitan son: un procesador Pentium IV, Intel Centrino o similar; al menos un puerto IEEE 1394 disponible; Windows XP o superior; 512 Mb RAM y al menos 10 MB de espacio de disco disponible para la instalación.

Gips y Betke (2009) son los responsables de la creación de la tecnología desarrollada conocida como *Camera Mouse*, la cual utiliza una cámara USB comercial para reemplazar al ratón de una computadora personal. De manera general, la *Camera Mouse* funciona enfocando la cámara a la cara del usuario y, luego de hacer clic sobre un punto específico que puede ser la punta de la nariz o alguno de los extremos de las cejas, el cursor se mueve coordinadamente con la cabeza del mismo. La primera versión gratuita de esta herramienta fue creada en el año 2007 pero luego han publicado dos versiones

mejoradas, una en el 2008 y la última en Febrero del 2009. Las mejoras en las versiones nuevas tienen relación con el tipo de cámaras compatibles y los sistemas operativos que soporta el programa. A pesar de que dicho programa ha sido un éxito para ayudar a personas con capacidades reducidas, requiere que el usuario tenga movimiento voluntario de su cabeza.

2.2 El ojo humano

El ojo humano es un órgano fotorreceptor que detecta la luz, cuya función es recibir los rayos luminosos que provienen de los objetos presentes en el mundo exterior, para luego ser transformados en impulsos nerviosos y dirigirlos al centro nervioso de la visión en el cerebro (Asociación Larense de Astronomía, 2009). La vista es uno de los cinco sentidos que permiten al ser humano comprender el mundo que lo rodea, el ojo es la base de este sentido. El globo ocular mide aproximadamente unos 2.5 cm. de diámetro y está recubierto por una membrana compuesta por varias capas. En la figura 2.1 se puede observar la imagen de un ojo con sus partes básicas visibles.

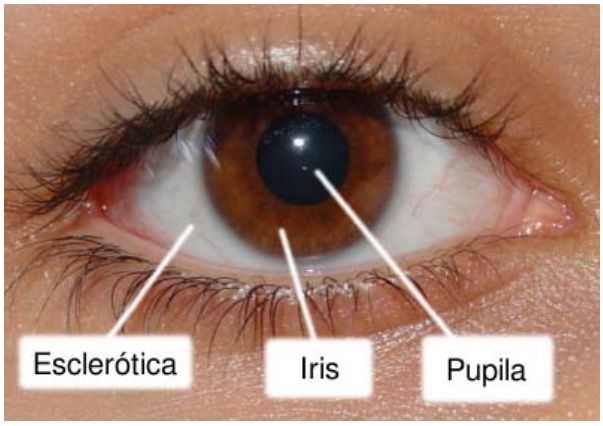

Figura 2.1: Vista frontal del ojo

La zona óptica del ojo está formada por la córnea, el iris, la pupila y el cristalino. En la figura 2.2 se observan estas partes, así como también el resto que lo compone. Por otro lado, la córnea es un material transparente que permite el paso de la luz hacia el interior del ojo, el cual sirve de protección al iris y al cristalino debido a que posee propiedades ópticas de refracción y funciona como una lente fija. El iris es una membrana coloreada y circular que posee una apertura interior variable de color negro que es denominada pupila, la cual comunica la cámara anterior del ojo con la posterior. El iris está constantemente activo para permitir que la pupila se dilate (midriasis) o se contraiga (miosis), esto con la finalidad de controlar la cantidad de luz que llega a la retina. A su vez, el cristalino está situado delante del humor vítreo y detrás del iris, tiene forma de lente biconvexa, se encarga de refractar la luz para poder proyectarla en la retina. Su función principal es permitir enfocar objetos que se encuentran a diferentes distancias (Turégano, 2006).

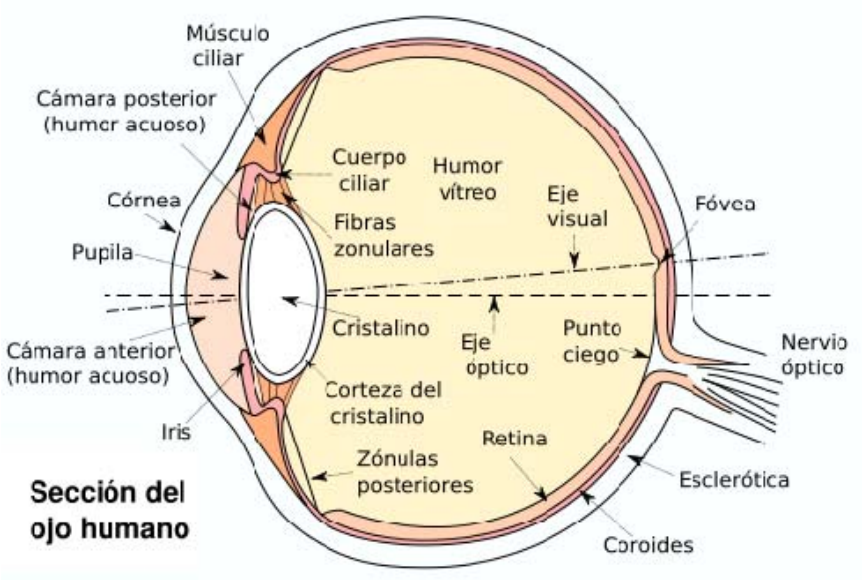

Figura 2.2: Sección del ojo humano

El humor vítreo es un líquido transparente y gelatinoso situado en la cámara posterior del ojo, entre el cristalino y la retina, el cual está formado por un 99.98% de agua y el resto de sales minerales. El humor acuoso es un líquido menos denso que el humor vítreo que fluye por la cámara posterior y la cámara anterior, permitiendo que los nutrientes circulen y que la presión de este líquido mantenga a la córnea convexa.

La retina se encuentra en la parte trasera del glóbulo ocular y es la que contiene los receptores sensibles a la luz, denominados fotorreceptores. Los fotorreceptores convierten la luz en impulsos eléctricos, que van a los centros encargados de la visión en el cerebro. Hay dos tipos de fotorreceptores denominados *conos* y *bastones*. Los *conos* se encuentran ubicados en una zona denominada fóvea, la cual es la parte central de la retina y la de menor tamaño. Los *conos* son sensibles a la luz de color o visión diurna y esta zona tiene mayor sensibilidad a la longitud de onda electromagnética. Por otro lado, los *bastones* se encuentran en la zona de la retina denominada mácula, la cual es de mayor extensión y la agudeza visual es mejor que en la fóvea. Los *bastones* son sensibles a la oscuridad y a la luz sin color o visión nocturna. La retina contiene aproximadamente 120 millones de bastones y 6 millones de conos (Platero, 2005).

2.3 Visión Artificial

La visión artificial es un campo de la inteligencia artificial que ha estado ganando popularidad en los últimos años debido a sus tantas características atractivas, entre las cuales se destaca que permite optimizar procesos donde se requiere el uso del ojo humano, mejorando la calidad y la velocidad de los procesos. Adicionalmente, la visión artificial tiene una alta velocidad de respuesta y hace posible mediciones sin contacto en tiempo real. Otro de los factores que hace tan atractivas las técnicas de visión artificial es que pueden ser aplicadas con éxito en numerosos campos, entre los cuales se encuentran las industrias automotrices, de componentes eléctricos, de productos farmacéuticos y, en

general, en cualquier proceso industrial donde se requiere del ojo humano para tareas específicas comunes como lo es el control de calidad.

De acuerdo a Platero (2005), la visión artificial o visión por computador pretende capturar la información visual del entorno físico para extraer características relevantes visuales, utilizando procedimientos automáticos. Asimismo, dicho autor afirma que los dos pilares del sistema físico de visión artificial son: el *sistema de formación de las imágenes* y el *sistema de procesamiento* de éstas.

2.3.1 Formación de imágenes

La formación de imágenes es la primera etapa de la visión artificial, constituido por la captación de la imagen de interés, su iluminación y la adquisición de la señal en el computador. El recién mencionado autor sostiene que los elementos más críticos en la calidad de la imagen y por ende en la métrica son el subsistema de iluminación, la óptica y la selección de arquitecturas de cámaras de estado sólido.

La representación de una imagen en el computador es una matriz cuyos valores corresponden a la intensidad de color de cada píxel de la imagen capturada. Cuando se trabaja con imágenes a color, se obtiene una matriz de matrices, donde las matrices internas representan los tres colores básicos reconocidos por un computador que son rojo, verde y azul. Por otro lado, cuando se trabaja con imágenes en escala de grises, el tratamiento resulta mucho más fácil ya que lo que se obtiene es una sola matriz sencilla con valores internos que oscilan entre 0 y 255, dependiendo de la intensidad del gris de cada píxel. En la figura 2.3 se puede observar un ejemplo de la captación de una imagen de la cual se toma una matriz 5x5 para ilustrar su representación computacional.

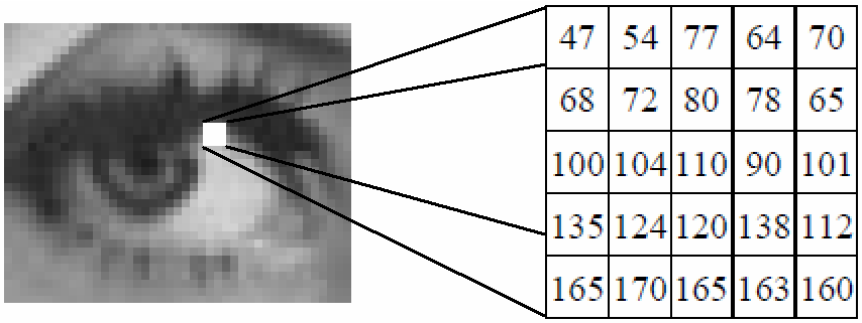

Figura 2.3: Representación matricial de una imagen a escala de grises

2.3.2 Procesamiento de imágenes

Luego de haber obtenido la imagen a través de la cámara se debe proceder a realizar el procesamiento de la misma, de tal forma que ésta pueda ser utilizada para obtener áreas con significado, las cuales son relativas al estudio que se realiza. Las técnicas utilizadas para el procesamiento de las imágenes se describen a continuación.

2.3.2.1 Segmentación de imágenes

La primera de las técnicas utilizadas es la segmentación, la cual permite particionar la imagen en áreas con significado, en donde dicho particionamiento depende del problema a resolver y debe detenerse cuando los objetos de interés de la imagen hayan sido aislados. La segmentación autónoma es una de las tareas más complicadas en el procesamiento de imágenes ya que dicha etapa del proceso determinará el éxito o fracaso del análisis, por esta causa se debe considerar aumentar la probabilidad de tener una segmentación robusta, en muchos casos para que esto se cumpla es recomendable poder manipular el entorno de dicha imagen.

La segmentación dada una semilla es un caso particular de la segmentación orientada a regiones, en la cual la estrategia a emplear está basada en los criterios de

similitud y continuidad de los píxeles que forman una región. Además de esto, Platero (2005) afirma que, bajo dicha perspectiva, la imagen se considera formada por n regiones disjuntas, cada una de las cuales tiene agrupada a los píxeles por alguna propiedad que los hace ser característicos de esa zona y discrepantes respecto al resto. Desde el punto de vista formal, las condiciones de la segmentación orientada a regiones serían:

a) $I = \bigcup_{i=1}^{n} R_i$ b) $R_i \cap R_j = \phi \quad i \neq j$

c) $p_i \in R_j \leftrightarrow P_{R_j}(p_i) = 1$ d) $p_i \notin R_j \leftrightarrow P_{R_j}(p_i) = 0$

en donde *I* es la imagen, R_i es una región de dicha imagen, pi un píxel cualquiera de la misma y P_{Rj} la regla de similitud de la región j. Las dos primeras condiciones describen la segmentación como un proceso de partición de la imagen en regiones disjuntas; las otras dos restantes hacen referencia a las propiedades de similitud de los píxeles agrupados y de discrepancia respecto al resto.

Por otra parte, la segmentación por crecimiento de regiones o segmentación dada una semilla consiste en que, a partir de píxeles semillas, se hace crecer la región tomando en cuenta alguna regla de similitud y considerando la propiedad de conectividad de los píxeles. Cuando ya se tiene la semilla definida, se analizan los píxeles vecinos según la regla de similitud de la región, luego los píxeles que cumplan la propiedad se añadirán a la región de crecimiento, los que no cumplan dicha propiedad pertenecerán a otra región. Posteriormente, con los nuevos píxeles unidos a la región de crecimiento, se volverán a procesar sus nuevos vecinos, este algoritmo llegará a su fin cuando todos los vecinos hayan sido estudiados. En la figura 2.4 se puede observar una imagen en la cual se aplica la técnica de segmentación dada diferentes semillas y las respectivas regiones resultantes.

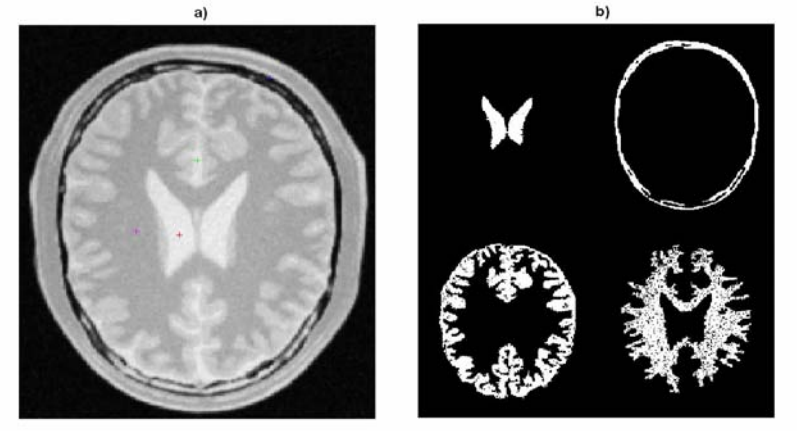

Figura 2.4: Ejemplo de aplicación de la técnica segmentación dada una semilla

Otra de las técnicas de segmentación es la umbralización, la cual se utiliza tanto para binarizar una imagen como para optimizarla. En líneas generales la umbralización binaria, consiste en transformar una imagen que está en escala de grises a una imagen en blanco y negro. Específicamente, cuando se trabaja con imágenes en escala de grises, se manipulan valores entre 0 y 255, donde cada valor intermedio indica una tonalidad diferente de gris, el 0 representa el negro y el 255 al blanco. De acuerdo al interés del procesamiento de la imagen, se asigna una condición para manipular la misma. Un ejemplo de esto sería establecer el umbral en un valor *x* y luego especificar que todos los píxeles con valores menores o iguales a *x* se conviertan en negro (0) y los mayores se conviertan en blanco (255).

2.3.2.2 Morfología

Seguido de la aplicación de la segmentación en muchas ocasiones resulta necesario el uso de otras técnicas como la de morfología. Esto se debe principalmente a que la primera de las técnicas mencionada no suele dar una imagen exacta de los límites de las regiones seleccionadas. La morfología, basada en la teoría de conjuntos, permite reconstruir parcialmente la imagen haciéndola mucho más precisa, ya que realza la geometría y la

forma de las imágenes. De manera general, para llevar a cabo la morfología de un determinado objeto es necesario hacer una erosión y dilatación del mismo. De acuerdo a Platero (2005) la erosión es una transformación antiextensiva cuya utilidad consiste en definir una geometría determinada al elemento estructurante y pasarlo sobre la imagen, en donde los objetos menores al elemento estructurante no aparecerán en la imagen resultante. Asimismo, dicho autor explica que los objetos que queden de la transformación habrán sido degradados y, por tanto, la erosión supone una degradación de la imagen. Es así que la aplicación iterativa de dicha transformación hace que se eliminen todos los objetos existentes en la imagen. En la figura 2.5 se puede observar cómo cambia una imagen mediante la binarización y erosión.

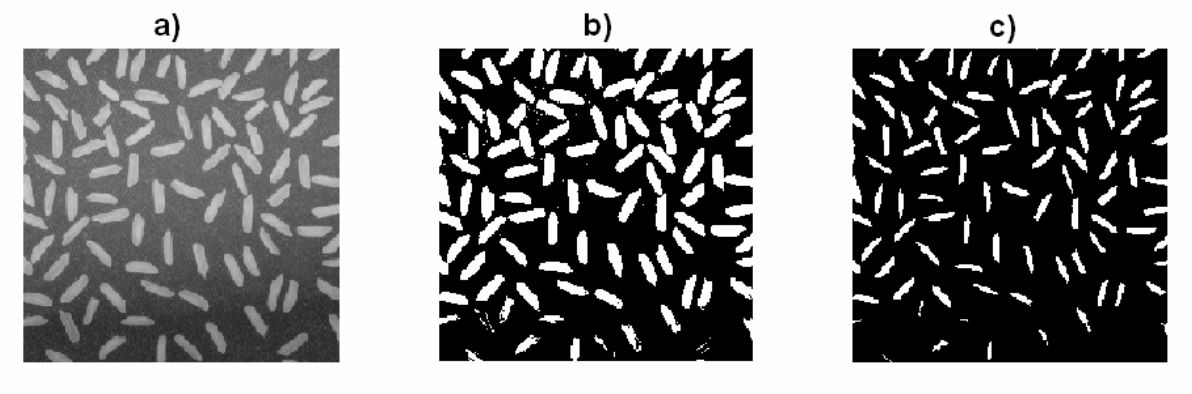

Figura 2.5: Ejemplo de proceso de erosión binaria

Por otro lado, la dilatación es la transformación dual a la erosión. Platero (2005) afirma que esta técnica se interpreta como el valor máximo del entorno de vecindad definido por el elemento estructurante. El autor aclara que las aplicaciones de las operaciones de erosión seguida con una dilatación no son conmutativas, ya que los resultados son diferentes dando paso a las aperturas y cierres morfológicos. En la figura 2.6 se representa una imagen a la cual se le aplica binarización y luego dilatación.

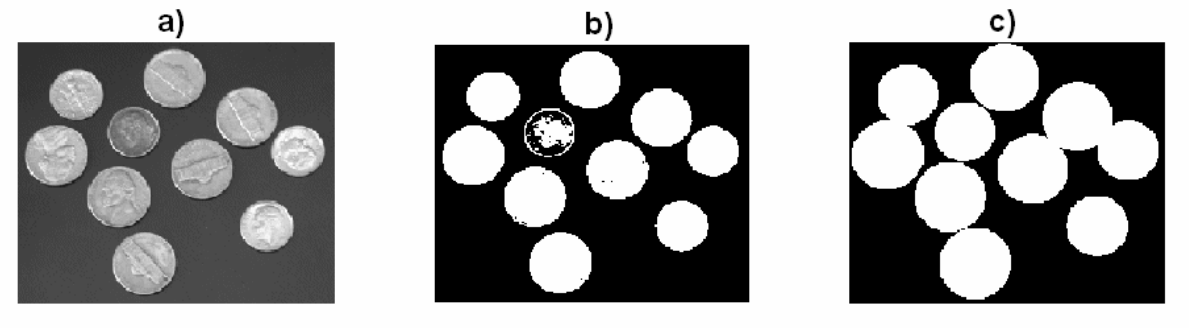

Figura 2.6: Imagen en la cual se observa el proceso de dilatación binaria

2.3.2.3 Procesamiento de histograma

El procesamiento de histogramas constituye una técnica que permite mejorar la calidad de una imagen. Existen diversos métodos para manipular los histogramas, entre los cuales se encuentra la ecualización y la especificación del histograma. De manera general, en ambos métodos los pixeles se modifican utilizando una función de transformación que se basa en la distribución de los niveles de gris en toda la imagen. Los histogramas reflejan información estadística importante que pueden ser utilizados en procesamientos de imágenes en tiempo real. Otra de las utilidades que brindan los histogramas es que pueden ser usados para hacer comparaciones entre ellos mismos, lo que representaría comparaciones entre imágenes (González y Woods, 1996).

2.4 La cámara web como sensor

Según Turégano (2006) existen cuatro técnicas básicas de medición de movimiento ocular, a saber: a) electro-oculografía (EOG); b) lentes de contacto; c) foto o video oculografía; y, d) detección por video basada en la pupila y la reflexión córnea. Cada una de las técnicas mencionadas tiene sus particularidades.

La electro-oculografía es una técnica para medir el potencial de reposo de la retina, fue una de las técnicas de detección de movimientos oculares más popular hace más de 40 años. Para medir los movimientos del ojo esta técnica normalmente usa cuatro electrodos, colocados a los lados y por arriba y abajo del ojo, los cuales captan si el ojo se mueve, por ejemplo del centro a la posición del un electrodo. En consecuencia de esto, se produce una diferencia de potencial con la cual se logra medir la posición. Como esta técnica mide la posición relativa del ojo respecto de la cabeza, no vale para medir la posición a la que el ojo esta mirando.

La *técnica de lentes de contacto* es, según Turégano (2006), una de las más precisas para medir el movimiento del ojo, y afirma que consiste en una lente de contacto con un pequeño hilo de metal en ella. Dicho hilo está enrollado en una vuelta y mide las variaciones de campo magnético. Este método mide la posición de los ojos respecto a la cabeza y no es útil para medir el punto de interés, además resulta tedioso para el usuario ya que tiene que colocar un objeto extraño en la zona ocular.

Por otro lado, la *video-oculografía* o *foto-oculografía* representa una gran variedad de técnicas bajo esta denominación, las cuales se encargan de extraer características de los ojos. Estas características son de rotación y traslación como lo son: la forma de la pupila, la unión entre el iris y la esclerótica, las reflexiones córneas, etc. A pesar de tantas formas que se encargan de hacer esta medición, ninguna proporciona una medición del punto de interés (Turégano, 2006).

Por último, la *técnica de detección por video basada en la pupila* y la *reflexión córnea*, a diferencia de las tres técnicas explicadas anteriormente, sí permite medir el punto de interés, es decir el lugar exacto a donde el ojo está mirando. Turégano (2006) explica que para poder medir hacia dónde esta mirando el ojo se necesita que la cabeza se encuentre quieta, de manera que la posición del ojo pueda coincidir con el punto de interés. Así, existen dos tipos de seguidores basados en luz infrarroja: los de pupila oscura y los de pupila brillante; cuando el haz de luz que ilumina el ojo es coaxial al

camino de visión, el ojo actúa como retroreflector produciendo una pupila brillante en la captura de la misma. Por otra parte, si existe un desplazamiento entre la luz y el camino de visión, la pupila será capturada con un color muy oscuro generando una pupila oscura. De acuerdo a todas estas características, se decidió utilizar la técnica de detección por video basada en la pupila y la reflexión córnea para la creación de la herramienta que se presenta.

Capítulo 3

Diseño del prototipo

Luego de revisar las técnicas y procedimientos que se deben utilizar para la creación de la herramienta de ayuda a discapacitados, se procede a diseñar y elaborar el prototipo. El diseño y la implantación de un prototipo es parte fundamental de la creación de un nuevo producto, ya que éste es el que hace posible realizar todas las pruebas necesarias para determinar la eficacia del mismo.

En el caso de la herramienta planteada se utiliza una cámara de video que permita capturar las imágenes en tiempo real. Respecto a la iluminación en la zona de la imagen, se recurre a un LED (*Light-Emitting Diode*) emisor de infrarrojo, ya que este tipo de luz no es percibida por el ojo humano, por lo que se evita cualquier molestia como dilatación de la pupila o encandilamiento. Adicionalmente, resulta necesaria la aplicación de diversas técnicas para lograr obtener una imagen del ojo con áreas significativas que pudiesen ser tratadas con el fin de lograr el movimiento del cursor, en la media que dichas áreas cambian de acuerdo a ciertos parámetros.

Por otro lado, para el desarrollo de este proyecto se parte de la premisa de lo expuesto por Turégano (2006), donde explica que la técnica de segmentación dado una semilla, permite detectar la zona de interés y, además, mediante la manipulación del entorno se lograría que dicha segmentación tuviese un alto grado de robustez. A continuación se explican los pasos realizados necesarios para lograr la construcción de la herramienta de ayuda a discapacitados deseada.

3.1 Creación del periférico alternativo

Una de las partes indispensables en el desarrollo del proyecto que se presenta es la creación de un periférico alternativo, el cual sustituye al ratón tradicional de un computador personal. La característica fundamental del dispositivo es que debe estar ubicado de manera que se pudiese capturar la zona de interés, es decir, la región del ojo del usuario, de manera precisa. La necesidad de enfocar únicamente la zona el ojo del usuario y no una mayor región se debe a la utilización de una cámara comercial económica de relativa baja resolución (1.3 mega píxel). A continuación se explican los pasos seguidos para la creación del periférico visual alternativo para el desarrollo de la herramienta planteada.

3.1.1 Modificaciones de la cámara

Bajo el criterio previamente establecido que se utiliza la técnica de detección por video basada en la pupila y la reflexión córnea para medir el movimiento ocular, el primer paso necesario para la creación del prototipo es el de hacer una modificación a la cámara para que pueda captar la luz infrarroja. Esto se debe a que todas las cámaras digitales comunes disponen de un sensor, normalmente un CMOS (del inglés *Complementary Metal Oxide Semiconductor*), los cuales son sensibles a toda la luz visible tanto para el ojo humano como a la luz infrarroja. Ahora, para evitar que la luz infrarroja sature los colores y genere una imagen irreal, dichos sensores están provistos de un filtro, de manera que sólo pueda pasar la luz visible. En este sentido, la modificación consiste en retirar el filtro que viene con la cámara y colocar en su lugar un filtro infrarrojo que bloquea la luz visible y deja pasar solo la infrarroja, logrando con esto transformar la cámara en infrarroja.

Actualmente, existen una serie de filtros comerciales como los de Lee, Cokin, entre otros, los cuales podrían ofrecer una mejor calidad en la imagen (Turégano, 2006). Sin embargo, dichos filtros tienen costos que elevarían considerablemente el valor del producto final, lo que desviaría la idea inicial de crear una herramienta económica que lograra el objetivo deseado. Tomando esto en consideración, se utiliza un negativo fotográfico velado, ya que brinda buena calidad de la imagen, y además, puede conseguirse fácilmente sin costo alguno.

A continuación se explican los pasos detallados para hacer la modificación a la cámara y transformarla en infrarroja. Para la explicación se usará una cámara Logitech QuickCam Express como la que se puede observar en la figura 3.1. No obstante, es importante destacar que en el proyecto también se utiliza una VideoCAM Look (Genius) muy similar a la Logitech, una Perfect Image Webcam (General Electric) y una Genius Eye 312.

Figura 3.1: Cámara Logitech QuickCam

1- El primer paso es abrir la cámara utilizando un destornillador adecuado.

Figura 3.2: Apertura de la cámara

2- Luego se retira la parte frontal y se ubica el lente de la cámara.

Figura 3.3: Ubicación de lente de la cámara

3- Se observa la ubicación del sensor de luz y se procede a retirar el lente donde se encuentra ubicado el filtro de luz infrarrojo.

Figura 3.4: Ubicación del sensor de luz y lente que contiene filtro

4- Dentro del lente se encuentra ubicado el filtro, por lo que se procede a retirar dicho lente para extraer el filtro de luz infrarroja.

Figura 3.5: Extracción del filtro de luz infrarroja

5- Posteriormente, en el lugar del cristal (filtro infrarrojo), se coloca un trozo, de un negativo de fotografía velado del mismo tamaño del filtro retirado para luego armar la cámara. Una vez realizado todo esto, se obtiene la cámara infrarrojo que se utilizará para la obtención de la imagen del ojo del usuario del sistema.

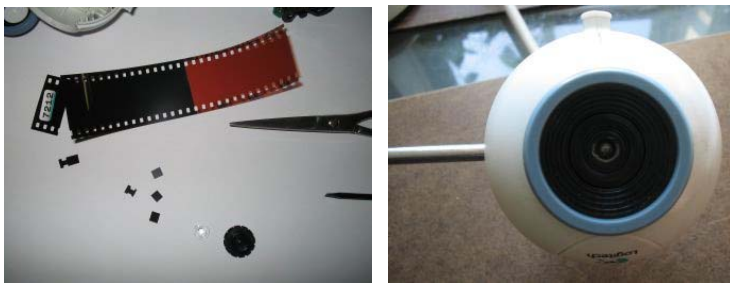

Figura 3.6: Colocación del negativo fotográfico y armado de la cámara

Figura 3.7: Cámara antes y después de la modificación

3.1.2 Iluminación de la imagen

Como se mencionó anteriormente, la iluminación es uno de los elementos más críticos en la calidad de la imagen a tratar. En el comienzo de la construcción del dispositivo para iluminar la zona del ojo, se utiliza un LED emisor de infrarrojos, una base para pilas AA, un trozo de cable y algo de cinta pegante. Todo esto, junto con las pilas correspondientes, se colocan junto a la cámara, pero resulta ser muy pesado. Debido a esto, se decide proporcionar la energía al LED a través de la computadora utilizando un cable USB (*Universal Serial Bus*). Sin embargo, dado que el LED funciona con un voltaje de máximo de 2 voltios y el puerto USB proporciona 5 Voltios, resulta necesario añadir una resistencia, de manera que el LED funcione correctamente sin quemarse. Para lograr la iluminación adecuada de la zona de interés a través de la fuente USB se utilizan entonces: un LED emisor de infrarrojos, un mini interruptor, una resistencia de 68 ohmios, una cable USB, cinta adhesiva para cable, un cautín y estaño. En la figura 3.8 se observan todos los materiales y el equipo utilizado.

Figura 3.8: Materiales utilizados para la creación del dispositivo de iluminación

El valor de la resistencia se determina usando la ley de Ohm (V = R . I) y el montaje se realiza según figura 3.9. En esta figura se observa un cable rojo que representa el voltaje positivo, el cual está unido a la pata positiva del led y, a su vez, el cable negativo que está en serie con la resistencia y el interruptor. Es importante resaltar que este montaje es sencillo y de bajo costo.

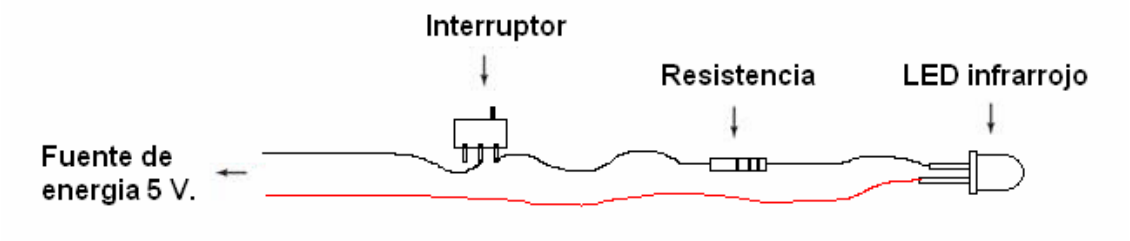

Figura 3.9: Esquema de conexión de los componentes del dispositivo de iluminación

3.1.3 Ensamblaje del periférico

Luego de realizar las modificaciones a la cámara y de crear el dispositivo para la iluminación de la imagen, se procede a ensamblar el periférico que se utilizará para el desarrollo de la herramienta que se presenta en este trabajo. Tomando en cuenta que la cámara debe posicionarse de manera que enfoque la región del ojo del usuario y, a su vez, se mantuviese fija a pesar del movimiento que pudiese tener el mismo, se decide utilizar una gorra para sujetar los dispositivos mencionados. En líneas generales, se pretende fijar la cámara y el componente de iluminación a la visera de la gorra utilizando un tornillo con una tuerca, de manera que la cámara quede enfocando la zona de interés. Adicionalmente, se añade alambre al cable de alimentación USB para poder direccionar el LED y, con esto, lograr iluminar adecuadamente la región del ojo. En la siguiente figura se observa el producto final obtenido.

Figura 3.10: Periférico alternativo visual

3.2 Metodología de las funciones de visión artificial

Al finalizar la propuesta de construcción de la parte física de la herramienta que permite mover el cursor del computador por medio del movimiento ocular, se plantea continuar con el desarrollo del *software* que le da funcionalidad al periférico diseñado. Tomando en cuenta que la aplicación requiere procesamientos en tiempo real para su ejecución, de manera que el cursor se mueva coordinadamente con el movimiento ocular del usuario, con los mejores tiempos de respuesta posible, resulta necesario utilizar un lenguaje de programación de bajo nivel. Investigando se encontró una librería llamada *OpenCV* (*Open Source Compunter Vision Library*), la cual tiene implementada más de 500 funciones que abarcan muchas áreas de la visión y está especialmente diseñada para el tratado de imágenes en tiempo real (Bradski y Kaehler, 2008). De esta manera se determina la utilización de *OpenCV* para el desarrollo del *software* y, con ella, el lenguaje de programación C++. La librería mencionada fue creada por Intel para que fuese de acceso libre y, además, multiplataforma (compatible con Mac Os, Linux y Windows). Una de las ventajas que prevaleció en la decisión de su utilización, es que los tiempos de ejecución de las funciones de visión artificial utilizadas en el proyecto resultaron cumplir con las especificaciones de diseño (latencia pequeña, menos de 130 milisegundos). A continuación se explican las funciones que se utilizan en el procesamiento de las imágenes.

3.2.1 Funciones de morfología binaria

Como se explicó anteriormente, esta técnica de visión artificial hace uso de la erosión y dilatación, las cuales están implementadas en la librería *OpenCV*. La erosión está formulada con un procedimiento llamado `cvErode()` y la dilatación con `cvDilate()`, los cuales están estructurados de la siguiente manera:

```
void cvErode(
     IplImage* src,
     IplImage* dst,
     IplConvKernel* B = NULL,
     int iterations = 1
);

void cvDilate(
     IplImage* src,
     IplImage* dst,
     IplConvKernel* B = NULL,
     int iterations = 1
);
```

donde `IplImage` es la estructura que usa *OpenCV* para la interpretación de las imágenes, la cual es utilizada por IPL (*Intel Image Processing Library*). Posteriormente, en el capítulo de la implementación del proyecto, se explicarán los campos de dicha estructura utilizados en el desarrollo del mismo.

Por otro lado, `src` y `dst` son variables de tipo puntero a `IplImage` que representan la imagen de origen y destino, respectivamente. `iterations` es un variable de tipo entero que representa el número de iteraciones que se realizarán.

`IplConvKernel` es una máscara de convolución o núcleo de coeficientes enteros o caracteres que está definida en IPL como un tipo específico. Este argumento es nulo por defecto, lo cual significa que el núcleo es 3 x 3. Dichos núcleos se asignan mediante `cvCreateStructuringElementEx()`, la cual es una función que crea un elemento estructurante para el núcleo y, por otro lado, se liberan mediante `cvReleaseStructuringElement()`, que es un procedimiento que libera la memoria de un elemento estructurante. Ambos procedimientos se definen a continuación:

```
IplConvKernel* cvCreateStructuringElementEx(
```

```
        int cols,
        int rows,
        int anchor_x,
        int anchor_y,
        CvElementShape shape,
        int* values=NULL
    );

    void cvReleaseStructuringElement(
        IplConvKernel** element
    );
```

donde `cols` y `rows` son variables enteras que representan el tamaño en columnas y filas del elemento, respectivamente. `anchor_x` y `anchor_y` representan las coordenadas X y Y donde está situado el elemento. `shape` es una variable que se puede definir de forma propia usando `CV_SHAPE_CUSTOM` o con una forma predefinida con `CV_SHAPE_RECT`, `CV_SHAPE_CROSS`, `CV_SHAPE_ELLIPSE`. En el caso del proyecto en cuestión se usó `CV_SHAPE_ELLIPSE` ya que aportaba mejores resultados. Por otra parte, `values` es un vector con las celdas del elemento estructurante que se usa en este caso como nulo (Bradski y Kaehler,2008).

3.2.2 Funciones de umbralización

Como se mencionó previamente, en el desarrollo de este proyecto se utiliza el método de umbralización para binarizar la imagen dado cierto umbral. Para hacer esto, *OpenCV* usa un procedimiento denominado `cvThreshold()`, el cual se detalla a continuación:

```
    void cvThreshold(
        CvArr* src,
        CvArr* dst,
        double threshold,
        double max_value,
        int threshold_type
    );
```

en donde `CvArr` es una estructura cuyos parámetros hacen que sea equivalente a pasar la estructura `IplImange`, ya que ésta se deriva de `CvMat` que, a su vez, se deriva de `CvArr`. Las variables `src` y `dst` representan la imagen origen y destino, respectivamente. `threshold` es una variable flotante que representa el parámetro inferior del umbral. `max_value` es una variable flotante que representa el valor máximo o límite superior (generalmente es 255) que se usa en los tipos de umbral `CV_THRESH_BINARY`, `V_THRESH_BINARY_INV` y `CV_THRESH_TRUNC`. Asimismo, `threshold_type` representa el tipo de umbral a utilizar. En el caso del proyecto que se presenta se utilizan `CV_THRESH_BIN` y `CV_THRESH_BINARY_INV` para binarizar la imagen, aunque *OpenCV* dispone de diversos tipos umbral.

3.2.3 Funciones de segmentación

En el estudio realizado se necesitan obtener áreas con significado y, específicamente, se precisa aplicar la técnica segmentación dada una semilla. *OpenCV* posee una función para realizar este procesamiento denominada `cvFloodFill()`, la cual está compuesta con los siguientes argumentos:

```
void cvFloodFill(
     IplImage* img,
     CvPoint seedPoint,
     CvScalar newVal,
     CvScalar loDiff = cvScalarAll(0),
     CvScalar upDiff = cvScalarAll(0),
     CvConnectedComp* comp = NULL,
     int flags = 4,
     CvArr* mask = NULL
);
```

donde `img` es una variable de tipo puntero a `IplImage` que representa la imagen de origen a rellenar según la semilla `seedPoint`, la cual es una variable del tipo `CvPoint`

que contiene las coordenadas del punto semilla a buscar en la imagen para calcular la segmentación. La variable `newVal` es de tipo `CvScalar` y contiene los valores RGB del color correspondiente a pintar en la imagen que se calcula la segmentación.

Las variables `loDiff` y `upDiff`, también de tipo `CvScalar`, contienen los valores del números de píxeles de la semilla por debajo y por arriba, respectivamente, se tomarán para la segmentación. La variable `comp`, de tipo `CvConnectedComp`, contiene información con la cual dicha región fue rellenada (número de píxeles, rectángulo contenedor y valor medio). El valor de dicha variable por defecto es nulo, ya que se usa sólo si el usuario lo necesita.

La variable `flags` es de tipo entero que indica el modo de relleno, donde los 3 bits menos significativos indican la conectividad de la vecindad. Los bits más significativos se activan con `CV_FLOODFILL_FIXED_RANGE` y `CV_FLOODFILL_MASK_ONLY`. El primero indica que es de rango fijo, donde la diferencia es respecto al punto semilla. Si éste no es colocado, el rango ya no sería fijo sino flotante, lo cual indicaría que la diferencia sería respecto al píxel adyacente que fue rellenado. Por otro lado, `CV_FLOODFILL_MASK_ONLY` indica que solo es máscara, es decir, que si es activado se modifica sólo la variable `mask` pero no la imagen origen. Cada píxel que es rellenado se coloca a 255 en la máscara. En donde `mask` es una variable de tipo apuntador a `CvArr` que debe contener una imagen de igual resolución que la imagen origen y que debe ser totalmente de color negro (0) para que pueda ser pintada.

3.2.4 Funciones de histogramas

El cálculo de histogramas proporciona la representación gráfica de las tonalidades de colores de una imagen. *OpenCV* tiene implementadas una serie de funciones y procedimientos para manipular histogramas. A continuación se mencionan los

procedimientos potencialmente a ser utilizados en el desarrollo de la herramienta. La primera, la cual permite calcular un histograma, se define como:

```
void cvCalcHist(
     IplImage** image,
     CvHistogram* hist,
     int accumulate = 0,
     const CvArr* mask = NULL
);
```

en donde `image` es una variable de tipo vector de `IplImage` que representa la imagen a la cual se le desea calcular el histograma. `hist` es una variable de tipo apuntador a `CvHistogram` que representa el histograma que se formará dada una imagen. En donde `CvHistogram` es una estructura que posee *OpenCV* para definir los tipos de datos de un histograma.

La variable entera `accumulate` puede tomar valores de 0 (*false*) o 1 (*true*). Si se coloca esta variable en *true*, al realizar el cálculo no se borra el contenido de las celdas anteriores, lo cual da la oportunidad de obtener histogramas acumulados de varias imágenes. `mask` es una variable constante de tipo apuntador a `CvArr` que representa una imagen llamada máscara sobre la cual se calcula el histograma. En caso de que no se necesite trabajar con la máscara, como fue el caso particular de este estudio, la función puede recibir sólo los dos primeros argumentos.

Es importante destacar que el procedimiento `cvNormalizeHist()` normaliza el histograma para que la suma de todas las celdas sea un valor dado (factor). Esto resulta útil para visualizar o comparar histogramas. En el caso de este proyecto, se precisa el cálculo de histogramas para hacer una comparación, y por lo tanto resulta necesario del mencionado procedimiento.

```
void cvNormalizeHist(
     CvHistogram* hist,
```

```
        double factor
    );
```

en donde `hist` es una variable de tipo apuntador a `CvHistogram` que representa el histograma que se va a normalizar. La variable flotante factor representa el valor del factor mediante el cual se realizará la normalización.

Luego de haber normalizado los histogramas, se pueden comparar con la función `cvCompareHist()`, la cual compara dos histogramas del mismo tipo (número de dimensión y número de celda por dimensión), usando diferentes métodos que representan las medias de distancia. Luego de hacer la comparación, dicha función devuelve la medida distancia o similitud entre ellos. La mencionada función se define como:

```
    double cvCompareHist(
        const CvHistogram* hist1,
        const CvHistogram* hist2,
        int method
    );
```

en donde `hist1` y `hist2` son variables constantes de tipo apuntador `CvHistogram` que representan los histogramas a ser comparados, estos histogramas deben ser del mismo tipo. La variable entera `method` representa la medida de distancia. Existen varios métodos implementados en *OpenCV* de los cuales se utiliza el de intersección (`method = CV_COMP_INTERSECT`), que calcula el solapamiento o intersección entre los histogramas.

3.2.5 Funciones para invertir y convertir imágenes en escala de grises

Para convertir imágenes en escala de grises e invertirlas, la librería *OpenCV* utiliza un procedimiento denominado `cvCvtColor()` que recibe tres parámetros, de los cuales dos son las imágenes que deben ser del mismo número de canales y el otro es el parámetro que representa el tipo de conversión. Este procedimiento transforma la imagen fuente en la imagen destino, según el tipo de conversión y se define como:

```
void cvCvtColor(
     const CvArr* src,
     CvArr* dst,
     int code
);
```

en donde `src` y `dst` son variables constantes de tipo apuntador a `CvArr` que representan la imagen fuente y la imagen destino, respectivamente. La variable entera `code` representa el tipo de conversión. En este punto, cabe destacar que *OpenCV* define una gran variedad de conversiones, de los cuales la mayoría son para conversiones de colores. En el caso de este proyecto se requiere `CV_RGBA2BGR`, que invierte la imagen sin cambiar el color y, además, `CV_BGR2GRAY` y `CV_RGB2GRAY`, los cuales transforman la imagen en escala de grises.

Capítulo 4

Implementación de la herramienta y análisis de resultados

Luego de realizar las modificaciones a la cámara *web* y al cable USB para el suministro de energía requerida en el LED emisor de infrarrojo, se procede a la implementación de la herramienta. En este punto, cabe destacar que los cambios realizados constituyeron un paso fundamental en el desarrollo de la herramienta (ayudaron a que la captura de la imagen a procesar fuese bastante robusta, dado los buenos resultados que esto genera en la imagen).

Para el proceso de implementación de la herramienta se utilizó el sistema operativo *Microsoft Windows XP* Profesional, usando el programa *Microsoft Visual Studio* 2005 y como lenguaje base C++. Adicionalmente, se usó la librería *OpenCV*, con la cual se realizó el procesamiento de las funciones de Visión Artificial y el procesamiento en tiempo real de las ventanas gráficas. También se usaron algunas funciones API (del inglés ***A****pplication* ***P****rogramming* ***I****nterface*) de *Microsoft Windows XP* para poder trabajar con los eventos del ratón y para obtener la resolución de la pantalla para posicionar ventanas. Además de todo lo anteriormente señalado, fue necesaria la creación de diversas funciones y procedimientos para lograr el completo funcionamiento de la herramienta desarrollada.

En este capítulo se definen detalladamente las funciones y los procedimientos utilizados en el desarrollo de la herramienta que permite mover el cursor de un computador mediante el movimiento ocular. Luego, se explica el tratamiento de las imágenes para la obtención de parámetros óptimos. Por último, se expone el proceso de ejecución de la herramienta.

4.1 Definición de funciones y procedimientos

4.1.1 Funciones y procedimientos de *OpenCV*

A continuación se explican las funciones y procedimientos asociados con la implementación de la herramienta que se usaron para el tratamiento de las imágenes e interfaces gráficas. Primero se detallan todas las pertenecientes a la librería *OpenCV* que fueron de utilidad para el desarrollo del proyecto.

```
void cvZero( CvArr* arr );
```

Este procedimiento recibe un solo parámetro de tipo apuntador a `CvArr`, el cual representa una imagen. `cvZero()` lo que hace es establecer todos los valores de todos los canales de la matriz a 0 (color negro).

```
void cvSub(
     const CvArr* src1,
     const CvArr* src2,
     CvArr* dst,
     const CvArr* mask = NULL
);
```

El procedimiento `cvSub()` recibe cuatro argumentos y su función es restar los dos primeros para retornar el resultado en el tercer argumento. `src1`, `src2`, `dst` son

variables del tipo apuntador a `CvArr` que representan imágenes que deben tener el mismo tamaño y número de canales ya que `src2` es substraído de `sr1` y el resultado es almacenado en `dst`. La variable `mask` es del tipo apuntador a `CvArr` constituye una imagen para almacenar una máscara. Por lo general este último argumento se deja en nulo, pero si es distinto de nulo solo se calculan los elementos de `dst` distintos de cero y son colocados en dicha máscara.

```
void cvCvtColor(
     const CvArr* src,
     CvArr* dst,
     int code
);
```

Este procedimiento básicamente lo que hace es convertir una serie de canales de un espacio de color a otro, al igual que brinda la posibilidad de invertir la imagen. Las variables `src` y `dst` son de tipo apuntador a `CvArr` y representan la imagen origen y destino, respectivamente. Dichas imágenes deben tener el mismo número de canales. `code` es una variable del tipo entero para indicar el tipo de conversión que se desea realizar. *OpenCV* provee diversos tipos de conversión entre los cuales se encuentran `CV_RGB2GRAY` y `CV_BGR2GRAY` que transforma una imagen de color a escala de grises y `CV_RGBA2BGR` que invierte la imagen sin cambiarla de color.

```
void cvLine(
     CvArr* array,
     CvPoint pt1,
     CvPoint pt2,
     CvScalar color,
     int thickness = 1,
     int connectivity = 8
);
```

El procedimiento `cvLine()` se utiliza para dibujar líneas, las cuales se trazan en un lugar específico de la imagen, dando un punto de comienzo y uno de fin, así como también el color, grosor y estilo de línea. La variable `array` de tipo apuntador a `CvArr` representa la imagen en la cual se va a dibujar la línea. `pt1`, `pt2` son variables de tipo `CvPoint` para indicar el punto de comienzo y final de la línea, respectivamente. Cada uno de ellos, al ser una estructura `CvPoint`, tienen coordenadas (x, y). La variable `color`, de tipo `CvScalar`, representa el color de la línea. Generalmente, se hace uso de la macro `CV_RGB (R, G, B)` para definir el color deseado, ya que R, G y B representan los colores Rojo, Verde y Azul respectivamente. Por otra parte, `thickness` es una variable entera que por defecto es 1, la cual refleja el grosor de la línea que viene dado en píxeles. La variable `connectivity`, también entera, representa el tipo de línea, por defecto es 8, lo que refleja una línea suave y lisa.

```
void cvRectangle(
     CvArr* array,
     CvPoint pt1,
     CvPoint pt2,
     CvScalar color,
     int thickness = 1
);
```

El procedimiento `cvRectangle()` se utiliza para dibujar un rectángulo en la imagen fuente y recibe cinco argumentos. El primero de ellos, `array`, es de tipo apuntador a `CvArr` y se refiere a la imagen fuente. Los dos siguientes, `pt1` y `pt2`, son variables del tipo `CvPoint` que representan las coordenadas (x, y) de la esquina superior izquierda y de la esquina inferior derecha del rectángulo a dibujar, respectivamente. Estas últimas variables permiten establecer el tamaño del rectángulo de se desea dibujar. Por último, las variables `color` y `thickness`, idénticas a las definidas en `cvLine()`.

```
int cvWaitKey(int time);
```

La función `cvWaitKey()` pide al programa que se detenga y espere el tiempo dado por la variable entera `time`. Si `time` es cero o negativo el programa va a esperar indefinidamente hasta que se presione alguna tecla, retornando el valor de dicha tecla. Esta función se utiliza para capturar las imágenes en el proceso de calibración y en la terminación de la ejecución del programa.

```
IplImage* cvLoadImage(
     const char* filename,
     int iscolor = CV_LOAD_IMAGE_COLOR
);
```

La función `cvLoadImage()` se utiliza para cargar imágenes que han sido guardadas en disco y retorna una tipo `IplImage` que representa el tipo de la imagen. Recibe dos argumentos donde el primero es de tipo constante apuntador a carácter, el cual es llamado `filename`, que almacena el nombre de la imagen a cargar. El segundo argumento es una variable entera llamada `iscolor`, la cual indica si la imagen a cargar es a color, en escala de grises, entre otros. Para el efecto de este proyecto se utilizó en 1 o lo que es lo mismo `CV_LOAD_IMAGE_COLOR`, para cargar imágenes a color.

```
int cvSaveImage(
     const char* filename,
     const CvArr* image
);
```

La función `cvSaveImage()` se ocupa de almacenar imágenes que han sido capturadas. El primero de los dos argumentos que recibe, denominado `filename`, es de tipo constante apuntador a carácter y representa la ruta, nombre y extensión del archivo a guardar. El segundo de ellos es una variable de tipo constante apuntador a `CvArr` que representa la imagen que se almacena. Esta función almacena sólo imágenes de 8 bits o de 3 canales para la mayoría de formatos de archivos de imágenes.

```
int cvNamedWindow(
     const char* name,
     int flags = CV_WINDOW_AUTOSIZE
);
```

La función `cvNamedWindow()` crea una ventana que permite mostrar imágenes recibiendo dos argumentos. El primero es una variable de tipo constante apuntador a carácter denominada `name`, que representa el nombre de la ventana, el cual aparecerá en la parte superior de la ventana. El segundo argumento es una bandera llamada `flags` que brinda la posibilidad de ajustar la ventana al tamaño original de la imagen usando 1 o `CV_WINDOW_AUTOSIZE`. De igual forma la bandera puede tomar el valor 0, con el cual se deja a criterio del usuario que ajuste el tamaño de la ventana.

```
void cvDestroyWindow(char* name);
```

El procedimiento `cvDestroyWindow()` cierra la ventana y libera la memoria usada por ella y recibe un sólo parámetro apuntador a carácter llamado `name` para indicar el nombre de la ventana.

```
void cvMoveWindow(
     const char* name,
     int x,
     int y
);
```

El procedimiento `cvMoveWindow()`, simplemente mueve una ventana a las coordenadas (x, y) que representan la esquina de la ventana. Posee tres argumentos en los cuales el primero representa el nombre de la ventana, el segundo y el tercero las coordenadas x y y, respectivamente.

```
void cvResizeWindow(
     const char* name,
     int width,
     int height
);
```

El procedimiento `cvResizeWindow()` hace posible redimensionar o darle el tamaño deseado por el usuario a un ventana determinada. El primero de los tres argumento que posee lleva por nombre `name` y se refiere al nombre de la ventana. Los dos siguientes `width` y `height`, son para indicar el ancho y alto de la misma, respectivamente.

```
void cvShowImage(
     const char* name,
     const CvArr* image
);
```

El procedimiento `cvShowImage()` es utilizado para mostrar una imagen en determinada ventana. El primer argumento es de tipo constante apuntador a carácter llamado `name` para especificar el nombre de la ventana en la cual se mostrará la imagen. El segundo y último argumento es de tipo constante apuntador a `CvArr`, con el cual se indica la imagen a mostrar.

```
CvCapture* cvCaptureFromCAM( int index );
```

`CvCapture` es una estructura que posee la librería para trabajar con la captura de video, la cual contiene la información necesaria para leer los marcos de una cámara o un archivo de video. Por otro lado, la función `cvCaptureFromCAM()` está diseñada para seleccionar y activar la cámara con la cual se va a realizar la captura de las imágenes. Como argumento recibe una variable entera llamada `index` que representa un

identificador que indica cómo se desea acceder a la cámara. Si dicho identificador es 0 quiere decir que se toma la primera cámara detectada y, si se tienen varias cámaras conectadas, este identificador puede ser -1 con lo cual se despliega un menú para seleccionar la cámara deseada.

```
int cvGrabFrame( CvCapture* capture );
```

La función `cvGrabFrame()` se utiliza para validar si se encuentra conectada una cámara al computador. Esto es posible debido a que recibe como argumento una variable de tipo apuntador a `CvCapture`, la cual contiene información que valida si hay o no una cámara conectada al computador. Dicha función retorna un valor entero que puede ser 0 o 1, donde 0 indica una respuesta negativa y 1 una respuesta afirmativa.

```
IplImage* cvRetrieveFrame( CvCapture* capture );
```

La función `cvRetrieveFrame()` se utiliza para capturar las imágenes en una variable de tipo `IplImage` y trabajar con la misma. Es usada una vez que es llamada `cvGrabFrame()`, ya que dicha función hace necesaria la transformación de la estructura `CvCaptur`e para retornar una estructura de tipo `IplImage`. Recibe un sólo argumento que corresponde a una estructura de tipo apuntador `CvCapture`.

```
IplImage* cvCreateImage(
          CvSize size,
          int depth,
          int channels
);
```

La función `cvCreateImage()` es usada para crear imágenes. Esto se debe a que cuando se define una variable que es de tipo `IplImage`, se puede asignar el tipo de

imagen que se va a almacenar en dicha variable, lo cual se logra con esta función. La misma recibe tres argumentos, en donde el primero de ellos es de tipo `CvSize` que corresponde al tamaño de la imagen (ejemplo: 640x480 píxeles). El segundo define la profundidad de la imagen, la cual se estableció en 8 para el proyecto que se presenta. Esto se debe a que la librería *OpenCV* la preestablece en dicho valor para el tratado de imágenes, debido a que arroja buenos resultados. El tercer y último argumento representa el número de canales de la imagen, en el caso del proyecto se usan de 3 y de 1 canal, ya que se trabaja con imágenes a color y en escala de grises, respectivamente. La mencionada función retorna un apuntador a `IplImage`, la cual tendrá las modificaciones requeridas.

```
void  cvReleaseImage( IplImage** image );
```

La función `cvReleaseImage()` es un procedimiento usado para liberar la memoria ocupada por una determinada imagen. Recibe un único argumento que representa la imagen a la cual se liberará la memoria.

```
void cvSetImageROI(
     IplImage* image,
     CvRect rect
);
```

El procedimiento `cvSetImageROI()` es utilizado para trabajar sólo con una porción de la imagen, resulta útil en casos cuando es necesario trabajar con una zona deseada de una determinada imagen y, con esto, agilizar los tiempos de procesamiento. Este procedimiento recibe dos argumentos, donde el primero representa la imagen a la cual se le tomará la porción de interés y el segundo argumento una variable llamada `rect` de tipo `CvRect()`. La mencionada estructura es de la forma

`cvRect(x,y,width,height)`, donde ella recibe 4 argumentos que representan las coordenadas (x,y), ancho y alto del rectángulo que formará la zona deseada, respectivamente.

```
void cvResetImageROI( IplImage* image );
```

El procedimiento `cvResetImageROI()` se usa para liberar la porción de la imagen que fue tomada para trabajar usando `cvSetImageROI()`.

4.1.2 Funciones y procedimientos de API

Tal y como se mencionó anteriormente, además de haber utilizado funciones pertenecientes a la librería conocida como *OpenCV*, se recurrió a la bondades de la librería de Interfaz de Programación de Aplicaciones (API), por sus siglas en inglés *Application Programming Interface*. API es un conjunto de funciones y procedimientos que ofrece el Sistema Operativo (SO) al programador, y representa una interfaz de comunicación entre diversos componentes de *software*. Debido a que para el proyecto realizado se trabajó bajo el sistema operativo *Windows* XP, se utilizaron las Win32, que son un conjunto de funciones y procedimientos API que permiten que una aplicación se ejecute bajo dicho sistema operativo. Entre los beneficios que se obtuvieron de la librería mencionada está la posibilidad de trabajar con los eventos del ratón, posicionar el cursor y obtener la resolución de pantalla. A continuación se detallan cada una de las rutinas que pasaron a formar parte del desarrollo del proyecto.

```
void mouse_event(
    DWORD dwFlags
    DWORD dx,
    DWORD dy,
    DWORD dwData,
```

```
    DWORD dwExtraInfo
);
```

El procedimiento `mouse_event()` permite ejecutar diferentes acciones que realiza un ratón, como el movimiento y los clic. Cabe destacar que, en este caso particular, solo se usó para generar el evento del clic. El primero de los cinco argumentos que recibe dicho procedimiento es `dwFlags`, que representa una bandera que especifica variantes de movimientos y los clics. Las banderas usadas en el proyecto son: `MOUSEEVENTF_LEFTDOWN`, el cual especifica que el botón izquierdo del ratón ha cambiado a estar pulsado y `MOUSEEVENTF_LEFTUP`, que especifica que el botón izquierdo ha cambiado a no estar pulsado. El resto de los argumentos no fueron usados ya que se usan para cambios de posición y movimiento de la rueda del ratón.

```
BOOL SetCursorPos(
    int X,
    int Y
);
```

La función `SetCursorPos()` se encarga de mover el cursor a las coordenadas de la pantalla que han sido especificadas en sus argumentos. Si las nuevas coordenadas no se encuentran en el campo de la pantalla, *windows* ajusta automáticamente las coordenadas de manera que este quede siempre dentro de la misma. La función posee dos argumentos, en donde el primer argumento `X`, representa el movimiento horizontal del cursor y el segundo `Y`, el movimiento vertical. Si la función tiene éxito al realizar el movimiento, retorna un valor distinto de cero y, en caso contrario, el valor retornado es cero.

```
BOOL GetCursorPos( LPPOINT lpPoint );
```

La función `GetCursorPos()` es usada para recuperar la posición del cursor en coordenadas de la pantalla. Dicha función posee un solo argumento que es `lpPoint`, el cual es un puntero a una estructura de tipo `POINT`, compuesta por X y Y para almacenar las coordenadas, y representa la dirección de la estructura para la posición del cursor.

```
int GetSystemMetrics( int nIndex );
```

La función `GetSystemMetrics()` recupera varios valores de medidas y de parámetros de configuración del sistema. Las medidas del sistema son las dimensiones (ancho y alto) de elementos del *display* de *windows*, el cual expresa todas sus medidas en píxeles. El parámetro de tipo entero que recibe `GetSystemMetrics()`, denominado `nIndex`, especifica la configuración del sistema a recuperar. Para efectos del proyecto se usó `SM_CXSCREEN` ó `0` y `SM_CYSCREEN` ó `1`, que representan ancho y alto, respectivamente.

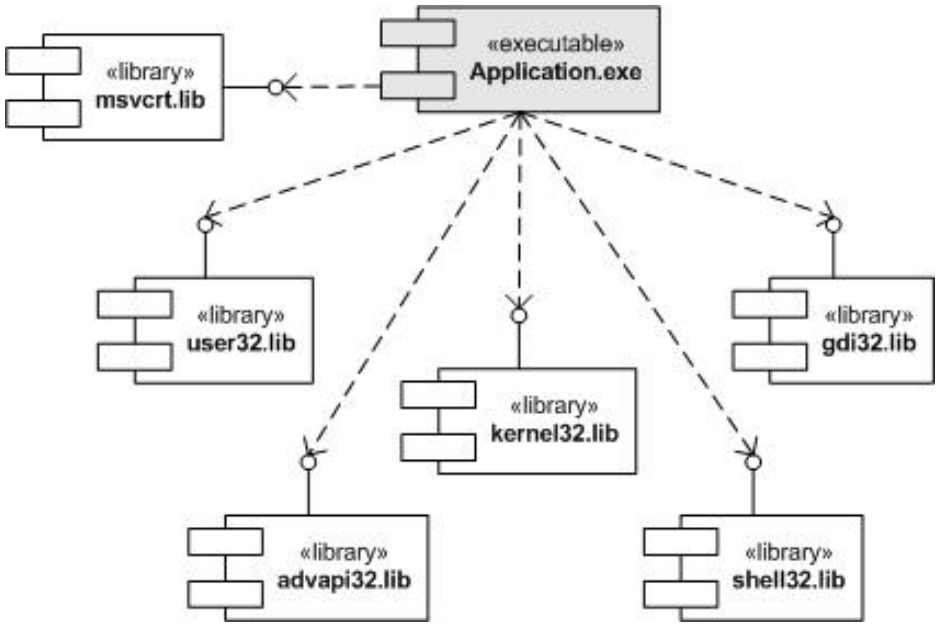

Figura 4.1: Diagrama de componentes de las dependencias de funciones API Win32

4.1.3 Funciones y procedimientos creados

Además de todas las funciones y los procedimientos especificados anteriormente, fue necesario desarrollar diversas rutinas para lograr el funcionamiento efectivo de la herramienta propuesta. Para esto, se recurrió el lenguaje de programación C++ utilizando *Visual Studio* 2005. A continuación se explican detalladamente cada una de las rutinas, indicando entre otras cosas, su funcionamiento y utilidad para la creación y ejecución de la herramienta que permitirá mover el cursor del computador mediante el movimiento ocular.

```
void buscar_semilla(
     int ancho,
     int alto,
     IplImage* img,
     uchar val_pix,
     uchar val_pix_aux,
     int &minX,
     int &minY
);
```

El primer procedimiento denominado `buscar_semilla()` se encarga de buscar en una imagen determinada el valor en coordenadas del píxel semilla. Dicho píxel debe ser el valor en la matriz que se aproxime más a cero o color negro. En este sentido, lo que hace es recorrer la matriz por filas y columnas de manera que se vaya buscando el píxel que se acerque más a cero, y se almacenan las coordenadas de dicho píxel. El procedimiento recibe siete argumentos, entre los cuales están las variables enteras `ancho` y `alto` que indican el ancho y alto de la imagen, respectivamente. La variable `img` , de tipo apuntador a `IplImage`, representa la imagen que va a ser objeto de la búsqueda de la semilla. El cuarto y quinto argumento son `val_pix` y `val_pix_aux`, respectivamente, ambos de tipo `uchar` y usadas como variables auxiliares, en las cuales se almacenan los valores correspondientes a las tonalidades de grises de la imagen. Por

último, `minX` y `minY` son variables de tipo entero que almacenan las coordenadas del píxel que resulta ser la semilla de la imagen especificada. Estas dos últimas variables se pasan por referencia para ser modificadas al usar el procedimiento.

```
uchar getPixel(
     IplImage*img,
     int lin,
     int col
);
```

La función `getPixel()` se utiliza para obtener el valor de un píxel dado las coordenadas X y Y. Dicha función es solo para imágenes en escala de grises y binarias ya que se trabaja sólo con un canal y posee tres argumentos, el primero de los cuales es una variable de tipo apuntador a `IplImage` denominado `img` que representa la imagen de la cual se va tomar el valor del píxel. El segundo y tercer argumento son variables enteras denominadas `lin` y `col` que representan las coordenadas X y Y del píxel, respectivamente. La función retorna el valor del píxel seleccionado.

```
void setPixel(
     IplImage*img,
     int lin,
     int col,
     uchar valor
) ;
```

El procedimiento `setPixel` es usado para colocar el valor de un píxel en una imagen en escala de grises o binaria. El primero de sus cuatro argumentos es una variable de tipo apuntador a `IplImage` denominada `img` que representa la imagen que se desea utilizar. Los argumentos `lin` y `col` son variables enteras que corresponden a las coordenadas X y Y, respectivamente, donde se pintará el píxel. El último argumento

denominado valor es de tipo `uchar` y representa el valor del píxel a pintar. Cabe destacar que los valores que puede toma dicha variable están entre 0 y 255.

```
void setPixelColor(
     IplImage*img,
     int lin,
     int col,
     uchar rojo,
     uchar verde,
     uchar azul
);
```

El procedimiento `setPixelColor()` hace lo mismo que `setPixel()` con la única diferencia que éste pinta colores. En este sentido, este procedimiento posee tres argumentos más en los cuales se coloca el valor de rojo, verde y azul que determinarán el color final a mostrar.

```
void calcular_centroide(
     IplImage* img,
     int* valores
);
```

El procedimiento `calcular_centroide()` es usado para calcular el centroide de una circunferencia aproximada que se encuentra en una imagen binaria. El procedimiento da por entendido que en la imagen hay una circunferencia de color blanco (255) y el fondo es negro (0). Este procedimiento recibe dos argumentos, donde el primero es una variable denomina `img` que es de tipo puntero a `IplImage` y representa la imagen en la cual se buscará el centroide. El segundo argumento es un vector de enteros el cual contiene los valores de la búsqueda del centroide, para lo cual en la posición 0 se encontrará la coordenada X y en la posición 1 la coordenada Y.

```
void dibujar_CRUZ(
     IplImage* img,
     int *valores,
     int tam
);
```

El procedimiento `dibujar_CRUZ()` dibuja una cruz en una imagen en escala de grises o binaria tomando como base las coordenadas del píxel centroide. El primer argumento que recibe es una variable de tipo apuntador a `img`, que representa la imagen en la cual se va a dibujar la cruz. El segundo es un vector de valores enteros llamado `valores`, el cual contiene los coordenadas X y Y del píxel centroide. El último argumento es una variable entera denominada `tam`, que representa el tamaño en píxeles de la cruz a dibujar.

```
void dibujar_CRUZ_Color(
     IplImage* img,
     int *valores,
     int tam
     uchar rojo,
     uchar verde,
     uchar azul
);
```

El procedimiento `dibujar_CRUZ_Color()` es muy parecido a `dibujar_CRUZ()` con la única diferencia que éste trabaja con imágenes de tres canales, es decir imágenes a color. Por lo tanto, éste adicionalmente recibe tres parámetros que representan los colores rojo, verde y azul, respectivamente, para indicar el color final a mostrar.

```
void MostrarImagen(
     char* Name,
     IplImage* imagen
);
```

El procedimiento `MostrarImagen()` muestra una imagen por pantalla de igual dimensión que la imagen original y posee dos argumentos, donde el primero es de tipo apuntador a carácter denominado `Name`, el cual representa la cadena de caracteres que conforman el nombre de la ventana. El segundo es la imagen que se quiere mostrar. Cabe destacar que este procedimiento crea la imagen al momento de mostrarla y luego con presionar cualquier tecla la ventana se cierra y se libera la memoria.

```
void ObservarMovimiento(
     char* NombreVentana,
     char* NombreImagen
);
```

El procedimiento `ObervarMovimiento()` muestra una imagen en la esquina inferior derecha de la pantalla. Es importante resaltar que dicho procedimiento no crea la ventana en la cual se carga la imagen ya que la misma debió haber sido previamente creada y, además, toma la imagen a mostrar de un archivo usando la función `cvLoadImage()`. Esta rutina recibe dos argumentos, ambos de tipo apuntador a carácter, denominados `NombreVentana` y `NombreImagen` que indican el nombre de la ventana y el de la imagen, respectivamente.

```
void Morfologia();
```

El procedimiento `Morfologia()` es usado para aplicar morfología binaria a la imagen en la fase de detección del centroide de la pupila. En este procedimiento se hace uso de los procedimientos implementados en *OpenCV* para hacer la erosión y dilatación. El mismo no recibe ningún parámetro, ya que las variables para trabajar con las imágenes origen y destino son declaradas como globales. El valor del entero n es 7 que era el valor

apropiado para lo que se estaba buscando y que se determinó por ensayo y error. Esto se explica mejor más adelante cuando se detalle la estructura de la herramienta.

```
void segmentacion(
     int x,
     int y,
     int lo,
     int up,
     IplImage* imgB
);
```

El procedimiento `segmentacion()` fue creado para ejecutar el algoritmo de segmentación dado una semilla con el fin de poder hallar el centro de la pupila. Éste recibe el punto semilla en coordenadas (x, y), para luego ser usado por el procedimiento `cvFlooFill()` para recorrer la región semilla. También se binariza usando un umbral específico la imagen mascara (`mask`), para luego en ella pintar la región segmentada. Dicho procedimiento posee cinco argumentos de los cuales `x` e `y` son variables enteras, que corresponden al punto semilla en coordendas X y Y, respectivamente. Las variables `lo` y `up` son variables enteras que corresponden a los valores alto y bajo (inferior y superior) para hacer el recorrido del algoritmo de segmentación. El quinto y último argumento `imgB` es una variable del tipo apuntador a `IplImage` que representa la imagen de entrada, a la cual se le aplicará el algoritmo de segmentación.

```
void DeteccionOjoAbiertoCerrado(
     IplImage* color_img0,
     IplImage* color_img,
     IplImage* gray_img0,
     IplImage* Ori_Gris,
     IplImage* color_img4,
     IplImage* color_img5,
     CvHistogram* hist1,
     CvHistogram* hist2,
     double &CompHist
```

```
);
```

El procedimiento `DeteccionOjoAbiertoCerrado()`, se usa para hacer una comparación de imágenes y determinar cuándo se está en presencia de un clic. El mismo hace uso de funciones y procedimientos implantados en la librería *OpenCV* para invertir imágenes, cambiar imágenes a escala de grises, restar imágenes, histogramas y seleccionar sólo una región de la imagen como zona de interés. El primer argumento `color_img0` es variable de tipo apuntador a `IplImage` que representa la imagen origen a ser tratada. La variable `color_img`, también de tipo apuntador a `IplImage`, representa la imagen que se muestra en el entorno gráfico. Esto se hace para aprovechar el algoritmo y los tiempos de ejecución, de manera que no sea necesario hacer otro procedimiento para invertir la imagen a mostrar. Cabe destacar que no se usa la misma imagen a tratar, ya que hay que transformarla en escala de grises para hacer el procesamiento, lo cual limita la posibilidad de poder pintar con diferentes colores el centroide de la pupila y los rectángulos que representan las regiones de interés. El tercer argumento, denominado `gray_img0`, es una variable de tipo apuntador a `IplImage` que se usa para hacer una copia de la imagen original pero en escala de grises, para luego hacer el procesamiento de buscar la semilla, hacer la segmentación, morfología y calcular el centroide de la pupila, lo cual se explicará posteriormente. Por otro lado, la variable `Ori_Gris` de tipo apuntador a `IplImage`, correspondiente al cuarto argumento, se usa para hacer la resta entre la imagen ojo cerrado capturada en el proceso de calibración y la que es capturada en tiempo real. La variable `color_img4` de tipo apuntador a `IplImage` representa la imagen ojo cerrado, capturada en el proceso de calibración y `color_img5`, también de tipo apuntador a `IplImage`, representa una imagen auxiliar que almacena la resta entre las dos imágenes mencionadas anteriormente. Esta última imagen se umbraliza para luego calcular y normalizar su histograma, hacer la comparación que retorna un valor para proporcionar la medida que permite saber si la

imagen es igual (0) o no (diferente de 0). Cabe destacar que la umbralización de dicha imagen se hace usando `CV_THRESH_BINARY_INV` para que la imagen se convierta en binaria e invertida. Los argumentos `hist1` e `hist2` son variables de tipo apuntador a `CvHistogram` en donde `hist1` representa el objeto donde se almacenará el histograma de la imagen resultante de la resta entre las dos imágenes y, por otro lado, `hist2` representa el histograma de una imagen que está compuesta sólo de color negro (todos los valores de dicha matriz son cero). El noveno y último argumento con nombre `CompHist` es una variable del tipo *double* que almacena el resultado de la comparación entre los histogramas.

```
void EncontrarCentroidePintar(
     IplImage* img,
     IplImage* mask,
     int semiX,
     int semiY,
     uchar val_pix_cali,
     uchar val_pix_aux_cali,
     int *valores
);
```

El procedimiento `EncontrarCentroidePintar()` es la síntesis de una serie de procedimientos que son llamados en él. Internamente son usados en el siguiente orden: `buscar_semilla()`, `cvZero()`, `segmentacion()`, `Morfologia()` y `calcular_centroide()`. Sólo con este procedimiento se logra calcular el centroide de la pupila. Recibe siete argumentos de los cuales `img` es el primero, el cual es una variable de tipo apuntador a `IplImage` que representa la imagen a la cual se le calculará el centroide de la pupila. El segundo, `mask`, es una variable de tipo apuntador a `IplImage` que representa la imagen máscara en la cual se dibujará el círculo aproximado usando el procedimiento segmentación. Las variables enteras `semiX` y `semiY` indican las coordenadas X y Y del píxel semilla. `val_pix_cali` y

`val_pix_aux_cali` son variables de tipo `uchar` que se usan como auxiliares para usarlas en el procedimiento `buscar_semilla()`. Y, por último, `valores` es un vector de enteros en el cual se retornarán los valores X y Y de las coordenadas del píxel centroide de la pupila.

```
void EventoClickMouse(
     double CompHist,
     int &band,
     int &counter,
     char* VentanaMove
);
```

El procedimiento `EventoClickMouse()` es usado para validar si se está en presencia de un clic, ya que recibe como parámetro la comparación de histogramas denominado `CompHist`, en donde se revisa si dicha variable vale cero lo que indica que las imágenes son idénticas y, luego, se comienza a incrementar un contador. Si el contador llega a cuatro se activa el procedimiento API llamado `mouse_event()`, el cual ejecuta el clic. El procedimiento recibe `CompHist` como primer argumento, la cual es una variable de tipo `double` que contiene el resultado de la comparación entre los histogramas. `band` es una variable entera que representa una bandera para validar si `CompHist` es igual a cero. La variable entera `counter` es el contador encargado de ir almacenando los incrementos que se dan cuando se está en presencia de imágenes iguales. El cuarto argumento, `VentanaMove`, es una variable apuntador a caracter que presenta la cadena de caracteres que indica el nombre de la ventana en la cual mostrar la imagen cuando se ha ejecutado un clic, para lo cual se hace uso del procedimiento `ObservarMovimiento()`.

```
void Mover_CursorTR(
     POINT posVieja,
     int *valores,
```

```
        POINT posActual,
        CvPoint factor,
        int paso,
        char* VentanaMove
    );
```

El procedimiento `Mover_CursorTR()` es para hacer el movimiento del cursor en tiempo real dado ciertos parámetros. Dicho procedimiento hace uso del procedimiento API denominado `SetCursorPos()`, que permite ubicar al cursor en la posición deseada. Recibe como argumento primeramente a `posVieja`, que es una variable de tipo `POINT` para indicar las coordenadas del centro de la pupila de la imagen de ojo abierto tomadas en el momento del proceso de calibración. Este centroide sirve también como base para dibujar un recuadro de color rojo, el cual sirve como guía al momento de mover el ojo hacia alguna dirección. El segundo argumento denominado `valores` es un vector de enteros que contiene la posición del centroide de la pupila en coordenadas X y Y en tiempo real. La variable `posActual`, de tipo `POINT`, representa las coordenadas X y Y de la ubicación del cursor en la pantalla, también en tiempo real. Para asignar el valor a esta variable se usa el procedimiento API llamado `GetCursorPos()` y se pasa la variable por referencia. `factor` es una variable de tipo `CvPoint` para indicar las coordenadas X y Y, la cual se utiliza para graduar o redimensionar el cuadrado rojo, que se usa para determinar la sensibilidad con la que se desplazará el cursor al mover el ojo. La variable entera `paso` se usa para graduar la velocidad de movimiento del cursor en tiempo real. Por último, `VentanaMove` es una variable que representa la cadena de caracteres que indica el nombre de la ventana en la cual se mostrarán las imágenes correspondientes al tipo de movimiento (derecha, izquierda, arriba y abajo). Para esto se hace uso de el procedimiento `ObservarMovimiento()`.

4.2 Tratamiento de la imagen para la obtención de parámetros óptimos

Luego de culminar la explicación de todas las funciones y los procedimientos que se utilizan en la implementación de la herramienta, se explica paso a paso cómo se obtuvieron los valores de umbrales, mediante ensayo y error. Primero que todo, se explicará como se realizó la selección de los parámetros con los cuales se logró hacer una buena segmentación de la imagen. En la figura 4.2 se puede observar un ejemplo de una imagen obtenida a través de la cámara modificada utilizada.

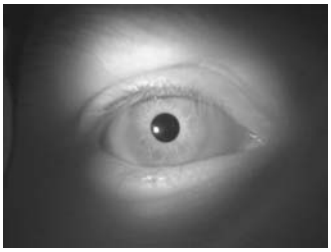

Figura 4.2: Imagen original obtenida de la cámara dadas las modificaciones

Luego de obtener la imagen se selecciona sólo una región de ella, tal como se observa en la figura 4.3, en donde la zona de interés es identificada con el rectángulo amarillo. Luego, se procede a aplicarle el algoritmo de `buscar_semilla()`, el cual buscará el píxel que se aproxime más al color negro. Como se observa en la imagen será un píxel que se encuentra en la región de la pupila.

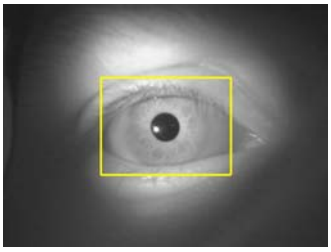

Figura 4.3: Imagen donde se observa la región a ser tratada

Luego de haber encontrado el píxel semilla se aplica el algoritmo llamado `segmentacion()`, el cual utiliza las coordenadas del píxel semilla y valores `lo` y `up` para aplicar a dicha imagen. El cálculo del valor `up` se realizó utilizando ensayo y error, para saber qué valores se acoplaban mejor a la imagen y zona de interés a ser tratada. Cabe destacar que el valor `lo` se fijó en 20 debido a que se trabaja con imágenes en escala de grises y, además, sólo se busca el valor que más se aproxime a 0 o al color negro. Esto se basa en que la zona de la pupila va a tener siempre valores que oscilan entre 20 y 35, es decir valores que indican colores cercanos a negro. Ahora, para saber el valor de `up`, se realizaron pruebas variando dichos parámetro.

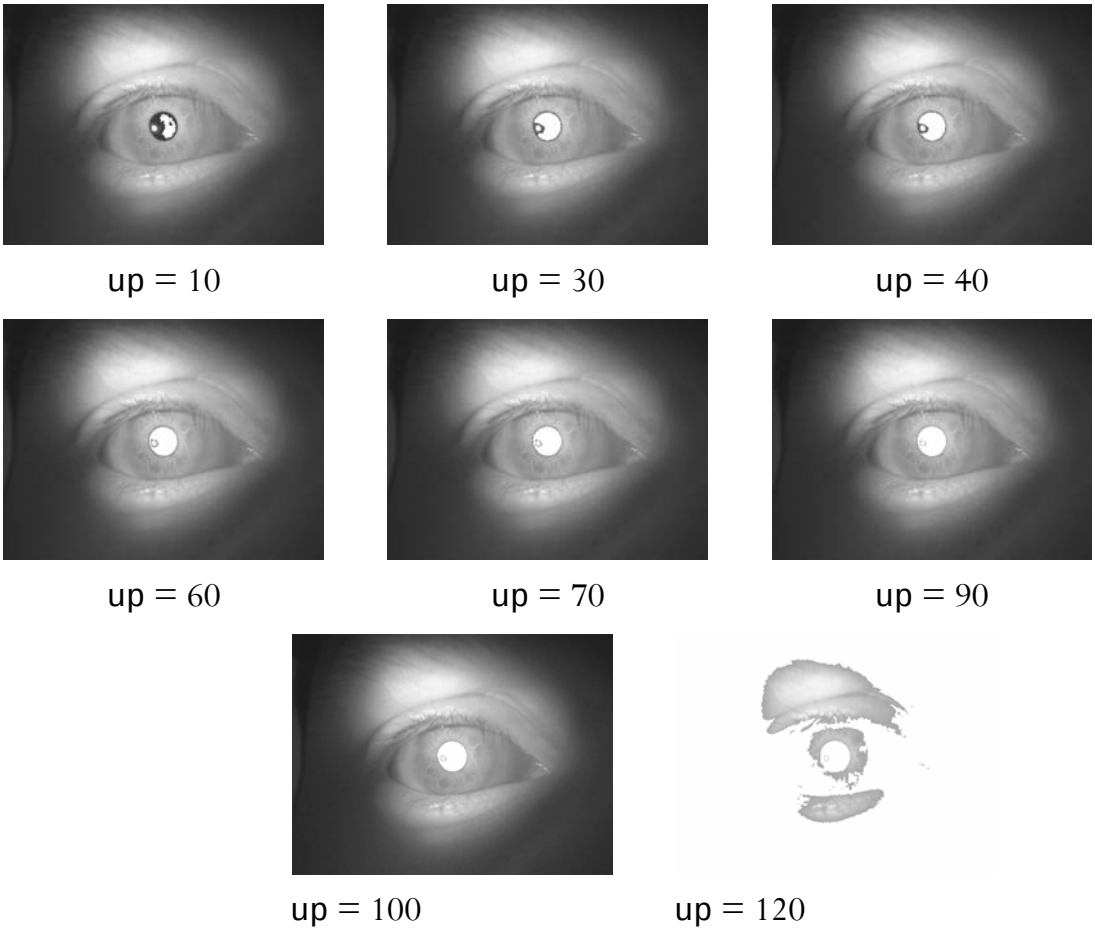

Figura 4.4: Variación del parámetro `up` en algoritmo de segmentación

De acuerdo al valor de dicho parámetro, se muestran en la figura 4.4 los diferentes resultados. Se puede observar las diferentes zonas que se pintan de blanco luego de recorrido el algoritmo. Cuando `up` tiene un valor de 10, la región resultante es muy escasa ya que no recorre en gran medida la zona de interés. Por otro lado, para `up` igual a 120, la zona es demasiado amplia y, además de tomar la región la pupila, también pintar gran parte de la imagen, lo que causa una pérdida de la zona de interés. Observando estas imágenes se decidió tomar el valor de `up` en 60, el cual es intermedio, con lo que se logra que el algoritmo no sea muy sensible a variaciones del entorno y que tampoco tome una región de interés no deseada.

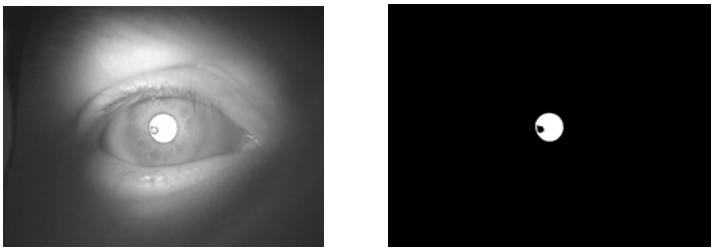

Figura 4.5: Imagen segmentada e imagen binaria de la zona de interés

Luego de tener seleccionados los parámetros que servirán para segmentar la imagen se binariza la misma de manera tal que se destaque sólo la pupila y el resto quede pintado de negro. En la figura 4.5 se muestra la imagen a tratar y la máscara que proporciona el algoritmo de segmentación, la imagen tiene una mancha negra que se debe al brillo de la luz infrarroja reflejado en la pupila, generando la imperfección en la imagen. Debido a que se necesitaba obtener un círculo aproximado, la mancha en la imagen genera una anomalía no deseada, y para lograr limpiarla, se usa la morfología binaria. Ahora, tomando como base la imagen binaria se aplica el algoritmo denominado `Morfologia()`, en el cual se aplica erosión y dilatación. *OpenCV* en los procedimientos de erosión y dilatación usa una estructura denominada `element` que corresponde al elemento usado para realizar la morfología. En este caso proyecto se usó `CV_SHAPE_ELLIPSE`, el cual es uno de los que más se adecúa al proyecto, ya que se

requiere aproximar a una circunferencia y éste elemento brinda esa posibilidad. Adicionalmente, se debe definir un variable entera denominada n, la cual varía el tamaño del elemento para realizar la morfología. Para decidir el valor más adecuado, se aplica el método de ensayo y error. Los resultados se muestran en la siguiente figura.

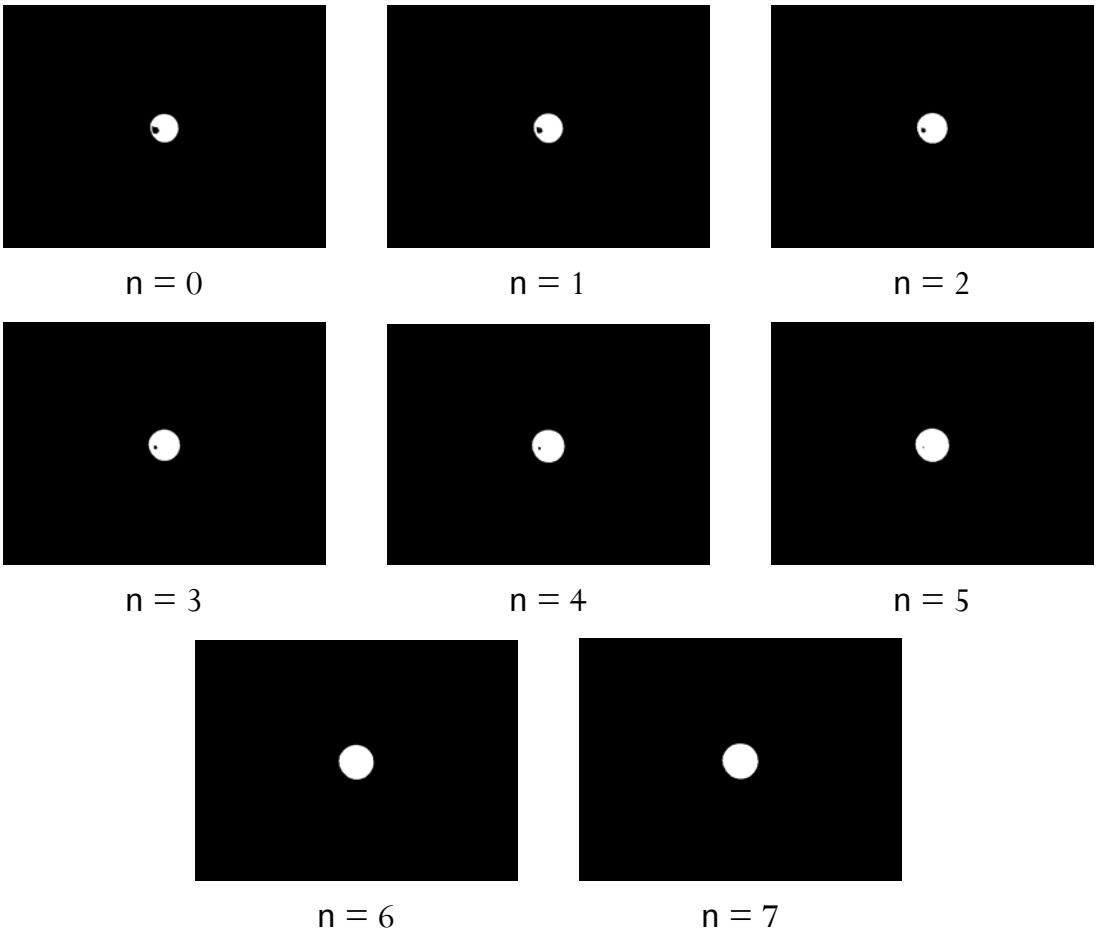

Figura 4.6: Morfología de la zona de interés variando el parámetro n

Como se observa en la figura 4.6 al incrementar el valor del parámetro n se va logrando limpiar la imagen. Cuando n es igual a 6 y a 7 ya se ha podido tener un círculo aproximado bastante bien. Con esto se pudo continuar con el procesamiento usando la zona de interés ya limpia, seleccionado n = 6.

Posteriormente, se procedió a aplicar el algoritmo `calcular_centroide()`, el cual, partiendo de la imagen binaria con `n` = 6 que se muestra en la figura 4.5, calcula el centro aproximado de dicha circunferencia. Para efectos de visualizar mejor este punto, se pinta un cruz usando el algoritmo `dibujar_CRUZ()` y lo cual da como resultado la imagen que se muestra en la figura 4.7. Luego, dibujando esta cruz pero a color y sobre la imagen del ojo usando el algoritmo de `dibujar_CRUZ_Color()`, el resultado sería como se muestra en la figura 4.8.

Figura 4.7: Resultado al encontrar centro y pintar la cruz en imagen binaria

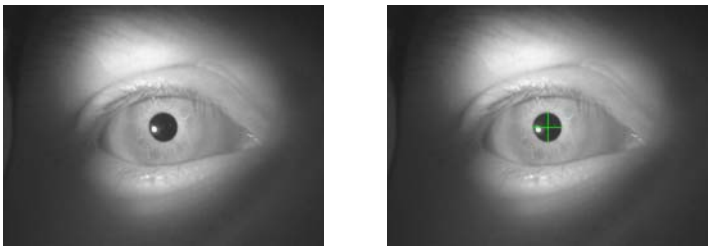

Figura 4.8: Resultado al pintar centroide con una cruz en imagen del ojo

Como se observa en la figura 4.8, al obtener el centro del círculo, se está en presencia del centroide de la pupila. Teniendo este punto indicador, se puede realizar el movimiento del cursor definiendo una zona, en la cual la cruz sale de un cuadro precalibrado. El tamaño de este cuadro se deja a criterio del usuario, ya que éste definirá la sensibilidad respecto al movimiento del ojo para mover el cursor de una posición a otra. El recuadro que se pinta en rojo, tiene un tamaño por defecto, pero puede ser modificado en tiempo de ejecución para comodidad del usuario y este queda representado como se muestra en la figura 4.9. Cabe destacar que dicho cuadro es

creado siguiendo como referencia la imagen de ojo abierto tomada en el momento del proceso de calibración. Por lo tanto, es importante que el ojo esté lo más centrado posible en dicho proceso, ya que partiendo de ese punto de calibración se dibujará el cuadro rojo en la imagen.

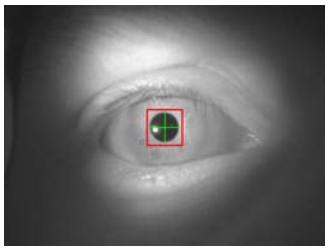

Figura 4.9: Cuadro rojo que delimita la zona de movimiento

Para hacer el movimiento se usa el algoritmo `Mover_CursorTR()`, el cual toma las coordenadas del centro de la pupila calculado a partir de la imagen de ojo abierto que se capturó en el proceso de calibración, para tenerlos como referencia al hacer el movimiento. Dicho procedimiento toma valores calculados en tiempo real de la imagen que se captura por pantalla, la sensibilidad del movimiento del cursor, entre otros. Con estos principales argumentos se hace una serie de sentencias de comparación para saber en qué momento el usuario desea mover el cursor hacia alguna posición. Para hacer que el cursor se mueva se usan las funciones API que se explicaron anteriormente. Un ejemplo de la ventana de calibración donde se toma la imagen ojo abierto es la que se muestra en la figura 4.10.

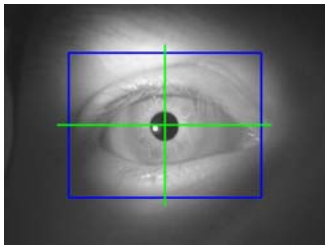

Figura 4.10: Ejemplo de la ventana de calibración capturando ojo abierto

Luego de lograr hacer el movimiento del cursor, se procedió a la implementación de clic, el cual lograría que el usuario, conjuntamente con el movimiento, pudiera acceder a las distintas aplicaciones que un computador personal brinda. Al llegar a este punto se determinó la necesidad de agregar al proceso de calibración la captura de una imagen de ojo cerrado, en la cual se captura la imagen de la manera que el usuario desea hacer el clic. Un ejemplo de la imagen que se toma en el proceso de calibración para el ojo cerrado es la que se observa en la figura 4.11.

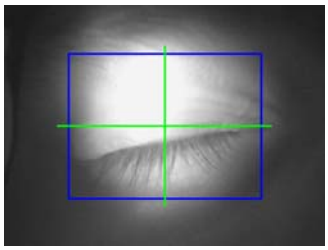

Figura 4.11: Ejemplo de la ventana de calibración capturando ojo cerrado

Con la imagen de ojo cerrado se realiza un proceso en el cual se hace la resta dicha imagen con las imágenes tomadas en tiempo real. El histograma de la resta obtenida se compara con el histograma de una imagen totalmente negra, si ambos histogramas son iguales se retorna un valor cero. De ser así, esto indicará que se está en presencia de un clic. No obstante, es importante resaltar que antes de comparar los histogramas de las imágenes fue necesario usar la umbralización, con el fin de transformar la imagen resultante de la resta en una imagen binaria. Esto se hace para poder regular que la imagen obtenida para identificar el clic no deba ser exactamente igual a la de ojo cerrado del proceso de calibración. Con ello se logra dar holgura a la forma cómo se cierra el ojo, ya que resultaría sumamente difícil lograr siempre colocar el ojo exactamente igual a como se hizo en el proceso de calibración.

Para seleccionar el valor del umbral adecuado, se usó la técnica de ensayo y error para observar cuál era el que más se adaptaba al caso de este proyecto. Cabe destacar que para hacer este procesamiento, mejorando los tiempos de ejecución, no se usa la

totalidad de la imagen, sólo se trabaja con la zona que encierra el cuadro amarillo como se observa en la figura 4.3. Para realizar este análisis se seleccionan como imágenes las tomadas del proceso de calibración de las figura 4.10 y 4.11, correspondientes a imágenes ojo abierto y ojo cerrado, respectivamente. En las figuras 4.12 y 4.13 se enseñan las regiones de las imágenes que se estudiarán, las cuales corresponden a la captura ojo abierto y ojo cerrado, respectivamente.

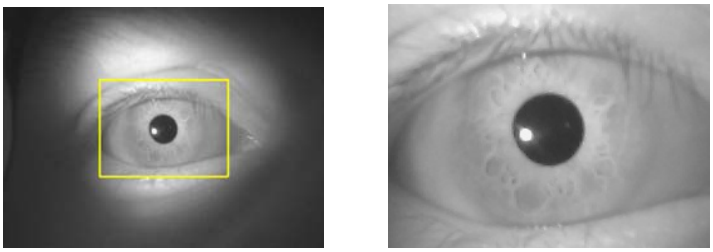

Figura 4.12: Selección de la región para hacer el procesamiento ojo abierto

Figura 4.13: Selección de la región para hacer el procesamiento ojo cerrado

La resta entre la selección de la imagen en la figura 4.13 con la selección de la figura 4.12 da como resultado la imagen que se observa en la figura 4.14. Por otro lado, la resta entre una imagen ojo cerrado y otra ojo abierto genera una notable diferencia. Cuando se hace una resta de dos imágenes ojo cerrado se obtiene una imagen como la que se muestra en la figura 4.15. Se puede observar que dicha resta resultante es una imagen totalmente negra, ya que ambas imágenes se cancelan, lo cual es equivalente a restar dos matrices de igual dimensión con valores iguales que generaría una matriz con valores cero. Partiendo de este punto, se procedió a aplicar la comparación de

histogramas, ya que ellos generan mucho mejor tiempo de respuesta que cuando se hace una comparación entre imágenes.

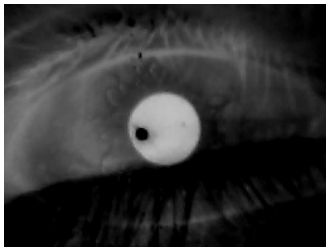

Figura 4.14: Resta entre imágenes ojo cerrado y ojo abierto

Figura 4.15: Resta entre dos imágenes ojo cerrado

Posteriormente, se procedió a aplicar una variación de umbrales a la figura 4.14 para seleccionar el más adecuado. Para aplicar un umbral, tal y como se mencionó anteriormente, se usa `cvThreshold()`. En la figura 4.16 se puede observar imágenes derivadas de aplicar la técnica de umbralización a la imagen resultante de la resta, con diferentes variaciones del valor del umbral. Es notorio que para valores muy bajos del umbral la imagen toma muchas características, generando así una imagen un poco distorsionada, mientras que para valores muy altos se pierden las características relevantes como es la pupila. Dado estos resultados y tomando en cuenta que se necesita una imagen limpia, en la cual no se perdieran características importantes, se decidió tomar el valor del umbral `threshold` igual a 130. Este valor permite eliminar los píxeles no deseados y deja sólo el área de la pupila, la cual funciona muy bien a la hora de hacer la comparación.

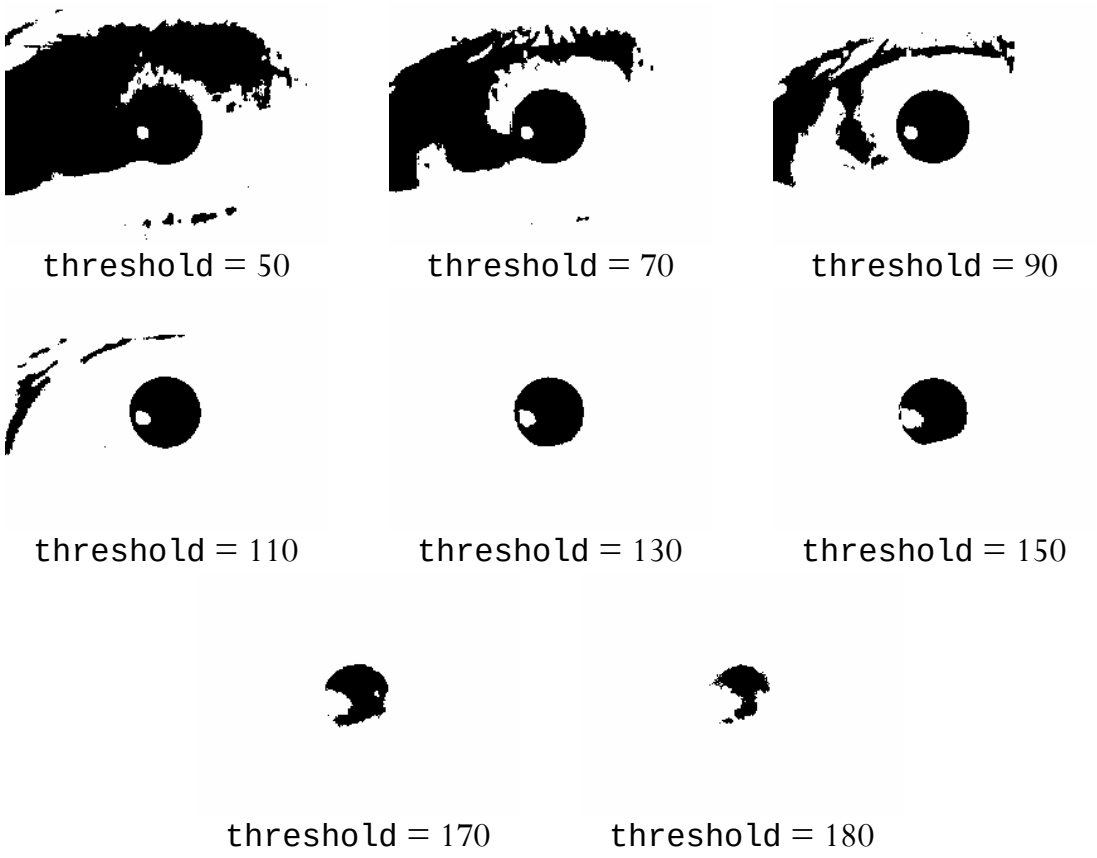

Figura 4.16: Variaciones del umbral en la binarización de la figura 4.14

Una vez implementado lo anteriormente expuesto, se está en capacidad de ejecutar el clic, ya que al tener la imagen binarizada solamente falta generar su histograma para luego compararlo con el histograma de la imagen negra. Luego, usando sentencias de validación y las funciones API correspondientes, se puede generar el clic mediante el movimiento ocular.

4.3 Proceso paso a paso de la ejecución de la herramienta

En esta sección se explica detalladamente el proceso en tiempo de ejecución de la herramienta, comenzando desde el calibrado hasta los movimientos e indicaciones del mismo y del clic. Primero, se obtendrá del proceso de calibración dos imágenes, una correspondiente a ojo abierto y otra a ojo cerrado como se observan en las figuras 4.12 y 4.13, respectivamente. Luego de tomadas dichas imágenes se genera una ventana con una imagen como la que se muestra en la figura 4.17. En la misma se puede observar una cruz de color verde que representa el indicativo del seguimiento de la pupila, y por ende el movimiento del cursor. Además, se refleja un cuadro rojo que indica la zona en la cual el cursor no se moverá si la cruz permanece dentro de esa área. En este sentido, si la cruz se sale del recuadro, el cursor se mueve respecto al lugar al que se haya salido; es decir, que según su ubicación fuera del cuadro rojo el cursor se mueve hacia arriba, abajo, izquierda o derecha. Otro elemento que se observa en la figura antes señalada es un rectángulo amarillo, que representa la zona en la cual se está realizando el estudio. Por lo tanto, si al calibrar la posición del ojo, este queda fuera o parcialmente fuera de el recuadro amarillo, el movimiento no será el esperado. Por último, se tiene un rectángulo azul que es de las mismas dimensiones al usado en el proceso de calibración. Este recuadro se hace con el fin de determinar si el ojo quedó centrado, ya que la imagen se reduce a una resolución menor para efectos de visualización en pantalla.

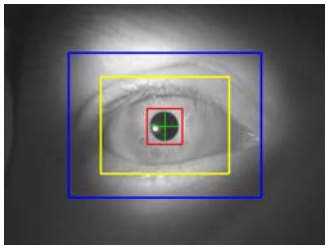

Figura 4.17: Imagen final de adquisición en tiempo real

Posteriormente, el movimiento del ojo se verá reflejado con el movimiento de la cruz de color verde sincronizado con la pupila. Es así que la cruz será el indicador del movimiento y dirección. Adicionalmente y de manera que la herramienta resultase más amigable, se anexó una ventana en la cual se indica con una flecha de determinado color cuándo se está realizando un movimiento específico. En las figuras 4.18, 4.19, 4.20, 4.21 se observan los movimientos reconocidos por la herramienta, en la cual se visualiza la imagen de la flecha y su color para indicar la dirección del cursor. Cabe destacar que la imagen de la flecha es mostrada usando una ventana que se encuentra ubicada en la esquina inferior derecha de la pantalla del computador.

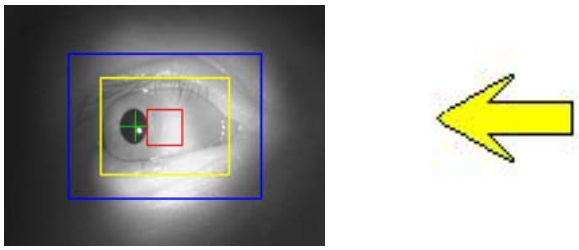

Figura 4.18: Movimiento del ojo hacia la izquierda e indicación del sentido con una flecha amarilla

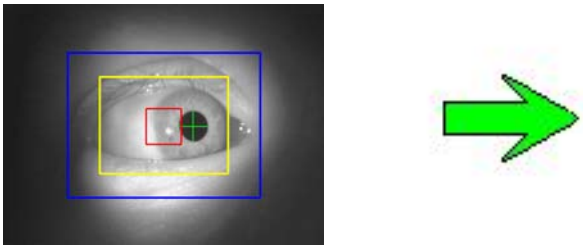

Figura 4.19: Movimiento del ojo hacia la derecha e indicación del sentido con una flecha verde

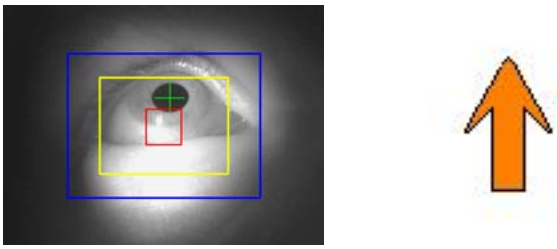

Figura 4.20: Movimiento del ojo hacia arriba e indicación del sentido con una flecha anaranjada

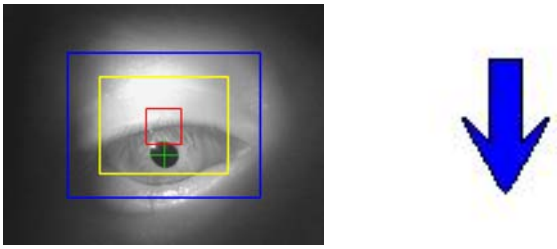

Figura 4.21: Movimiento del ojo hacia abajo e indicación del sentido con una flecha azul

En la figura 4.22 se muestra la acción al hacer el clic, lo cual se genera cuando el usuario mantiene el ojo cerrado por un tiempo estimado de un segundo. Al ocurrir un clic aparece en la ventana indicadora un recuadro color negro. En caso de querer ejecutar un doble clic se tendrá que tener el ojo cerrado por un periodo mayor a 1 segundo y aparecerá en la ventana indicadora un recuadro color rojo.

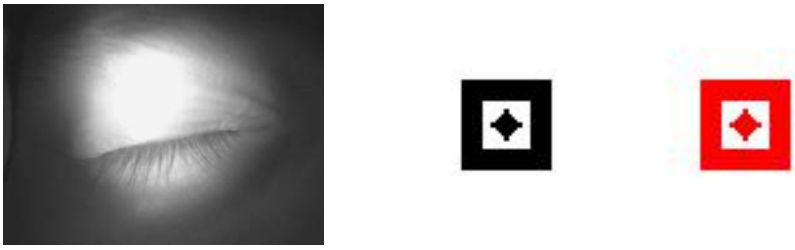

Figura 4.22: Cerrado del ojo e indicación de la ejecución del clic y el doble clic

4.4 Representación del algoritmo mediante Redes de Petri

Las Redes de Petri (RP) son una herramienta matemática que puede utilizarse para el modelado de sistemas de diversa naturaleza. La misma fue desarrollada por Carl Adam Petri en su tesis doctoral como un modelo de propósito general para la descripción de las relaciones existentes entre condiciones y eventos. De manera general, la herramienta consiste en un grafo orientado formado por dos tipos de nodos, los lugares y las transiciones, los cuales están unidos alternativamente por arcos orientados. Cada lugar representa el estado al que puede llegar el sistema, mientras que la transición indica la posibilidad de que ocurra un evento que altere el estado del sistema. Por otro lado, los arcos, representados por líneas rectas dirigidas, se utilizan para unir los estados con transiciones y viceversa (García, 2004).

En la figura 4.23 se muestra el sistema de la herramienta desarrollada mediante una Red de Petri y en las tablas 4.1 y 4.2 la descripción de los estados y transiciones, respectivamente.

Estado	Descripción
L1	Ventana de bienvenida.
L2	Verificación de la variable Capture, para validar si se está capturando la imagen de la cámara web.
L3	Muestra una ventana de advertencia que indica que no tiene cámara conectada al computador.
L4	Muestra una ventana en la cual se está capturando en Tiempo Real (TR) la imagen que se obtiene de la cámara web, esta ventana se usa para hacer el proceso de calibración ojo abierto.
L5	Muestra una ventana en la cual se está capturando en TR la imagen que se

	obtiene de la cámara web, esta ventana se usa para hacer el proceso de calibración ojo cerrado.
L6	Muestra una ventana en la cual se esta ejecutado en TR el algoritmo para detectar el centroide la pupila, la región de movimiento y cuando se quiere hacer un clic o doble clic.
L7	Muestra menú que se despliega al hacer clic derecho sobre el icono de la aplicación, en donde se observan dos opciones, la de ajustar la sensibilidad y la de cerrar la aplicación.
L8	Activa las funciones API correspondientes para realizar el clic o doble clic.
L9	Activa función API para realizar el movimiento del cursor hacia la izquierda.
L10	Activa función API para realizar el movimiento del cursor hacia la derecha.
L11	Activa función API para realizar el movimiento del cursor hacia arriba.
L12	Activa función API para realizar el movimiento del cursor hacia abajo.
L13	Muestra el menú que se despliega al hacer clic sobre ajustar sensibilidad, en el cual se pueden ajustar la sensibilidad del movimiento del cursor, del tamaño del cuatro rojo y de la rapidez del doble clic.
L14	Observa la bandera de validación que verifica si se culminó el proceso de calibración, al culminar el proceso de calibración la bandera se coloca en uno, en caso contrario en cero. Este estado se activa cuando se elige la opción de ajustar sensibilidad del movimiento del cursor.
L15	Observa la bandera de validación que verifica si se culminó el proceso de calibración, al culminar el proceso de calibración la bandera se coloca en uno, en caso contrario en cero. Este estado se activa cuando se elige la

	opción de ajustar el tamaño del cuadro rojo.
L16	Observa la bandera de validación que verifica si se culminó el proceso de calibración, al culminar el proceso de calibración la bandera se coloca en uno, en caso contrario en cero. Este estado se activa cuando se elige la opción de ajustar sensibilidad del doble clic.
L17	Incluir barra de graduación en la ventana llamada Tiempo Real, para ajustar la sensibilidad del movimiento del cursor.
L18	Incluir barra de graduación en la ventana llamada Tiempo Real, para ajustar el tamaño del cuadro rojo.
L19	Incluir barra de graduación en la ventana llamada Tiempo Real, para ajustar la sensibilidad del doble clic.
L20	Muestra una ventana de advertencia la cual indica que para poder activar la opción de ajustar sensibilidad del movimiento del cursor, tiene que haber culminado el proceso de calibración.
L21	Muestra una ventana de advertencia la cual indica que para activar la opción de ajustar el tamaño del cuadro rojo, tiene que haber culminado el proceso de calibración.
L22	Muestra una ventana de advertencia la cual indica que para activar la opción de ajustar sensibilidad del doble clic, tiene que haber culminado el proceso de calibración.

Tabla 4.1: Descripción de los estados de la Red de Petri

Transición	**Descripción**
T1	Si presiona Iniciar.
T2	Si Capture == 0.
T3	Si Capture ≠ 0.

T4	Si presiona Aceptar.
T5	Si presiona cualquier tecla.
T6	Si presiona cualquier tecla.
T7	Si presiona cualquier tecla.
T8	Si se capturan imágenes similares a la almacenada en proceso de calibración de ojo cerrado, por un tiempo aproximado de un segundo.
T9	Si cruz verde que indica el centroide de la pupila no sale del cuadro rojo.
T10	Si cruz verde que indica el centroide de la pupila sale del cuadro rojo por el lado izquierdo.
T11	Si cruz verde que indica el centroide de la pupila sale del cuadro rojo por el lado derecho.
T12	Si cruz verde que indica el centroide de la pupila sale del cuadro rojo por el lado de arriba.
T13	Si cruz verde que indica el centroide de la pupila sale del cuadro rojo por el lado de abajo.
T14	Si se ejecutó el clic.
T15	Si el movimiento fue realizado.
T16	Si se ejecuta clic derecho sobre el icono de la aplicación que se encuentra en la barra de Windows.
T17	Si presiona clic en cerrar aplicación.
T18	Si presiona clic en ajustar sensibilidad.
T19	Si presiona clic en ajustar sensibilidad del movimiento del cursor.
T20	Si presiona clic en ajustar el tamaño del cuadro rojo.
T21	Si presiona clic en ajustar sensibilidad del doble clic.

T22	Si bandera == 1.
T23	Si bandera == 0.
T24	Si bandera == 1.
T25	Si bandera == 0.
T26	Si bandera == 1.
T27	Si bandera == 0.
T28	Si se incluye la barra para ajustar sensibilidad del movimiento del cursor en la ventana llamada Tiempo Real.
T29	Si se incluye la barra para ajustar el tamaño del cuadro rojo en la ventana llamada Tiempo Real.
T30	Si se incluye la barra para ajustar sensibilidad del doble clic en la ventana llamada Tiempo Real.
T31	Si presiona Aceptar, regresa a la ventana de Bienvenida.
T32	Si presiona Aceptar, regresa a la ventana de Calibración ojo abierto.
T33	Si presiona Aceptar, regresa a la ventana de Calibración ojo cerrado.

Tabla 4.2: Descripción de las transiciones de la Red de Petri

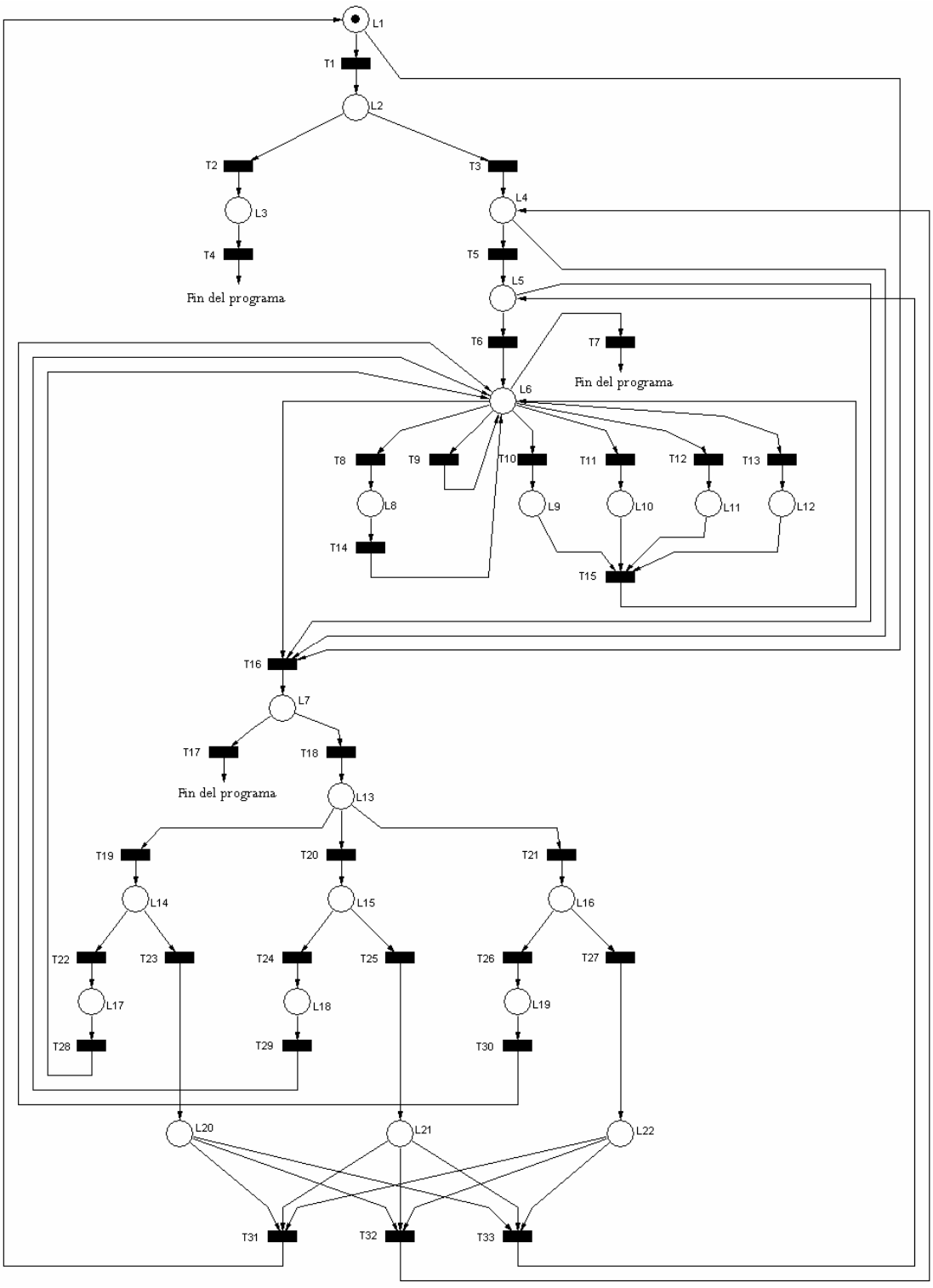

Figura 4.23: Representación del algoritmo mediante una Red de Petri

4.5 Resultados de las pruebas

Esta es la fase que consume mayor tiempo y consiste en probar el desempeño adecuado de la herramienta. Se realizaron diversas pruebas para determinar que el mismo es cómodo y permite con una interfaz amigable, que el usuario pueda interactuar sin complicaciones.

En el proceso de pruebas continuas realizadas a la herramienta se notó que la colocación de la gorra debía ser en un ángulo adecuado, de manera que las pestañas no bloqueen la zona del ojo, ya que esto influiría en el tratado de la imagen. Por lo tanto, se decidió que la colocación de la gorra al usuario debía hacerse de manera tal que la cámara junto con el ángulo de inclinación de la visera de la gorra, capte la zona de interés sin interrupción.

La herramienta fue probada con personas de ojos claros y ojos oscuros para observar si para cada caso, la misma funcionaba correctamente. De hecho, el programa resultó funcionar correctamente en ambos casos. Sin embargo, se pudo apreciar que esto no ocurría cuando el usuario usaba maquillaje oscuro, ya que este fue realizado para detectar la zona más oscura de la imagen a tratar y se da el caso que el maquillaje oscuro hace que se resalte otra zona y por ende hace que el sistema no funcione correctamente. Luego, la herramienta se probó con personas con maquillaje claro y la herramienta funcionó correctamente. Dado ese hecho se decidió colocar como condición que para usar la herramienta, la persona no debe usar maquillaje oscuro.

En un inicio, el proceso de calibración se tornó muy tedioso ya que cada vez que se iniciara la herramienta, el usuario tenía que mirar a cada esquina de la pantalla para dibujar el recuadro que delimita la zona de movimiento. Debido a esto se decidió tomar solamente dos imágenes en el proceso de calibración (ojo abierto y ojo cerrado) y, por defecto, establecer los valores que dibujan dicho recuadro. Sin embargo, para hacer la

herramienta más sensible a los diferentes usuarios y sus respectivos requisitos, se agregó la posibilidad de graduar esta región al gusto de cada usuario.

Adicionalmente, se decidió agregar la opción para que el usuario pudiese regular la velocidad del cursor. Esto tomando en cuenta las diferentes agilidades que pueden variar entre personas con capacidades reducidas. De esta manera, el proceso se puede hacer más lento o más rápido, de acuerdo a la comodidad de quien esté en disposición de usar la herramienta. Del mismo modo se realizó con la sensibilidad del doble clic, ya que dependiendo de la necesidad del usuario, esté tendría que ejecutarse más o menos rápido.

4.6 Factibilidad económica de la herramienta

El factor costo juega un papel importante en las aplicaciones similares existentes. Por lo tanto, es fundamental verificar la rentabilidad que tiene el sistema desarrollado y enfatizar en los beneficios que este trae consigo.

Uno de los objetivos del estudio de la factibilidad económica es estimar los costos asociados al desarrollo e implantación de la herramienta creada. Para esto, es esencial hacer énfasis en el hecho de que no se incluyen los gastos asociados al *hardware* y *software* básicos con el cual funciona dicha herramienta, debido a que esta inversión no ha de ser considerada. En este sentido, se hace una estimación del costo asociado a lo que es la creación del periférico visual alternativo, fundamentándose en que ya el interesado tiene como base una computadora personal con los requisitos mínimos establecidos, en cuanto a sus componentes y al sistema operativo instalado.

Tomando en cuenta todos los costos implicados en el desarrollo de la herramienta la misma tiene un costo estimado de 1515.85 bolívares fuertes, equivalente a 533.38€. En este costo se contempla el gasto por la licencia de Microsoft Visual Studio 2005 que tiene un costo de $199, correspondiente a 427,85 bolívares fuertes aproximadamente.

Además, se incluye el costo de una cámara web Genius Eye 312, la cual cuesta alrededor de 88 bolívares fuertes. Adicionalmente, se incluye el costo de la programación de la herramienta que fue estimada en 900 bolívares fuertes y el costo de las modificaciones a la cámara web y la creación del dispositivo de iluminación, estimados ambos en 100 bolívares fuertes.

Resulta interesante hacer un análisis comparativo del costo total estimado de la herramienta creada respecto al de otras herramientas que se encuentran en el mercado, las cuales tienen fines similares al de este proyecto. Para esto, se muestra en la tabla 4.3 los costos mencionados. Es notorio el bajo costo que tiene la aplicación que se presenta y, con esto, es indiscutible el gran potencial a réplica que tiene consigo el sistema desarrollado.

Herramienta	**Costo**
Iriscom QG2	6800 €
Iriscomo QG3	7800 €
Eye Mouse	2800 €
Sistema *I4Control*	1600 €
Nuevo periférico visual alternativo	1515.85 Bs. F. ≅ 533.38 €

Tabla 4.3: Costos de herramientas similares

Capítulo 5

Conclusiones y recomendaciones

5.1 Conclusiones

En este proyecto se tuvo la oportunidad de estudiar conceptos, técnicas y aplicaciones de la Visión Artificial, la cual es una rama de la ciencia de gran importancia en el área de la tecnología, que se está utilizando en la actualidad con mucho éxito en diversos ámbitos. El desarrollo de una herramienta que permite mover el cursor de un computador mediante el movimiento ocular, utilizando técnicas de Visión Artificial, permite concluir que:

- Los conocimientos básicos de programación resultaron fundamentales para lograr el éxito del desarrollo de la herramienta planteada, debido a que éstos brindaron la posibilidad de implementar una serie de funciones y procedimientos necesarios para lograr el desarrollo de la misma.
- La fase de pruebas del sistema fue de gran importancia ya que permitió hacer cambios y mejoras para que el usuario final pueda hacer uso de la herramienta sin inconvenientes. Además, hizo posible corroborar el logro de los objetivos propuestos.
- El prototipo construido brinda a personas con capacidades reducidas la posibilidad de interactuar con el mundo exterior, con lo cual podrán valerse por sí mismos para realizar diversas tareas. En este sentido, la herramienta desarrollada permite mejorar la calidad de vida de dichas personas y, con esto, ayudar a motivarlas y elevar su autoestima.

- El sistema desarrollado posee una gran rentabilidad económica, resultando mucho menos costoso que productos similares que se encuentran en el mercado en la actualidad. Esto permite afirmar que el proyecto tiene un gran potencial a réplica.
- El desarrollo del proyecto fue un éxito ya que se lograron cumplir los objetivos planteados, finalizando satisfactoriamente la elaboración de la herramienta previamente planteada en el tiempo establecido.

5.2 Recomendaciones

- Ampliar la herramienta de manera que sirva para la ejecución de funciones adicionales que brinda un ratón tradicional para PC que no se contemplaron en la herramienta desarrollada. Entre éstas se encuentran: el clic derecho, selección de una región, mover y arrastrar, etc.
- Modificar la herramienta de manera que una persona pueda realizar otras tareas como manejar una silla de rueda, prender la luz, el televisor, radio, etc.

Referencias bibliográficas

Asociación de Esclerosis Lateral Amiotrófica (2007), *Iriscom*. San Sebastián, España. [Documento WWW], recuperado: http://www.iriscom.org/

Asociación Larense de Astronomía (2009), *El ojo humano*. Barquisimeto, Venezuela. [Documento WWW], recuperado: http://www.tayabeixo.org/que_obs/ojo.htm

Bradski, G. y Kaehler, A. (2008), *Learning OpenCV*. Primera edición. Editorial O'Reilly Media, Inc. California, Estados Unidos.

Caño, C. (2008), *PDI de bajo coste*. Madrid, España. [Documento WWW], recuperado: http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=583

Deitel, H. y Deitel P. (1999), *C++ Cómo programar*. Segunda edición, Prentince Hall. México.

Departamento de Cibernéticas (2004), *System I4Control* , Departamento de Cibernética, Facultad de Ingeniería Eléctrica. Universidad Técnica Checa. Praga, República Checa. [Documento WWW], recuperado: http://cyber.felk.cvut.cz/i4c/en_system.html

End of the World Production, LLC (2007), *IR QuickCam*. Nueva York, Estados Unidos. [Documento WWW], recuperado: http://www.theparticle.com/hardware/irqcam/

García, E. (2004), *Automatización de Procesos Industriales*. Alfa Omega. España.

Gips, J. y Betke, M. (2009), *Camera Mouse*. Boston Collage. Boston, Estados Unidos. [Documento WWW], recuperado: http://www.cameramouse.org/

González, R. y Woods, R. (1996), *Tratamiento digital de imágenes.* Primera edición, Editorial Addison-Wesley/Diaz de Santos. Madrid, España.

Health System (2008), *La Medicina y la rehabilitación*. Universidad de Virginia, [Documento WWW], recuperado: http://www.healthsystem.virginia.edu/UVAHealth/adult_pmr_sp/spcrd.cfm

Palleja, T. y Rubion, E. (2007), *HeadMouse.* Grupo de Robótica, Universidad de Lleida. Lleida, España. [Documento WWW], recuperado: http://robotica.udl.cat/

Piedra, R. (2001), *Ingeniería de la Automatización Industrial*. Alfa Omega. España.

Platero, C. (2005), *Apuntes de visión artificial*. Departamento de Electrónica, Automática e Informática Industrial de la Universidad Politécnica de Madrid, España. [Documento WWW], recuperado: http://www.elai.upm.es/spain/Asignaturas/Robotica/InfoRobotica#Examenes

Real Academia Española (2001), *Diccionario de la lengua española,* Vigésima segunda edición, Real Academia Española. Madrid, España.

Sample, I. (2005), *Un tetrapléjico logra mover un brazo robótico con un chip en su cerebro*, [Documento WWW], recuperado: http://www.elmundo.es/2005/04/01/ciencia/1777617.html

Scott, S. (2006), 'Neuroscience: Converting thoughts into actions', *Nature* 442, pp. 141-142, United Kingdom.

Tchalenko, J. (2000), *The Eye Mouse Proyect*. Camberwell College of Arts. Londres, Inglaterra. [Documento WWW], recuperado: http://www.arts.ac.uk/research/eyemouse

Turégano, E. (2006), 'EyeBoard: Un Periférico Alternativo Visual'. [Documento WWW], recuperado: http://robolab.unex.es/research/doc/libro.pdf

Ward, D. y MacKay, D. (2002), 'Artificial intelligence: Fast hands-free writing by gaze direction', *Nature* 418, p. 838, United Kingdom.

ANEXOS

UNIVERSIDAD DE LOS ANDES

Manual de usuario

Autor: Br. Francisco A. Justo T.

Facultad de Ingeniería

Mérida Estado Mérida, Venezuela

INFORMACIÓN GENERAL

Características generales

La herramienta que se presenta fue denominada VisionClic ya que ella permite interactuar con el computador utilizando la visión. De ahora en adelante para referirnos a la herramienta lo haremos usando su nombre de aplicación VisionClic.

Como requisitos mínimos de sistema, para que VisionClic funcione moderadamente, se necesita un computador Pentium 4 de 1.5 GHz, 512 Mb de RAM, acceso a puerto USB, 5 Mb de espacio en disco para la instalación, SO Windows XP con Service Pack 2. Es importante destacar que con un equipo de mayor capacidad funciona mucho mejor. También requiere una cámara web, para realizar las modificaciones y capturar la zona del ojo, la resolución óptima de la cámara deberá ser 640x480 y 1.3 MP. La cámara web recomendada es una Genius Eye 312 o versiones similares.

ACCESAR AL SISTEMA VisionClic

Luego de instalar VisionClic en el computador, se crea un acceso directo en el escritorio, así como también un acceso en el menú Inicio de Windows XP, el icono es como el que se muestra en la figura 1. Haciendo doble clic sobre este, aparecerá una ventana de bienvenida.

Figura 1: Icono VisionClic

VENTANA DE BIENVENIDA

En esta ventana se explica para qué está diseñado. Adicionalmente, indica que luego de presionar el botón Iniciar se comenzará el proceso de calibración. En la figura 2, se observa cómo es dicha ventana.

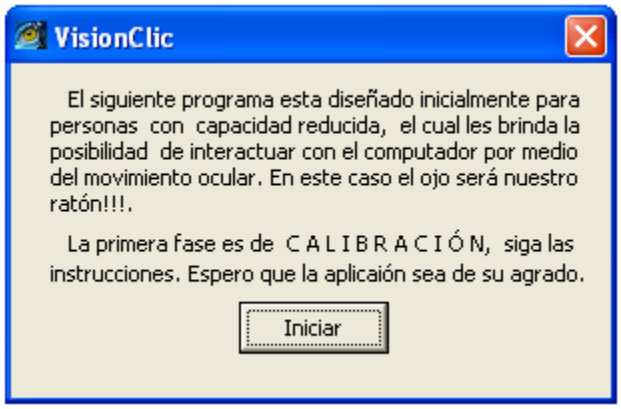

Figura 2: Ventana de bienvenida

Si se presiona con el clic derecho sobre la imagen que se encuentra en la esquina superior izquierda, se desplegara un menú como el que se muestra en la figura 3.

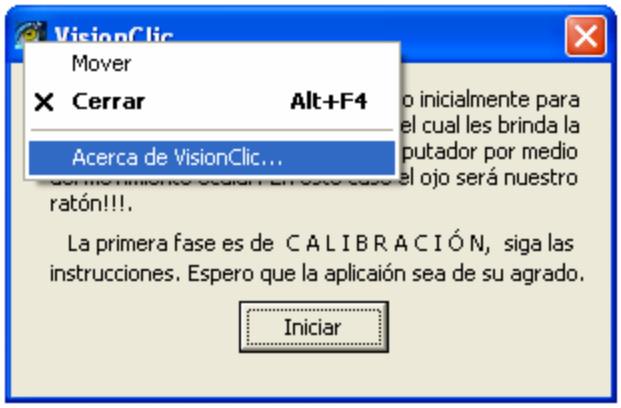

Figura 3: Menú desplegado al presionar clic derecho sobre la imagen icono

Al seleccionar la opción Acerca de VisionClic…, se abrirá otra ventana en la cual se indica la versión, el año de creación y el autor. Esto se observa en la figura 4.

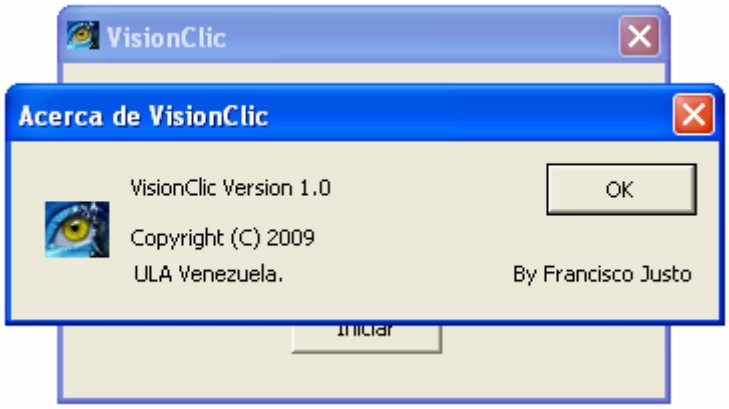

Figura 4: Ventana de información sobre VisionClic

SELECCIÓN DE DISPOSITIVO DE VIDEO

Luego de presionar el botón Iniciar, si la computadora tiene más de una cámara web conectada, aparecerá una ventana en la cual podrá seleccionar el dispositivo de video que se usará para la captura de imágenes. Si ya se seleccionó un dispositivo de video para

la captura, pero luego se quiere cambiar por otro, se deberá iniciar la sesión con este dispositivo y hacer doble clic sobre el icono de VisionClic que parece en la barra de tareas. Esto abrirá otra ventana de bienvenida, en la cual deberá presionar el botón Iniciar y se desplegará la opción en la cual podrá seleccionar el dispositivo de video deseado. En figura 5 se muestra un ejemplo de lo mencionado.

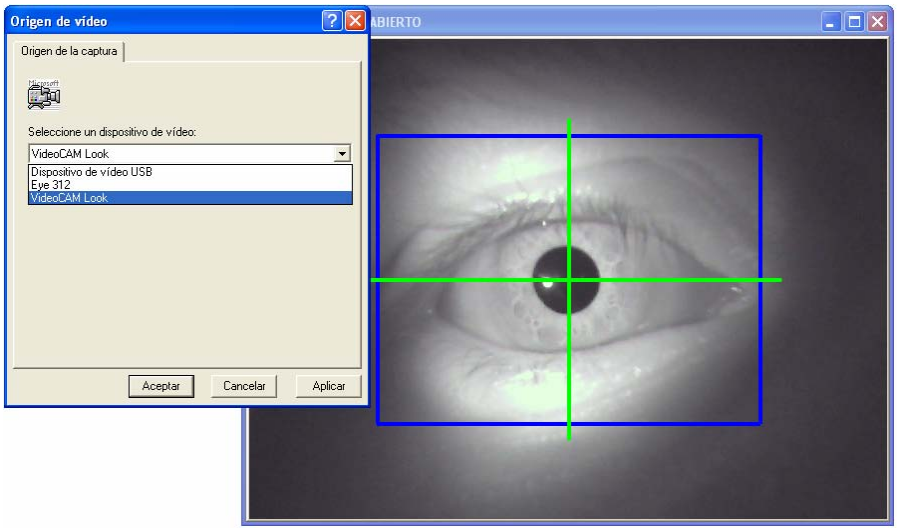

Figura 5: Ejemplo de selección de dispositivo de video

CALIBRACIÓN

Luego de presionar el botón Iniciar y haber seleccionado el dispositivo de video deseado, se despliega la primera ventana de calibración, la cual se usa para capturar la imagen ojo abierto. En este punto se debe centrar el ojo del usuario de tal forma que la pupila quede lo más centrado posible en la cruz y el recuadro que se pinta en la imagen. Una vez logrado esto, se presiona cualquier tecla para continuar. La figura 6 muestra un ejemplo en el cual es centrado el ojo del usuario en esta ventana.

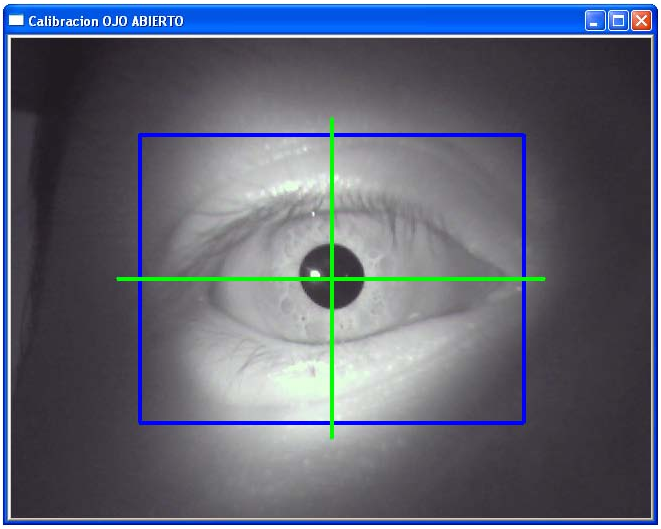

Figura 6: Ejemplo de calibración ojo abierto

La siguiente ventana que se despliega es usada para capturar la imagen ojo cerrado que será usada para la ejecución del clic y doble clic. Cabe destacar que el usuario debe tener en cuenta que para poder ejecutar el clic deberá cerrar el ojo de igual formo como lo hizo al capturar esta imagen, ya que si lo hace de forma diferente es posible el clic no se genere. En la figura 7 se muestra un ejemplo de la ventana en captura de la imagen ojo cerrado.

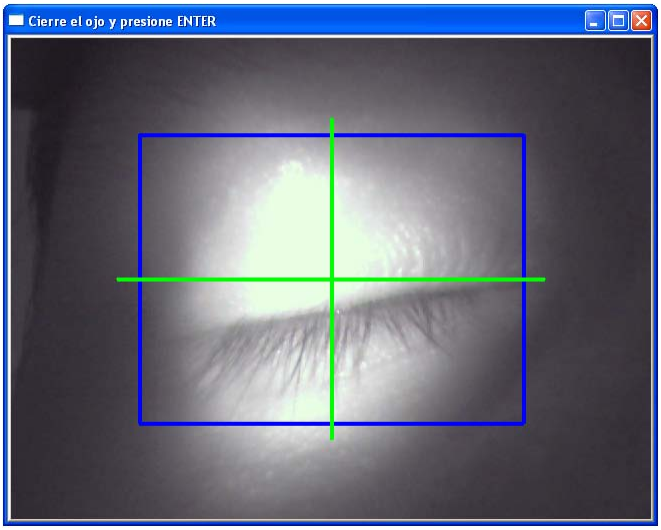

Figura 7: Ejemplo de calibración ojo cerrado

AJUSTE DE PARÁMETROS DE SENSIBILIDAD

Luego de culminar el proceso de calibración, se está en capacidad de realizar el movimiento del cursor, así como también el clic o doble clic, con los valores predeterminados que posee VisionClic. En la figura 8 se muestra un ejemplo de cómo sería la ventana final con los valores predeterminados.

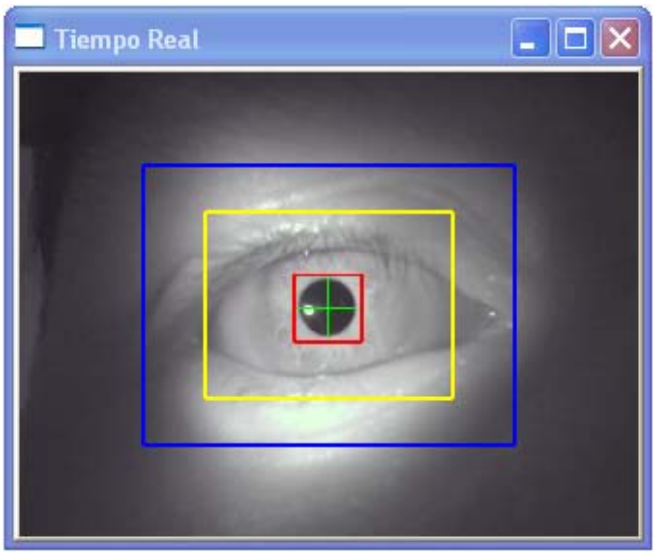

Figura 8: Ejemplo de ventana que muestra la detección de la pupila en tiempo real

Adicionalmente, se pueden modificar los valores de ciertos parámetros de sensibilidad usando el botón derecho del *mouse* y presionado sobre el icono de VisionClic que muestra en la barra de tareas de Windows, para seleccionar el tipo de cambio. En la figura 9 se observa el icono de VisionClic en la barra de tareas de Windows y en la figura 10 la ejecución del clic derecho sobre este icono.

Figura 9: Barra de tareas de Windows donde se observa el icono de VisionClic

Figura 10: Menú al presionar clic derecho sobre el icono de VisionClic

Al presionar clic derecho se muestra un menú que tiene dos opciones, la primera para ajustar la sensibilidad y la segunda para cerrar VisionClic. Si se presiona clic sobre Ajustar Sensibilidad se despliega un submenú en el cual aparecen tres opciones, como se observa en la figura 11.

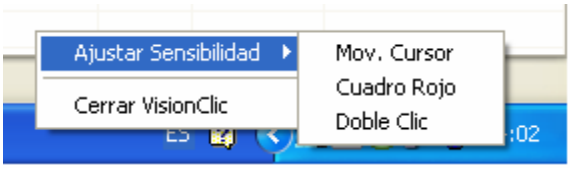

Figura 11: Submenú al presionar clic sobre Ajustar Sensibilidad

Si se presiona clic sobre Mov. Cursor, se mostrará en la ventana Tiempo Real una barra para ajustar la sensibilidad de la velocidad de movimiento del cursor. En la figura 13 se muestra un ejemplo en el cual aparece la barra para ajustar la sensibilidad del cursor.

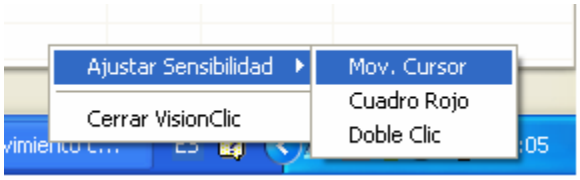

Figura 12: Selección de opción Mov. Cursor

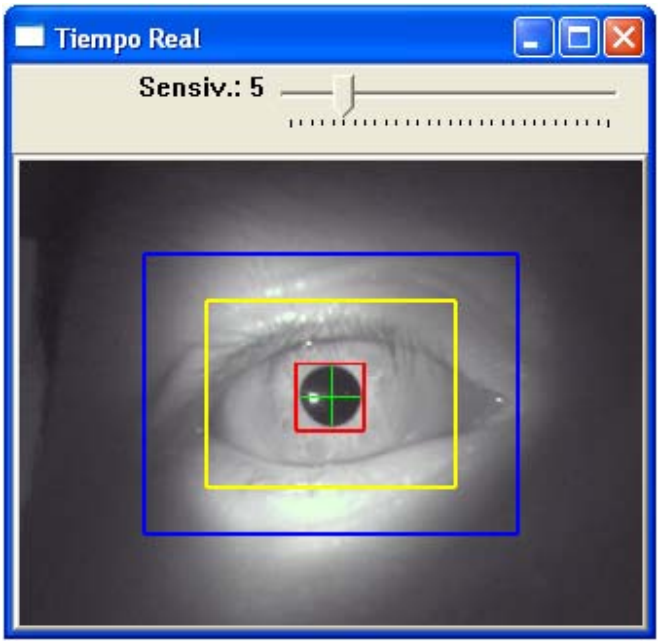

Figura 13: Ejemplo que muestra la barra para ajustar la velocidad del movimiento del cursor

Si se presiona clic sobre Cuadro Rojo, se mostrará en la ventana Tiempo Real una barra para ajustar el tamaño del cuadro rojo, con lo cual se puede ajustar la sensibilidad para mover el cursor, al realizar un movimiento con el ojo. En la figura 15 se muestra un ejemplo en el cual aparece la barra para ajustar el tamaño del cuadro rojo.

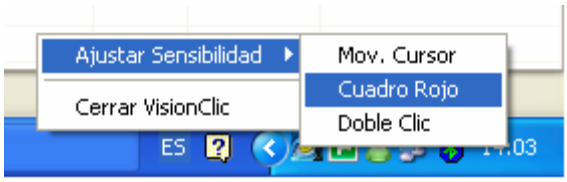

Figura 14: Selección de opción Cuadro Rojo

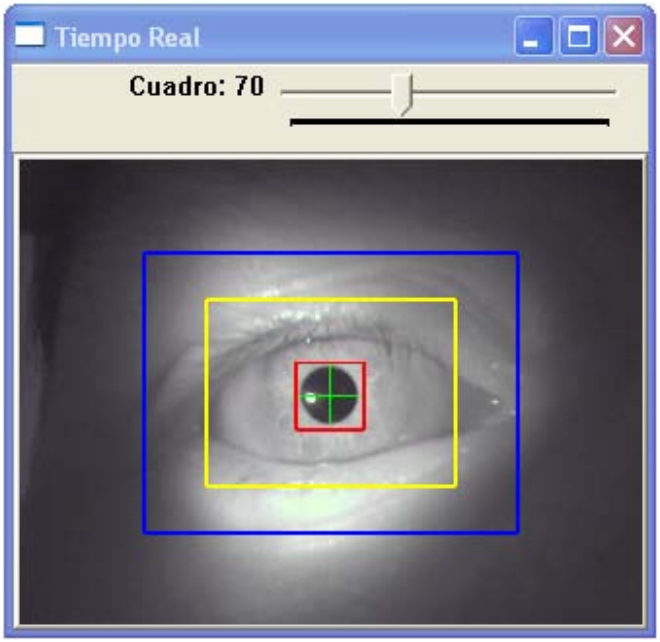

Figura 15: Ejemplo que muestra la barra para ajustar la sensibilidad para mover el cursor al realizar un movimiento con el ojo

Si se presiona clic sobre la opción Doble Clic, se mostrará en la ventan Tiempo Real una barra para ajustar la sensibilidad para generar el doble clic, en ella se puede disminuir el tiempo hasta si se quiere hacer doble clic al mismo tiempo que se genere el clic normal. En la figura 17 se muestra un ejemplo en el cual aparece la barra para ajustar la sensibilidad del doble clic.

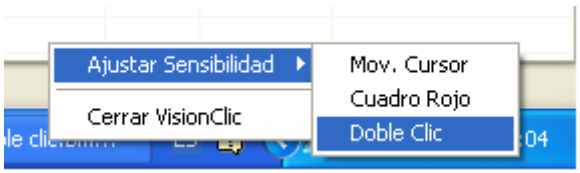

Figura 16: Selección de opción Doble Clic

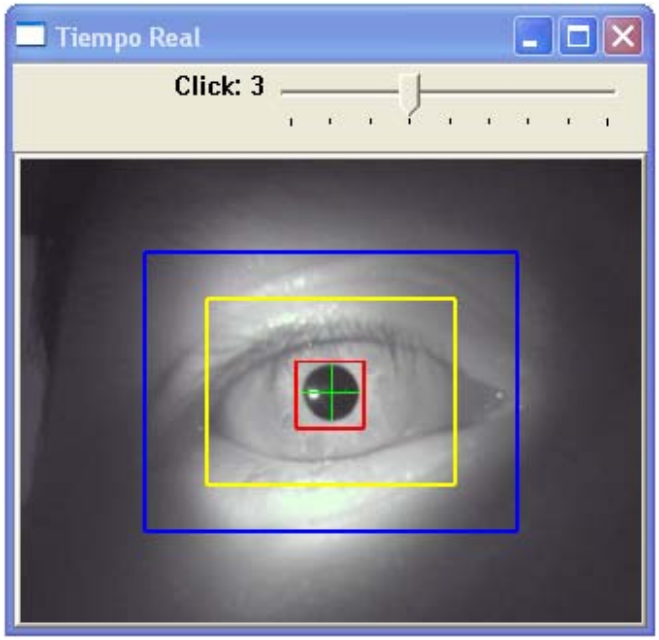

Figura 17: Ejemplo que muestra la barra para ajustar la sensibilidad del doble clic

EJECUCIÓN DE CLIC Y DOBLE CLIC

Al momento de aplicar un clic se debe mantener cerrado el ojo aproximadamente un segundo, que como se dijo, debe capturarse en el proceso de calibración una imagen ojo cerrado, como se observa en la figura 7, para que el clic pueda generarse sin inconvenientes. En la figura 18 se observa un ejemplo en el que se genera un clic, como se puede ver en la ventana de tiempo real se desaparecen los recuadros pintados sobre esta, así como también es indicado que se está ejecutando el clic en una ventana que se muestra en la esquina inferior derecha de la pantalla. En esta ventana se ve una imagen de un recuadro color negro que indica que se realizó el clic.

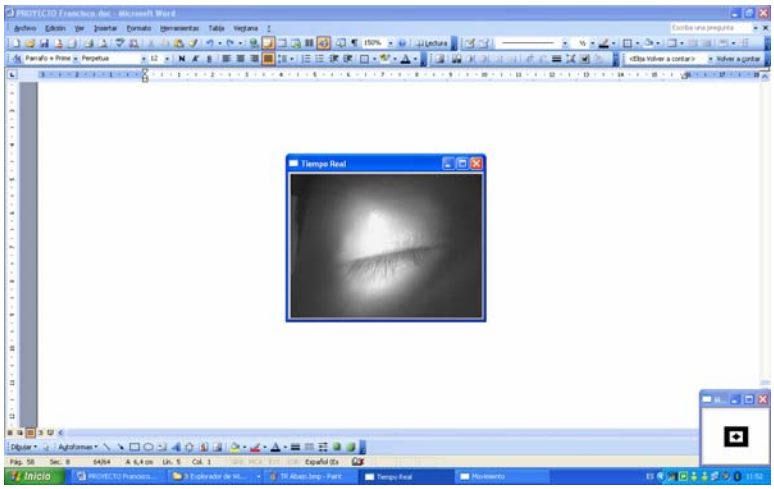

Figura 18: Ejemplo de la ejecución de un clic

Cuando se desea hacer un doble clic, se debe mantener cerrado el ojo un tiempo mayor a un segundo aproximadamente, esto dependerá del ajuste que el usuario haya realizado al mismo. En la figura 19 se muestra un ejemplo de la ejecución del doble clic, en el cual se observa al igual que en el caso del clic, que se desaparecen los objetos pintados en la imagen y aparecerá en la esquina inferior derecha un ventana que contiene una imagen de un recuadro rojo que indica que se realizó el doble clic.

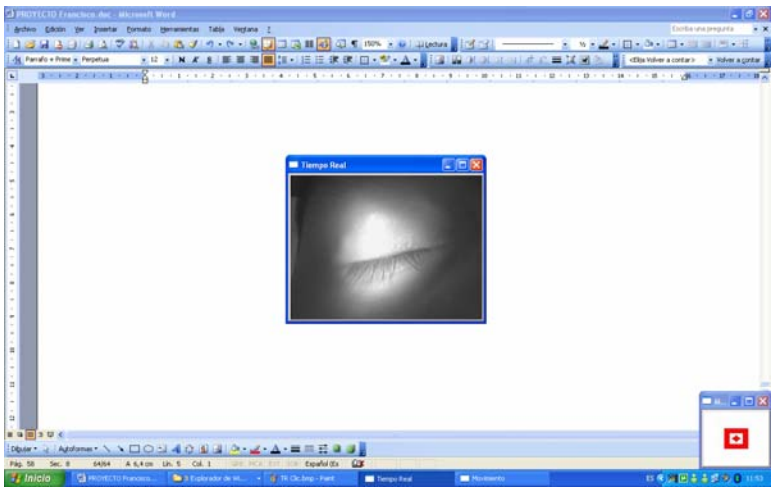

Figura 19: Ejemplo de la ejecución del doble clic

MOVIMIENTOS DEL CURSOR

En las siguientes figuras se mostrarán las diferentes direcciones que puede tomar el cursor al hacer el movimiento ocular, así como también ejemplos que ilustran como es realizado cada uno de ellos. Cabe destacar que para realizar el movimiento, la cruz verde que está pintada en el centro de la pupila deberá salir del cuadro rojo, dependiendo del lado que salga, el movimiento se hará en esa dirección. Si la cruz permanece dentro del cuadro rojo el cursor no se moverá. En las figuras 20, 21, 22, 23 se muestran ejemplos de cómo serían los movimientos izquierdo, derecho, arriba y abajo, respectivamente. Dependiendo de la dirección en que se esté haciendo el movimiento, se mostrará en una ventana ubicada en la esquina inferior derecha de la pantalla, una imagen de una flecha que indica la dirección del movimiento. Para cada dirección la flecha tendrá un color específico.

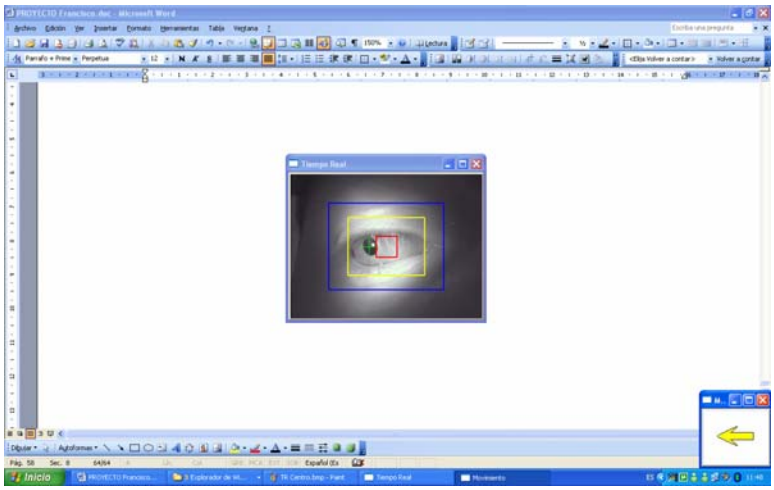

Figura 20: Ejemplo del movimiento del cursor hacia la izquierda

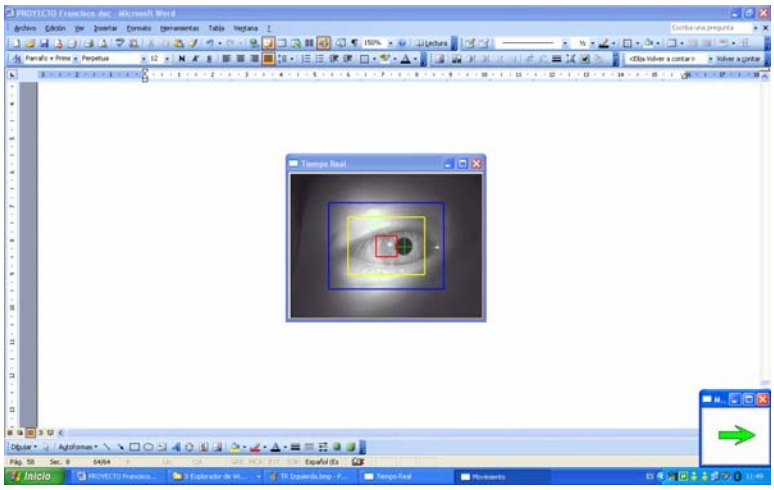

Figura 21: Ejemplo del movimiento del cursor hacia la derecha

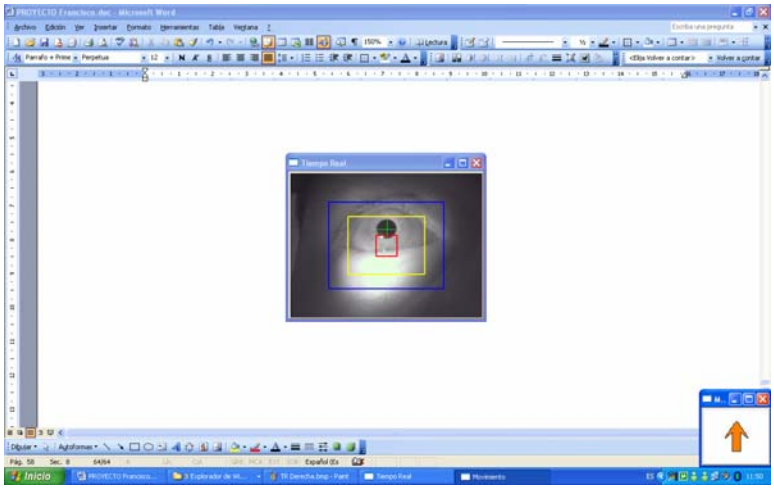

Figura 22: Ejemplo del movimiento del cursor hacia arriba

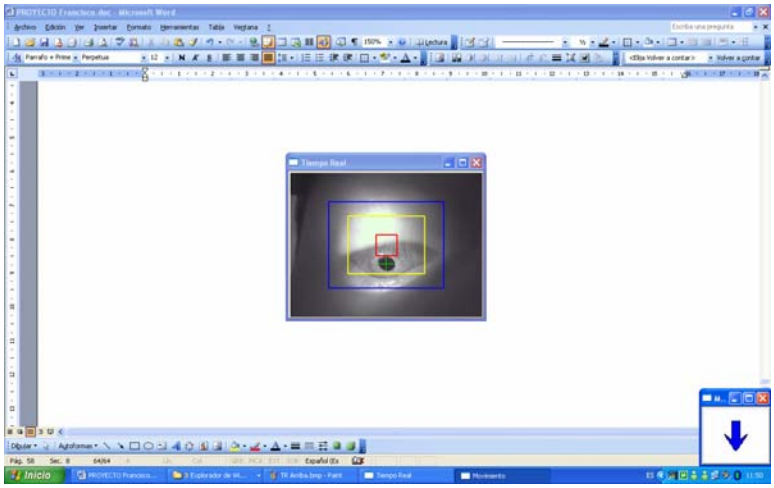

Figura 23: Ejemplo del movimiento del cursor hacia abajo

VENTANAS DE ADVERTENCIA AL SELECCIONAR OPCIONES NO VÁLIDAS

En diversas ocasiones en las que se seleccionan opciones que generan errores, VisionClic mostrará una ventana de advertencia indicando el por qué del error. En la figura 24 se muestra una ventana que indica que la cámara web está mal instalada o desconectada del computador, por lo que no reconoce el periférico de video disponible para la captura de imágenes.

Figura 24: Advertencia al no encontrar dispositivo de video conectado al computador

Cuando se intenta iniciar la captura de video con un dispositivo que ya está siendo usado por VisionClic, se genera la advertencia mostrada en la figura 25.

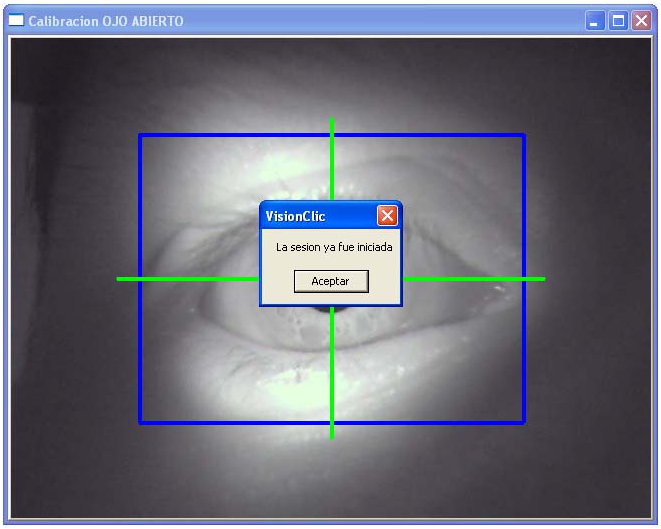

Figura 25: Advertencia cuando se intenta iniciar otra sesión de video teniendo una ya iniciada

Como se explicó anteriormente, se puede regular la sensibilidad de varios parámetros, luego de haber culminado el proceso de calibración. Si se trata de acceder a ellos sin haber culminado el proceso de calibración, VisionClic mostrará una ventana de advertencia como la que se observa en la figura 26.

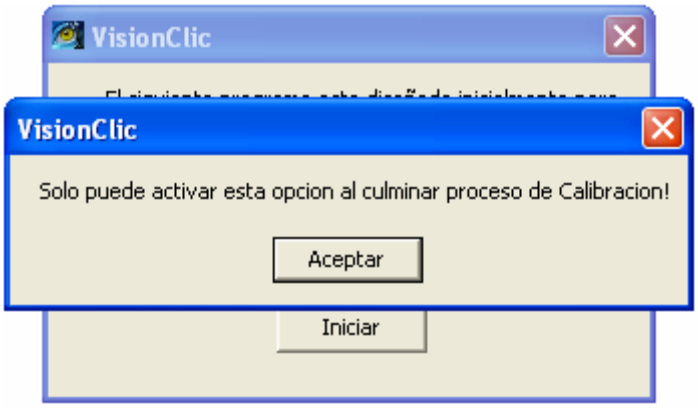

Figura 26: Advertencia generada al tratar de cambiar sensibilidad de parámetros sin haber culminado proceso de calibración