Universidad
Carlos III de Madrid

Departamento de Teoría de la Señal y Comunicaciones

PROYECTO FIN DE GRADO

Herramientas para un sistema de seguimiento de la mirada con visión estéreo en Matlab

Javier Alba de Viana-Cárdenas
Grado en Ingeniería de Sistemas Audiovisuales
(Plan 2011)

Tutor: Carmen Peláez Moreno

Leganés, septiembre de 2014

Título: Herramientas para un sistema de seguimiento de la mirada con visión estéreo en Matlab.
Autor: Javier Alba de Viana-Cárdenas
Tutor: Carmen Peláez Moreno

EL TRIBUNAL

Presidente: María Luz Durbán Reguera

Vocal: Vanessa Gómez Verdejo

Secretario: Enrique San Millán Heredia

Realizado el acto de defensa y lectura del Proyecto Fin de Grado el día 7 de Octubre de 2014 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a mi tutora la dedicación y la paciencia que ha tenido a la hora de implicarse en este proyecto. No ha sido posible hacerlo con la perfección que hubiéramos querido por diversos motivos personales y Carmen siempre lo ha entendido y me ha apoyado.

A lo largo de la carrera he tenido a mis compañeros de clase acompañándome en todos los momentos, por eso quiero darle las gracias a Aroa Bellés, Marta Chacón y Diego Aramendi por toda la ayuda prestada, haciendo especial mención a Alejandro Arjonilla, con el que tantas horas he pasado encerrado entre las paredes de esta universidad.

No me puedo olvidar de los amigos que llevan toda mi etapa universitaria dándome su apoyo y haciéndome más llevadera la carrera, muchas gracias por todo a Ricardo y a Asun, y en la recta final de mi grado, cuando más ayuda he necesitado, a Patri.

Me quedo sin palabras para expresar la gratitud hacia mi novia Alicia, que siempre ha estado a mi lado en todos los buenos momentos, pero sobre todo en los malos. Gracias por lo que has tenido que aguantar en mis épocas de exámenes y sobre todo en estos últimos meses de proyecto. Gracias por todo. Y por supuesto, gracias a mi suegra por confiar en mí en todo momento.

Pero el mayor agradecimiento de todos es para mi familia, que siempre ha estado a mi lado y me ha aconsejado a la perfección. Gracias a mis hermanos Juan y Víctor, y en especial a mis padres, Víctor y Esperanza, cuya educación, interés, ilusión, cariño y generosidad son los que han hecho posible que yo esté en esta fase de mi vida.

Gracias.

Resumen

A través de los ojos se pueden detectar enfermedades y patrones de comportamiento, así como también es posible hacer estudios e identificar dónde mira un individuo en una situación concreta. A medida que la sociedad evoluciona, es más necesario entender el comportamiento del ser humano para conseguir mejorar la vida de las personas, ya sea en el ámbito de la investigación para detectar enfermedades o en el ámbito de mejorar la facilidad de uso y accesibilidad de los aparatos y entornos que rodean al ser humano.

El seguimiento de la mirada es una técnica informatizada que sirve para ubicar los ojos de una persona y seguir su movimiento, pudiendo incluso detectar el punto exacto al que está mirando el usuario.

En este proyecto se estudian las principales técnicas y las aplicaciones más importantes del seguimiento de la mirada, más conocido por su anglicismo *eye-tracking*.

Partiendo de una serie de limitaciones de hardware y de software, en esta memoria se exponen las herramientas necesarias para hacer un sistema de seguimiento ocular de bajo coste con visión estéreo. Después de enumerar las herramientas necesarias, se han utilizado las más útiles para implementar en Matlab un ejemplo sencillo de seguimiento de los ojos. Posteriormente se extrae el punto al que el usuario está mirando mediante un algoritmo que se debe mejorar en líneas futuras. (Esta técnica en inglés se llama *eye-gaze tracking*).

Al final de este trabajo fin de grado se exponen alternativas y posibles vías por las cuales se puede continuar este proyecto.

Palabras clave: Matlab, Seguimiento de la mirada, visión estéreo, seguimiento ocular, punto de observación, binocular, calibración, estimación de la mirada, 3D, Viola & Jones.

Extended abstract

Eye tracking is a computerized technique to locate a person's eyes fixations and follow its motion. With an eye tracker, it is possible to detect how long a user is looking at something in particular and the path her eyes are following.

In this project the main techniques and the most important applications of eye tracking are explained. Eye tracking has been applied to numerous fields including human factors, cognitive psychology, marketing, and the broad field of human–computer interaction. [2]

As society evolves, it becomes more necessary to understand human behavior in order to improve the life of people. In particular, the interaction with computers can be improved if they are endowed with capabilities for detecting and predicting where users are looking at. Besides, from the point of view of research on diseases the automatic characterization of gaze can be useful for detecting behavioral patterns. Usability studies is one important use of eye tracking, their objective being to determine how user explores an interface with her gaze while interacts with the system.

Eye-tracking and eye-gaze techniques have received a lot of attention by professionals and business of the user experience sector. This is motivated by the proliferation of commercial solutions with enhanced precision and decreasing prices.

In the context of human-computer interaction, it is possible to distinguish two broad classes of eye tracking applications:

- As input device: although the precision of eye tracking used as input device is far from others like mouse or keyboard, it may have a variety of practical applications, such as its use in virtual reality or for motion-impaired. Besides, for particular operations like selection of objects from a graphical interface, gaze can be faster than mouse action.

- As a tool for objective evaluation of interfaces, where the information extracted is not required for analysis in real time, but is registered in log files for its posterior interpretation and analysis.

Two main groups of eye-trackers can be distinguished: those placed in the user’s head and those registering ocular motion from the distance, normally located in the user’s

monitor. The first ones are adequate for activities in which the participant must have total freedom to move but the second one are less intrusive.

From the point of view of technology, there exists a big variety, each of which has its own advantages and drawbacks. One of the more precise techniques implies the physical contact with the eye through a mechanism based on contact lens. Inevitably, these systems are uncomfortable for users. Most recent systems avoid this annoying characteristic based on the use of cameras that project infrared rays to the person's eyes. Another alternative is the use of simple high definition cameras, or web cams with a good illumination.

Eye-trackers require a previous calibration process, although this should not be regarded as a problem because it is usually a simple and fast process. However, there are some problems with certain amount of participants whose eyes can't be calibrated and this implies that this person cannot participate in the study or application. Wearing glasses or contact lenses are examples of such problematic subjects.

Besides, for some applications, displaying the results is considered a problem, for example to assess the times that a particular user is looking at a point on a web page. One solution employed is based on heatmaps. In these representations the hot zones or zones with high color intensity represent the focus of the user's attention for longer periods of time.

In this cases, understanding the amount of information obtained in a test isn't a trivial task. This requires a big knowledge of cognitive theory and metrics employed by the tester. Some metrics and their meanings are:

- Total number of fixations: a high number of fixations means less search efficiency and it can be indicate a problem in the interface layout.

- Number of fixations on an area of interest: a high number of fixations means high importance for user.

- Duration of gaze on an area of interest: a longer duration may indicate a higher difficulty to interpret the area content, for example.

- Spatial density of fixations: when a fixation is focused on a smaller zone it means that we have a better efficiency in the visual search, while if the fixations are more dispersed, the searching is being less efficiently. [17]

In this project a set of computational tools for eye-tracking have been developed for computer interaction. The chosen method is the least intrusive eye tracker, i.e., with two web cams from the distance. Along this report, these tools for eye-tracking with stereo vision will be presented. A simple eye-tracker has been implemented to demonstrate it.

Matlab has been chosen as the implementation language and the hardware used for recording users' gaze is composed of two web cameras of low definition. This poses many challenges since techniques employing high definition cameras under the Matlab environment do not allow simultaneously recordings from two cameras and a third-party software has been employed for this task.

To make this project feasible several restrictions have been posed on the application: face motion has to be limited and the illumination needs to be constant and sufficient.

These are the steps followed for the construction and programming of the application:

- First, the recording from two web cams simultaneously has to be enabled. After the consideration of several alternatives the software ISpy is selected. Besides its graphical interface it also has a command interface that facilitates its integration with the rest of Matlab code.

- Calibration is always required, in this case, the two web cameras are mounted in a fixed setting to avoid displacements. A calibration video has been developed.

- A graphical user interface is created in Matlab to ease the management of the program.

- The eye detection and location is an important phase. We have used Matlab's vision toolbox to detect the face constituents by using the well-known Viola&Jones algorithm. It would possible detect the right eye and the left eye separately, but the algorithm is more reliable if it detect both eyes. After this, is necessary to perform the segmentation of each eye to detect the iris.

- Iris detection is the most difficult part due to the low definition cameras. Several alternatives have been tried out as the Hough circular transform and other methods measuring amounts of white pixels, but we have finally implemented Daugman's integro-differential operator.

- Iris tracking is the next step. Matlab's vision toolbox includes a point tracker that has been employed to track the irises for which the center points in the irises have been selected as tracking target. These points are saved for each frame of the video.

- Finally, the extraction of eye-gaze is the last step. 3D techniques able of taking advantage of two web cams (stereo vision) are necessary. Both eyes are separated and treated individually, extracting the coordinates of the point of the screen the user is looking at by implementing a mathematical algorithm that makes use of the difference of the distances between both. The algorithm has been invented by student. Due to the limited resolution of the cameras the screen has to be divided into five positions being the precision limited to these five options.

Conclusions of this project are explained in chapter 6 (Conclusion and further work). [3]

Keywords: Matlab, Eye-Tracking, Eye-Gaze, stereo vision, calibration, gaze estimation, 3D, Viola & Jones.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

El ser humano tiene comportamientos asociados a los movimientos de los ojos y a la dirección de la mirada. El seguimiento de la mirada, en inglés '*eye-tracking'*, es una técnica que permite conocer la posición y el movimiento de los ojos de un individuo. Esta metodología permite estimar a que punto de una pantalla o proyector se está mirando, es decir, qué punto se está contemplando, en inglés '*eye-gaze tracking'*. Es posible saber cuánto tiempo se está observando un punto e incluso detectar el camino que siguen los ojos del usuario. Aunque la tecnología del eye-tracking pueda parecer reciente, el estudio del movimiento ocular tiene más de un siglo de historia y su primera aplicación data de los años 50 a manos de Fitts, Jones y Milton. Su temática son estudios sobre ergonomía en la aeronáutica. [16]

El motivo por el que el alumno ha realizado este proyecto ha sido por la cantidad de utilidades que presenta el eye-tracking. También ha influido en la decisión las preferencias de la tutora en un tema que actualmente está en auge.

El proyecto trata de estudiar las técnicas de eye-tracking actuales y de buscar las utilidades para el ser humano. Después de revisar las técnicas y los principales usos, se llega a la conclusión de que hay algoritmos y equipos muy sofisticados que realizan un buen seguimiento de la mirada, pero el factor común a todos ellos es su elevado coste. Este es el motivo por el cual se elige hardware y software de bajo coste y se exponen herramientas de procesado que pueden permitir realizar un seguimiento de la mirada

preciso sin demasiado coste computacional. Se decide concentrar el trabajo en una técnica específica de eye-tracking: seguimiento de la mirada de ambos ojos con dos webcams para formar visión estéreo tratando de utilizar técnicas de 3 dimensiones en Matlab.

Sería muy útil e importante obtener un programa que con un coste bajo sirviera para estimar el punto exacto al que está mirando un usuario. Ya que cualquier persona en su vivienda podría comprarse dos webcams e instalar el programa y obtener facilidades a la hora de usar su computadora o directamente poder usarla si se tiene alguna discapacidad física en las extremidades superiores. En este proyecto se proponen herramientas y una solución muy sencilla de eye-tracking con la finalidad de que en posteriores trabajos fin de grado se mejore la robustez y la precisión [1][2].

1.2 Introduction

Certain human behaviors can be associated with eye motion and gaze direction. Eye tracking is a technique that allows the detection and tracking of the position and motion of a person's eyes. Based on this methodology a posterior estimation of the point being observed on a screen or projector is known as eye-gaze tracking. It is possible to compute the amount of time a user is looking at a particular point and even detect the trajectory of the user's gaze.

The motivation for this project is the fact that eye-tracking presents a large number of applications together with the fact that it is a hot research topic.

The project aims at studying the current eye-tracking techniques and possible applications. After a review of the available techniques and its main uses, the conclusion is that in spite of the existence of many algorithms and hardware to perform reliable eye tracking, but the common denominator of most of them is their high cost. This is why low-cost hardware and software has been chosen for this project. It was decided to concentrate on a specific eye-tracking technique: tracking the gaze of both eyes with two web cams to compute stereo vision with the goal of using 3D techniques in Matlab.

Our main long-term purpose is to provide a low-cost program able of estimating the exact point at which a user is looking. In this way, anyone in their home could buy two web cams and install the program enabling an ocular interface for motion-impaired persons, for example. In this project, the necessary tools are reviewed and proposed and a simple eye-tracking solution with a low accuracy is presented, leaving improvements for further work [1][2].

1.3 Objetivos

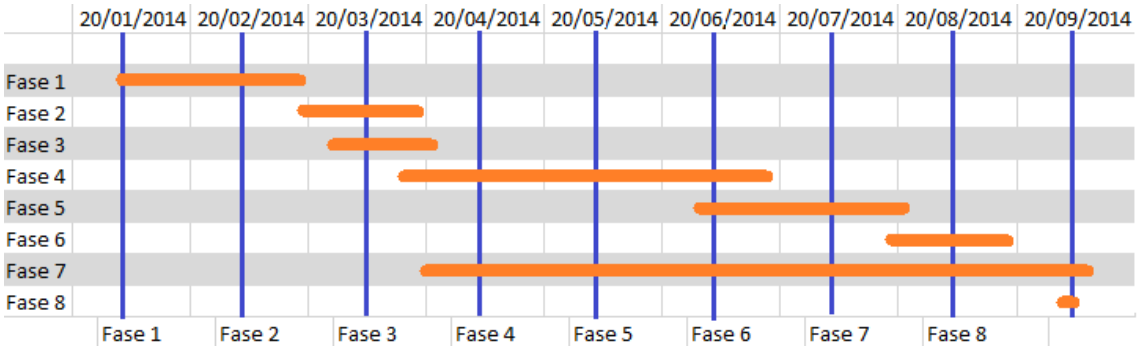
El objetivo fundamental del trabajo fin de grado es el de proponer una serie de herramientas que sirvan de base para la implementación futura de un algoritmo robusto que haga un buen eye-tracking y que detecte el punto al que está mirando el usuario de manera precisa. En base a ese objetivo principal, se proponen los siguientes objetivos parciales:

- Obtener las herramientas suficientes para hacer una implementación completa de un seguidor de la mirada, en inglés *'eye-tracker'*.
- Conseguir implementar un eye-tracker básico que partiendo de unas limitaciones, haga un seguimiento de los ojos aceptable.
- Plantear una solución para estimar el punto al que el usuario está mirando en la pantalla mediante técnicas estéreo. Método *'eye-gaze'*.

1.4 Fases del desarrollo

Las fases en las que se ha realizado el proyecto han sido las siguientes:

- Fase 1: decisión del enfoque general del proyecto.
- Fase 2: búsqueda de información sobre eye tracking.
- Fase 3: selección y organización de las herramientas necesarias para un eye-tracker.
- Fase 4: búsqueda y realización de pruebas de programas que cumplen los requisitos de las herramientas buscadas.
- Fase 5: diseño del programa en Matlab.
- Fase 6: pruebas y mejoras del programa.
- Fase 7: redacción de la memoria.
- Fase 8: grabación de videos de demostración.



	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5	Fase 6	Fase 7	Fase 8
Inicio	20/01/2014	03/03/2014	10/03/2014	07/04/2014	16/06/2014	04/08/2014	07/04/2014	18/09/2014
Duración	42	30	27	90	49	30	165	2

Tabla 1: Diagrama de Gantt de las fases del proyecto.

1.5 Medios empleados

Para poder realizar este proyecto se ha necesitado disponer de diferentes medios.

1.5.1 Red privada virtual

Para acceder a recursos comprados por la Universidad Carlos III de Madrid, por ejemplo, artículos alojados en páginas como ScienceDirect o IEEEXplore, se necesita conectarte a internet a través de una dirección ip de la universidad. En el caso de no estar físicamente allí se puede crear una red privada virtual (*'VPN' Virtual Private Network*), la cual permite simular que se está conectado a internet desde una ip de la universidad sin necesidad de encontrarse en ella. El alumno ha seguido el siguiente tutorial proporcionado por la Uc3m: https://asyc.uc3m.es/index.php?Id=168 .

1.5.2 Dispositivos de grabación

La universidad ha provisto al alumno de dos webcams Logitech v-u0011 cuya resolución óptica es de 640x480 VGA. [8]

La baja resolución de las webcams ha sido un papel clave en el diseño del programa.

Para evitar el calibrado cada vez que se hiciera una prueba diferente, el alumno modificó las webcams y fabricó una solución casera para que las cámaras siempre se encuentren en la misma posición. También es necesario una iluminación uniforme que no produzca sombras en la cara del usuario, ya que la aplicación tiene limitaciones debidas a la iluminación. La figura 1 muestra el montaje realizado:

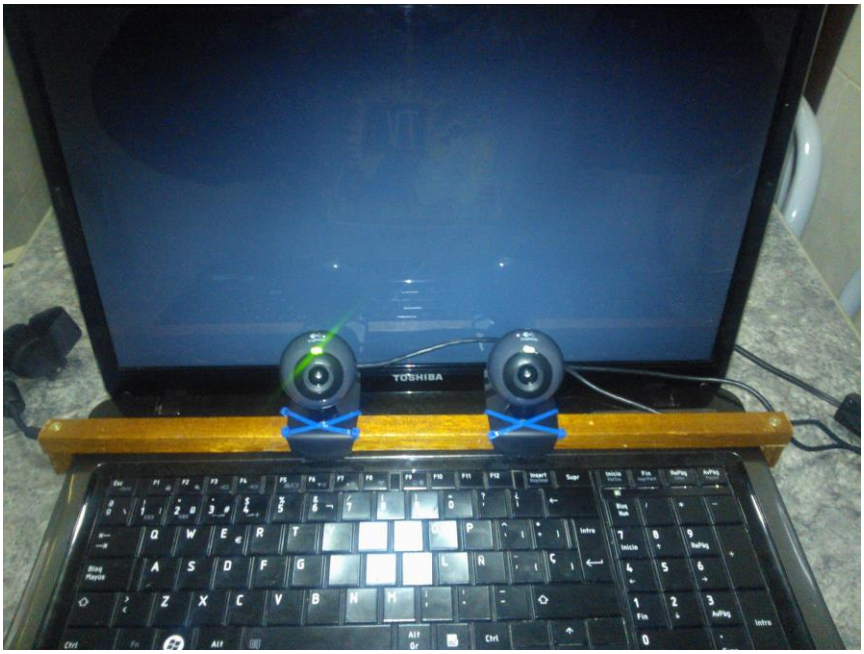


Figura 1: fotografía del montaje.

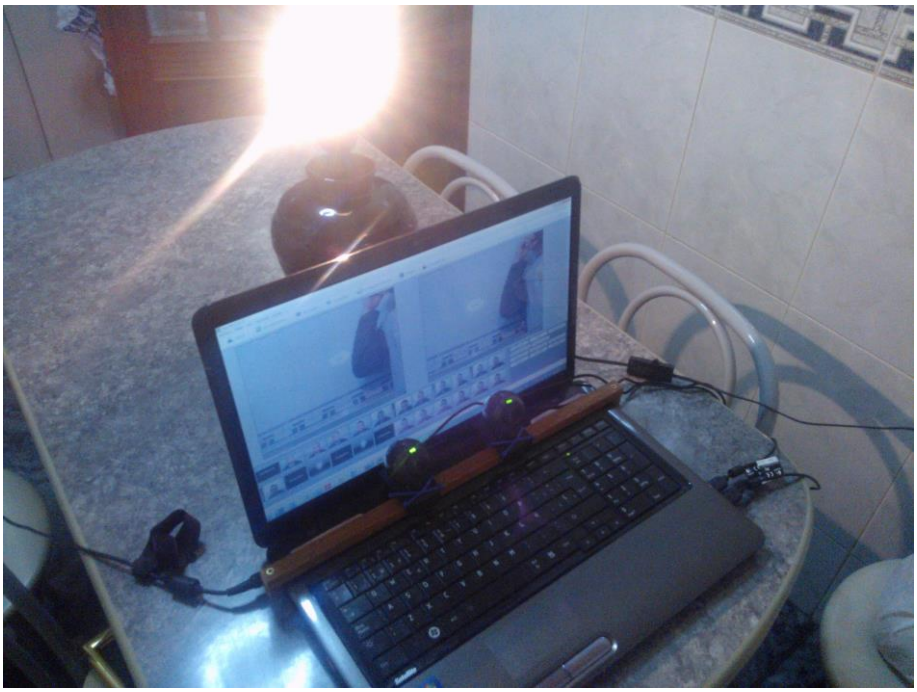


Figura 2: fotografía del montaje con iluminación controlada.

1.5.3 Matlab

MathWorks es el líder en desarrollo de software de cálculo matemático para ingenieros y científicos. MathWorks, fundada en 1984, cuenta con más de 3.000 trabajadores en 15 países y su sede central está situada en Natick, Massachusetts, Estados Unidos.

Matlab es un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, la visualización y la programación. Mediante MATLAB, es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. El lenguaje, las herramientas y las funciones matemáticas incorporadas permiten explorar diversos enfoques y llegar a una solución antes que con hojas de cálculo o lenguajes de programación tradicionales, como pueden ser C/C++ o Java.

Matlab se puede utilizar en una gran variedad de aplicaciones, tales como procesamiento de señales y comunicaciones, procesamiento de imagen y vídeo, sistemas de control, pruebas y medidas, finanzas computacionales y biología computacional. Más de un millón de ingenieros y científicos de la industria y la educación utilizan Matlab, el lenguaje del cálculo técnico. [9]

Para este proyecto se ha instalado Matlab R2012b en un sistema operativo Windows 7 de 64 bits. El PC en el que corre el programa es un portátil Toshiba Satellite con procesador Intel Core i5 a 2,53 GHz, y 4 Gb de memoria RAM.

Matlab pese a todos los beneficios de los que dispone, tiene algunas limitaciones. En el caso actual, la caja de herramientas de *'videoinput'* de Matlab, de ahora en adelante '*toolbox*', permite la grabación de dos webcam de manera casi simultánea, pero el

instante de tiempo en el que comienzan a grabar cada cámara no es exactamente el mismo. Podría servir perfectamente para el cometido del proyecto, pero se intentó buscar un programa que permitiera comenzar la grabación en el mismo instante.

1.5.3 iSpy

ISpy es un software de cámaras de seguridad de código abierto. Tiene la funcionalidad de grabar simultáneamente dos webcams. Además es posible manejar ciertos aspectos del programa desde línea de comandos de Windows, y por lo tanto también desde Matlab. Por ejemplo se puede empezar a grabar y parar la grabación directamente desde Matlab.

La versión instalada es iSpy de 64 bits versión 6.2.4.0. [10].

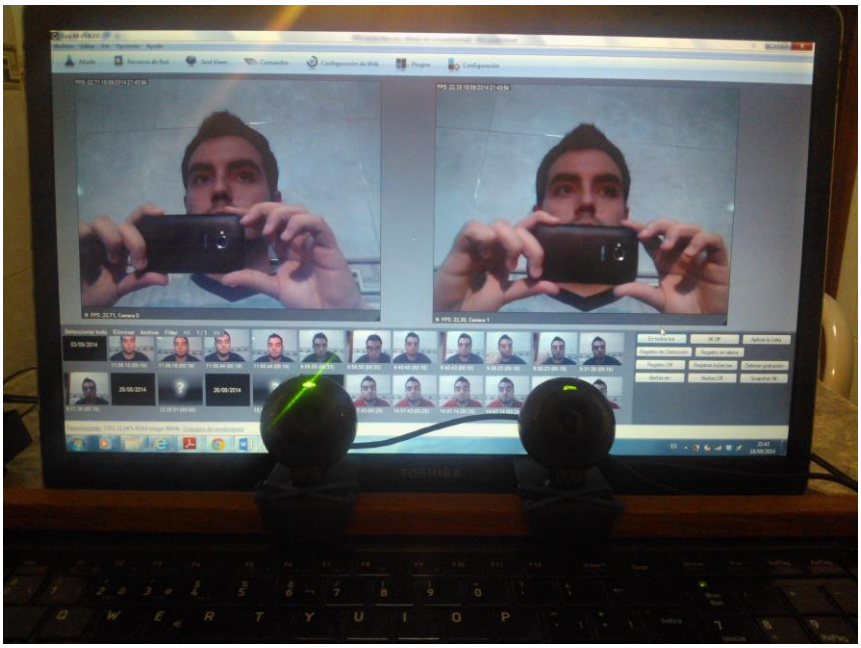


Figura 3: Programa iSpy con el montaje antes de grabar.

1.6 Estructura de la memoria

La memoria incluye los siguientes capítulos:

- **Capítulo 2: Planteamiento del problema, factores a tener en cuenta.**
  Se estudia el estado actual de las tecnologías sobre detección de la mirada y teniendo en cuenta el marco regulador técnico y legal y el entorno socio-económico, se plantea el problema a resolver.

- **Capítulo 3: Búsqueda de herramientas.**
  Se clasifican todas las herramientas necesarias y se comparan diferentes alternativas para un diseño final.
- **Capítulo 4: Diseño técnico de la posible solución.**
  Se plantea una posible solución diseñada por el alumno, exponiendo detalladamente el procedimiento seguido y las limitaciones que tiene.
- **Capítulo 5: Conclusiones y línea futura.**
  Se comprueban los resultados y se estudia el presupuesto. Se presentan soluciones alternativas y líneas de acción futura.
- **Capítulo 6: Conclusion and future works.**
  En inglés las conclusiones y las líneas de acción futura.

Capítulo 2

Planteamiento del problema, factores a tener en cuenta

2.1 Estado del arte

Los mecanismos de seguimiento de la mirada abren multitud de posibilidades y cada vez se utilizan técnicas graficas más avanzadas. Las aplicaciones del eye-tracking se pueden separar en dos grandes categorías: interacción y diagnóstico.

En la función de diagnóstico, el seguimiento de los ojos ofrece pruebas objetivas y cuantitativas de los procesos de atención visual de un usuario. En general, lo único que interesa en este ámbito son los movimientos oculares que se registran para determinar patrones de atención del usuario sobre un determinado estímulo, es decir, no interesa tanto el eye-gaze. Suele ser deseable que el rastreador ocular sea discreto e incluso disfrazarlo para que los usuarios no sean conscientes de su presencia. Por otra parte, para la modalidad de diagnóstico, lo normal es que no sea necesario evaluar los movimientos del ojo en tiempo real, sino que se puede hacer una grabación y después de la fase de captación, obtener los resultados del experimento.

En la modalidad de interacción, el mecanismo más utilizado es la detección de la mirada (eye-gaze), y es un poderoso dispositivo de entrada que puede ser utilizado por muchas aplicaciones con interfaz gráfica. Lo esperable es que el sistema detecte la mirada del usuario y que reaccione interactuando con él. La aplicación más conocida es la de usar únicamente con la mirada un teclado proyectado en una pantalla. Se hace un

seguimiento de la mirada del usuario y mediante calibraciones y diferentes cálculos se consigue estimar el punto del proyector al que está mirando el individuo.

2.1.1 Tipos de tecnologías de seguimiento de la mirada

La primera clasificación que se puede hacer es la de técnicas de eye tracking de un solo ojo o de ambos ojos. Dependiendo de la aplicación y de los equipos disponibles se puede hacer eye tracking monocular o binocular. Para estudios psicológicos y fisiológicos de posibles enfermedades es obligatorio que el seguimiento sea de ambos ojos, pero para hacer un seguimiento del punto de observación de un usuario sano, sería suficiente hacer seguimiento de la mirada de un ojo. Si se siguen ambos ojos siempre será más fiable y preciso que seguir sólo un ojo.

Se usan tecnologías muy avanzadas para detectar el movimiento del ojo que requieren tener contacto directo con el usuario, por ejemplo mediante lentillas especiales, sensores de campo magnético o electrodos que permiten medir el potencial eléctrico para detectar el movimiento. No se explicará más en detalle ya que este proyecto se centra en metodologías no invasivas.

La tecnología más utilizada para el seguimiento de la mirada es a través de una o varias cámaras de rayos infrarrojos, que permiten detectar las pupilas con facilidad. Esta técnica ha sido muy utilizada en investigación sobre el sistema visual, psicología cognitiva y en el diseño de publicidad para productos. Se implementan también en gafas y cascos especiales.

Otro tipo de eye-tracking es la llamada reflexión en la córnea. Consiste en utilizar una luz para iluminar los ojos, la cual causa una reflexión en la córnea que es detectada por una cámara de alta resolución. La imagen capturada se utiliza para identificar la reflexión de la luz en la córnea y la pupila y con algoritmos avanzados de procesamiento de imagen se consigue estimar el punto que se está contemplando.

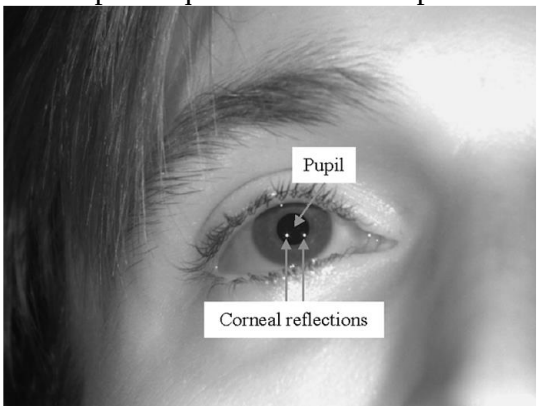


Figura 4: **Ejemplo de un ojo con dos reflejos en la córnea. [7]**

Por último, existen seguidores de la mirada (eye-trackers) que utilizan videocámaras o webcams normales con un potente software que permite hacer una estimación del movimiento del ojo. Hay algoritmos que permiten hacer una estimación bastante

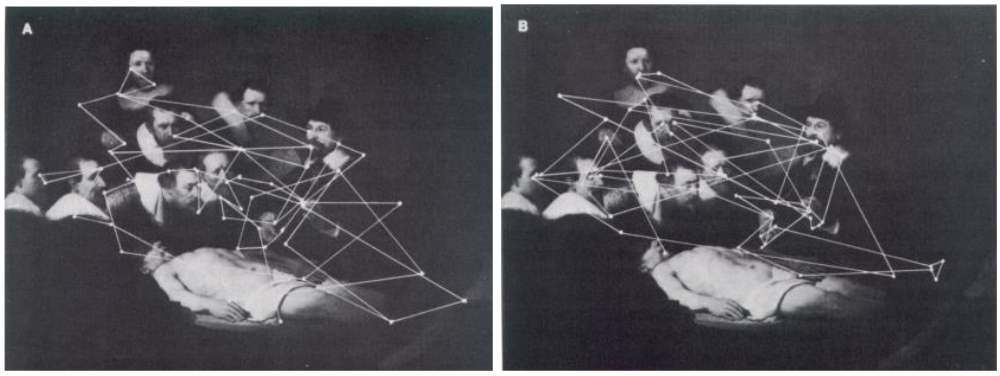
aceptable con una única cámara, pero lo normal es que se utilicen múltiples cámaras de alta definición para hacer un seguimiento de la mirada fiable.

2.1.2 Usos del seguimiento de la mirada

En la actualidad el seguimiento de la mirada es muy utilizado en diversos campos:

- **Investigación científica.** Universidades europeas y norteamericanas utilizan el seguimiento de los movimientos oculares en sus estudios básicos sobre percepción, atención y búsqueda visual.

- **Percepción del arte.** Hay estudios que mediante el seguimiento de los ojos permiten saber el tipo de interés de un individuo que este observando una obra de arte, pudiendo ser interés semántico o interés en lo artístico. En la figura 5 se observan pequeñas diferencias en los patrones de visión entre dos sujetos observando un cuadro.



(A) Grupo artístico. (B) Grupo semántico.

Figura 5: Gráficos de 60 puntos de visión de dos individuos observando "Rembrandt's Anatomy Lesson" [6].

- **Sistemas de seguridad en automóviles.** En la actualidad hay fabricantes de automóviles que implementan en sus vehículos sistemas de detección de los ojos del conductor. Este sistema permite detectar la fatiga y avisa al conductor de cuando debería parar a hacer un descanso.

- **Marketing y publicidad.** Se observa la mirada de un usuario cuando accede a una página web y con la técnica del eye-gaze se analiza en qué áreas centra su atención y con qué frecuencia. Se suele representar gráficamente con mapas de calor, colores cálidos en las regiones en las que más se ha fijado la mirada, y colores fríos en las que menos. [http://www.kuaest.com/2011/09/eye-tracking.html]. Con esta información es posible saber si la publicidad en una página web está bien situada para captar la atención del usuario o si es necesario reubicar los segmentos publicitarios. Un hallazgo importante gracias al eye-tracking en el ámbito de las páginas web y las interfaces de búsqueda ha sido el llamado "triángulo de oro" descubierto en 2005, que es aquella zona que capta más miradas de la mayoría de los usuarios, que está situada en la esquina superior izquierda de un listado de resultados. Este

descubrimiento ha permitido a los buscadores elegir la posición de los principales resultados y la mejor ubicación para publicidad y para las páginas web que pagan por aparecer en la primera página.

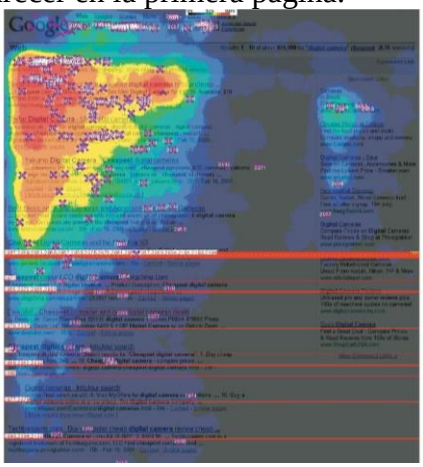


Figura 6: **Mapa de calor que representa el triángulo de oro de google [11].**

Antes de lanzar un anuncio se hacen estudios previos de mercado en los que se analizan qué partes de la publicidad llaman más la atención de los consumidores. Gracias a estas investigaciones los publicistas pueden profundizar en los procesos implicados en la comunicación con sus clientes para crear formas más efectivas de promoción.
Existen equipos y empresas que se dedican a mejorar la colocación de productos en las estanterías de los supermercados. Esto es posible gracias al eye tracking.

Figura 7: Estudios sobre el consumo en un mercado mediante las gafas de seguimiento de la mirada de la empresa Tobii [4].

- **Pedagogía.** Cuando se lee un texto en un lenguaje distinto al que se está acostumbrado, un sistema de eye tracking detecta dificultades en la lectura, por ejemplo patrones repetitivos o fijaciones duraderas. Mediante este método se pueden identificar casos de discapacidad cognitiva. Además ampliando el campo del eye tracking a la detección de variaciones de la pupila, se abre la posibilidad de identificar la carga del trabajo mental de un individuo.

- **Ergonomía.** Conocidas empresas de aeronáutica y fabricantes de automóviles utilizan la tecnología del eye gaze en sus estudios de ergonomía, con objeto de verificar la mejor disposición de los paneles de mandos, especialmente en situaciones de emergencia.

- **Estudio y mejora del rendimiento deportivo.** En baloncesto, por ejemplo, se ha utilizado el seguimiento de la mirada para entrenar lanzamientos a canasta, ayudando al jugador a que aprenda a fijarse en determinadas zonas del tablero para mejorar la técnica de tiro. También se analiza el movimiento ocular para calcular tiempos de reacción, por ejemplo para estudiar los reflejos de atletas profesionales.

- **Salud.** Se realizan diagnósticos clínicos y corrección de defectos gracias al análisis del movimiento del ojo. Actualmente se aplica con éxito en operaciones con láser de corrección de problemas como la miopía, hipermetropía o astigmatismo.

- **Accesorios avanzados para videojuegos.** Existen cascos con sistemas de seguimiento de la mirada que permiten interaccionar con un videojuego como herramienta añadida a los mandos tradicionales.

- **Interacción con dispositivos móviles.** Se ha programado software que permite a los terminales móviles que disponen de cámara frontal hacer un seguimiento de la mirada básico. Por ejemplo en los dispositivos Android, hay aplicaciones que permiten al usuario bajar la barra de desplazamiento de un documento o una página web únicamente bajando la mirada.

- **Interacción con ordenadores.** Es posible separar esta sección en dos partes:
    - Interacción humano-ordenador para discapacitados físicos. Actualmente es una de las aplicaciones más significativas. Hay determinadas discapacidades que impiden a una persona utilizar las manos, como por ejemplo la esclerosis múltiple. Para permitir que una persona con discapacidad física en las extremidades superiores pueda manejar un ordenador existen softwares y hardwares que permiten esta interacción gracias al eye-tracking y eye-gaze.

Figura 8: Equipo de seguimiento de la mirada para discapacitados de la empresa *'Allied vision technologies' [5]*.

- Interacción humano-ordenador para mejorar la facilidad de uso. Hay estudios que han demostrado que en tareas sencillas se puede ahorrar hasta un 60% de tiempo en seleccionar un objeto con la técnica del eye gaze en comparación con el ratón, pero aún se puede mejorar mucho en este campo. También está presente el problema de que con los ojos no se puede hacer clic como con un ratón. Hay implementaciones en las que se puede hacer clic pestañeando un determinado número de veces o dejando la visión fija en el punto que queremos seleccionar durante un tiempo específico [6].

Para muchas de estas aplicaciones se necesita poder medir de alguna manera que significa cada fijación. En la literatura científica hay una extensa revisión de las posibles métricas, describiendo como se miden y su significado, a continuación se exponen algunos ejemplos:

- Número total de fijaciones: Un mayor número de fijaciones indica una menor eficiencia de la búsqueda, lo que puede indicar un problema en el layout (esquema organizativo) de la interfaz.

- Número de fijaciones sobre un área de interés: Mayor número de fijaciones indica mayor importancia para el usuario.

- Duración de la mirada fija sobre un área de interés: Mayor duración indica mayor dificultad para interpretar el contenido del área.

- Densidad espacial de las fijaciones: Cuando las fijaciones se concentran en una zona más pequeña indica mayor eficiencia en la búsqueda visual,

mientras que si son más dispersas sugieren que la búsqueda está resultando menos eficiente.

- Tiempo transcurrido hasta la primera fijación: Cuanto menos tiempo transcurra hasta que el usuario se fije por primera vez en un área de interés, mayor será la capacidad de las propiedades gráficas del área para atraer la atención visual [17].

2.1.3 Seguidores de la mirada con grabación estéreo

Se ha comentado en esta memoria que hay diferentes tecnologías para realizar el eye-tracking. Para estimar el punto de observación, eye-gaze, es mucho más fiable un sistema con más de una cámara que en mono visión, para así poder formar una imagen en tres dimensiones y de esta forma calcular distancias de manera exacta y extraer el punto al que se está mirando.

En proyectos de diferentes universidades se han hecho experimentos y proyectos sobre eye-tracking con varias cámaras de alta definición en soportes especializados o con cámaras de visión nocturna, lo que implica un coste más alto que el previsto en este proyecto.

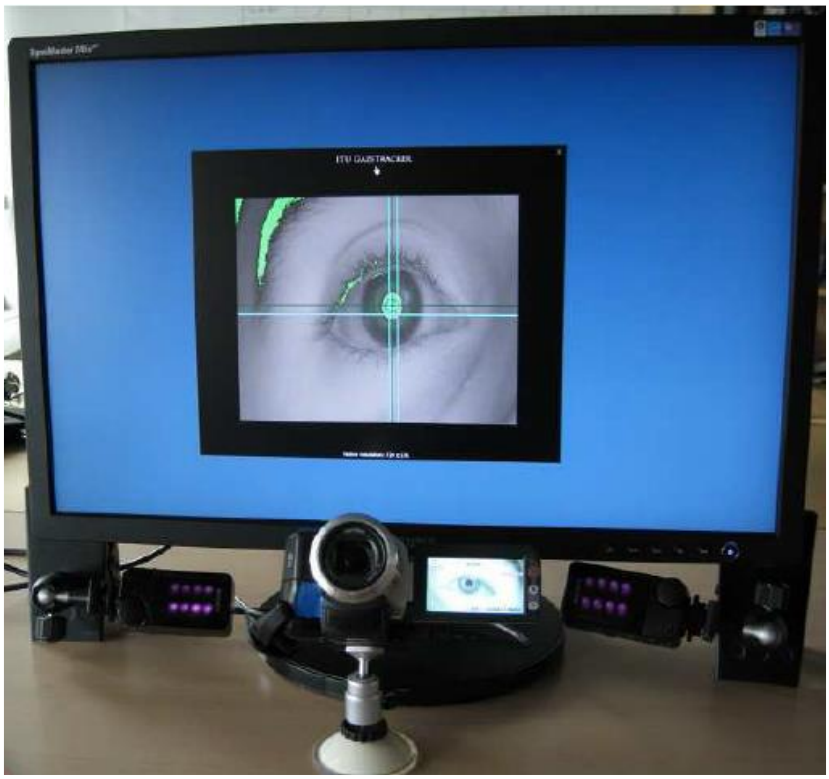


Figura 9: **Eye tracker con una videocámara de visión nocturna y dos fuentes de luz infrarroja [12].**

Muchas empresas se dedican a fabricar dispositivos para hacer seguimiento de la mirada con componentes estéreo. La intención de estas empresas es vender el producto a otras compañías para que hagan sus propios estudios de publicidad o marketing, o a universidades y hospitales con fines de investigación.

Las empresas más conocidas en la fabricación de eye-trackers son:

- Tobii Technology, dedicada a fabricación de eye trackers de todo tipo, cuyo producto más conocido son las "Tobii Glasses", un seguidor de la mirada instalado en unas gafas. Muy utilizadas para el estudio de marketing en comercios y productos. El mayor inconveniente es el precio, ya que las gafas más básicas rondan los 14.000 dólares.

Figura 10: 'Tobii Glasses 2' y 'Tobii X2-60 Eye Tracker' [4].

- MyGaze, empresa que ha lanzado un dispositivo llamado 'MyGaze Eye Tracker', de alta tecnología que permite una interacción robusta y fácil con un PC mediante el movimiento de los ojos. Dicho dispositivo cuesta 499 Euros.

Figura 11: Dispositivo MyGaze Eye Tracking en las posibles ubicaciones en un monitor [13].

Como se ha visto, no se encuentran eye-trackers estéreo de bajo coste.

2.2 Entorno socio-económico

Como hemos visto en el estado del arte, el eye-tracking y el eye-gaze tienen multitud de aplicaciones para la sociedad. Actualmente vivimos en un mundo en constante desarrollo, y la investigación del comportamiento humano está en auge.

El eye-tracking puede suponer descubrir enfermedades neuronales y psicológicas con mucha más rapidez que en la actualidad, también es muy importante para estudios sobre pedagogía y ergonomía. También permite dar la posibilidad de interactuar con un ordenador a discapacitados físicos, pero el aspecto más importante para la sociedad es la posibilidad de utilizar el eye tracking para solucionar problemas de salud.

Respecto a la economía, el eye-tracking se puede convertir en un factor clave para el marketing y la publicidad en el futuro. Actualmente ya es importante y es muy utilizado

para saber cómo ubicar publicidad tanto en webs como en productos, y para colocar de mejor manera los artículos en los supermercados. Las empresas que vayan a lanzar un nuevo producto estarán muy interesadas en comprar eye-trackers que les permitan estudiar de qué manera atraerán más la atención de los compradores.

2.3 Marco regulador

En la actualidad no existe ningún marco regulador técnico. Aunque la investigación de esta tecnología data de hace más de 20 años, es en la década actual cuando se están dando grandes pasos en este ámbito.

Ya es posible identificar hacia donde está mirando un usuario a una distancia prudente sin que el usuario sea consciente de ello, por lo tanto el primer paso a seguir sería hacer una regulación para la protección de los individuos frente a estos sistemas.

En cuanto al marco legal, para la mayor parte de los eye-trackers que salen al mercado se necesitan hacer experimentos previos con humanos, y por lo tanto se deben cumplir unos principios éticos contemplados en la ley de protección de datos de carácter personal (LOPD).

2.4 Planteamiento del problema, requisitos y restricciones

Después de estudiar el estado de desarrollo y las aplicaciones de eye-tracking que existen, se observa que los eye trackers que trabajan en estéreo tienen un elevado coste, o son con técnicas de rayos infrarrojos. Por lo tanto el problema que se plantea es el de obtener las herramientas necesarias para programar un eye tracker que permita señalar aproximadamente el punto que se está mirando mediante técnicas estéreo. Se ha propuesto una solución simple a este problema, con una serie de requisitos y restricciones.

La primera restricción que se tiene en cuenta es la baja resolución de las webcams, debido a ello se descartan técnicas de eye tracking que requieran alta resolución, como por ejemplo la llamada reflexión en la córnea. También implica que el algoritmo que se desarrolle sea muy sensible a los cambios de iluminación, ya que las técnicas de detección de los ojos y de los iris en situaciones de baja resolución funcionarán mejor basándose en cambios de color, que en formas definidas.

El lenguaje y entorno de programación elegido para desarrollar la aplicación es Matlab. Mathworks dispone de una toolbox de visión con muchas herramientas útiles, una de ellas permite la grabación de una webcam desde el propio Matlab, o incluso de dos webcams diferentes de manera consecutiva. Sería suficiente para este proyecto, Pero se busca que la grabación de ambas webcams sea exactamente simultánea, lo que

conlleva que se busque otro programa alternativo para registrar el video de las webcams. El utilizar un programa externo es un inconveniente que tiene la implicación de que no se pueda hacer el seguimiento de los ojos en tiempo real, es decir, según se va grabando, sino que hay que esperar a que terminen las grabaciones para poder procesar el video. Del mismo modo, si se realizara la grabación desde Matlab, no se podría hacer el eye tracking en tiempo real debido a la carga computacional que supone la detección de los ojos y el tracking de los mismos.

Durante el proyecto se buscan herramientas para hacer el eye tracking y se proponen técnicas e ideas para dar más robustez al algoritmo, pero en cuanto a la aplicación de ejemplo creada por el alumno se establecen unas restricciones previas para centrarse en el objetivo principal del trabajo fin de grado. Estas restricciones prefijadas son:

- No se va a contrarrestar el movimiento de la cabeza, por lo que para hacer una buena estimación del movimiento de los ojos, no se puede mover la cabeza del sitio.

- Como se ha visto con anterioridad, se necesita una buena iluminación que no produzca sombras en la cara.

- No se controlarán los pestañeos, por lo que si se pestañea demasiado, el algoritmo se perderá con mucha frecuencia.

- La etapa de calibración será sencilla y necesaria cada vez que se instale el montaje en el ordenador. El algoritmo está diseñado para que funcione bien con el montaje propuesto, si se cambian las posiciones de las webcams, el programa no será tan preciso.

- No se va a evaluar el sistema para diferentes usuarios ya que se limitará a hacer pruebas con uno sólo.

Capítulo 3

Búsqueda de herramientas

3.1 Identificación de las herramientas necesarias

Una vez que se decide el hardware que se va a usar, en este caso dos webcams conectadas al portátil, se ha de encontrar una metodología para grabar video en las dos cámaras de manera simultánea.

Para que las técnicas de estimación del eye gaze resulten precisas es necesario introducir una fase de calibración del montaje.

Hacer el programa más sencillo de usar y más llamativo visualmente es bastante recomendable, para ello se puede diseñar una interfaz gráfica en Matlab.

Es necesario encontrar herramientas para hacer un eye tracking robusto. Un primer paso sería detectar la cara del usuario para que no afecte su movimiento al seguimiento de la mirada. A continuación se tienen que detectar los ojos del usuario y posteriormente los iris. Todo este procedimiento se ha de hacer por duplicado, para ambos videos procedentes de las webcams.

Una de las etapas más importantes es la de hacer el seguimiento de los iris y poder capturar los puntos donde se encuentran a cada instante.

Por último se requiere extraer el punto de observación, el eye gaze, a partir del seguimiento de la mirada realizado.

Para cada una de estas herramientas se han intentado encontrar programas completos que sirvan para los objetivos del alumno. En los próximos apartados se explican las herramientas encontradas para cada fin.

3.2 Grabación de video estéreo

Para seguir los ojos de un individuo en un video, es suficiente con la grabación de una sola cámara, pero el fin del proyecto es proponer herramientas para estimar el punto al que está mirando una persona y es más preciso hacer la captación con dos cámaras para proporcionar una visión estéreo. De esta forma se podrán utilizar técnicas 3D para estimar el eye-gaze. Por lo tanto se necesita un programa que permita grabar video procedente de dos fuentes diferentes, las dos webcams conectadas al PC. El requisito indispensable es que la grabación sea simultánea y sincronizada, ya que para extraer el punto exacto al que un usuario está mirando, hay que poder obtener el fotograma (de ahora en adelante *'frame'*) de los dos videos en el mismo instante de tiempo.

La primera solución que se buscó fue la más evidente, tratar de capturar el video desde Matlab. Se encuentra la toolbox *'videoinput'* de Matlab, que permite obtener una vista previa de una o varias webcams. Al programar un sencillo algoritmo para grabar video procedente de las dos webcams se detecta que los instantes de tiempo iniciales de las dos grabaciones no son los mismos. Hay 5 frames de media de diferencia entre una grabación y la otra. Controlar esta diferencia no es trivial, ya que los frames de diferencia entre un video y otro no son constantes porque dependen de la velocidad del procesador del ordenador en cada momento. También se ejecuta la detección de ojos en tiempo real, según se van captando los frames de video, pero como el algoritmo de detección requiere demasiada carga computacional, se pierden frames de la grabación y hace imposible la detección de los ojos en tiempo real, por este motivo se descarta realizar en esta aplicación la detección en tiempo real. A continuación se expone el código utilizado para la captación del video procedente de las dos webcams para posibles trabajos futuros:

```
clear all;
clc;

vid = videoinput('winvideo',1, 'YUY2_320x240');
vid2 = videoinput('winvideo',2, 'YUY2_320x240');
set(vid, 'FramesPerTrigger', Inf);
set(vid, 'ReturnedColorspace', 'rgb');
set(vid2, 'FramesPerTrigger', Inf);
set(vid2, 'ReturnedColorspace', 'rgb');

vid.FrameGrabInterval = 1; % Distancia entre frames
start(vid)
vid2.FrameGrabInterval = 1;
start(vid2)
aviObject = avifile('myVideo.avi');
aviObject2 = avifile('myVideo2.avi');% Creación de un nuevo fichero avi
for iFrame = 1:100                    % Capturamos 100 frames
```

```
  I=getsnapshot(vid);
  I2=getsnapshot(vid2);

  F = im2frame(I);          % Convertir una imagen en un frame de video
  aviObject = addframe(aviObject,F);  % Añadir el frame al fichero avi
  F2 = im2frame(I2);
  aviObject2 = addframe(aviObject2,F2);
end
aviObject = close(aviObject);         % Cerramos el fichero avi
stop(vid);
aviObject2 = close(aviObject2);
stop(vid2);
```

[18].

De todas maneras, se podría implementar la solución a partir de este software pero al no obtener los exactamente los resultados buscados por medio de Matlab y al rechazar la idea de realizar el eye tracking en tiempo real, se buscaron programas externos para realizar la grabación que comenzaran y pararan la grabación exactamente en el mismo instante de tiempo.

Se encontró el software libre onuprova 3D, que permite grabar video procedente de dos fuentes. Automáticamente genera una previsualización de una imagen para poder verla en 3 dimensiones si se dispone de unas gafas con los filtros rojo y cyan. También permite ajustar las imágenes de las dos webcams para que se genere una imagen con más profundidad o con menos. El inconveniente que hace se haya descartado este software para este proyecto es que sólo se puede capturar el video de las dos webcams en un único clip con la imagen procedente de las dos cámaras en rojo y azul para permitir la visualización 3D. Este clip no es útil para trabajar con él en Matlab ya que no existen algoritmos que permitan detectar los ojos de un usuario en imágenes de estas características [14].

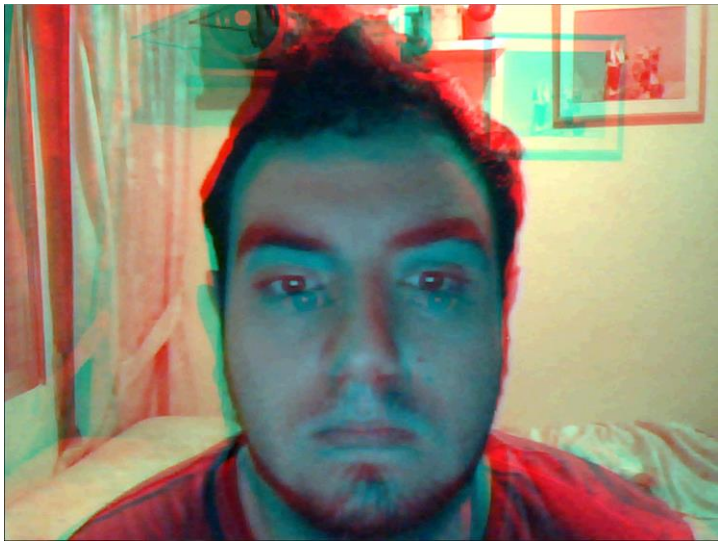


Figura 12: Tipo de imagen extraída del software onuprova.

Otro programa encontrado fue AMCap. Es posible descargar una versión de prueba, la cual ofrece la utilidad de capturar video procedente de dos webcams. Los inconvenientes son que no ofrece la posibilidad de guardar el video grabado con ningún

tipo de compresión, por lo que los clips pesan mucho, además tampoco permite iniciar la grabación de ambas cámaras a la vez, sino que manualmente hay que iniciar la grabación de una webcam y posteriormente de la otra. De la misma manera, no se pueden parar las grabaciones a la vez. Esto es un gran inconveniente para sincronizar la grabación de ambas cámaras, ya que se necesita que comiencen y terminen en el mismo instante de tiempo para poder hacer un seguimiento de la mirada fiable [15].

Al final de la búsqueda se encontró ISpy, que permite grabar las dos webcams comenzando la captura de ambas en el mismo instante. Como ISpy es un programa orientado a sistemas de seguridad, se puede activar una función que hace que se comience a grabar una cámara cuando se detecte movimiento en ella. Para la aplicación que se quiere implementar no se necesita esta funcionalidad, pero puede ser muy útil para futuros trabajos en el caso de cambiar la metodología de captura. Incluye la ventaja de que se puede manejar el software a través de comandos desde el terminal. Matlab ofrece la posibilidad de ejecutar comandos del terminal de Windows, por lo que podemos ejecutar el programa y sus funciones sin salir de Matlab. Los comandos más importantes que se pueden ejecutar son:

- ALLON permite encender y obtener una vista previa de todas las webcams conectadas y configuradas.
- ALLOFF permite apagar todas las cámaras.
- RECORD comienza la grabación de todas las cámaras de manera simultánea.
- RECORDSTOP detiene las grabaciones en curso.

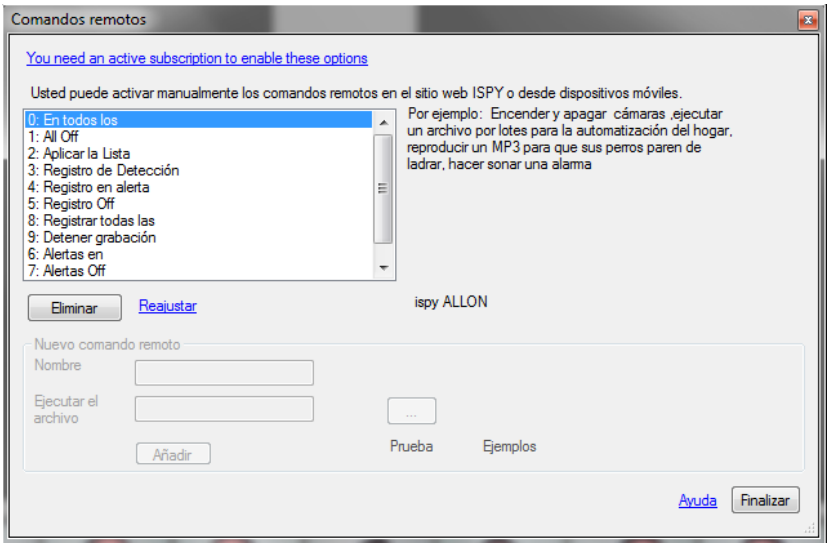


Figura 13: lista de comandos remotos ISpy.

Los archivos de video capturados por cada cámara se guardan en un directorio que se selecciona desde el programa, y dichos archivos toman el nombre de la fecha y la hora a la que han sido grabados.

3.3 Calibración

Al usar 2 cámaras se necesita una calibración previa para mantener unas distancias entre las cámaras y poder usar técnicas 3D de manera más precisa. Se realizó una búsqueda de herramientas de Matlab que permitieran calibrar las dos webcams. En la toolbox de Matlab 'vision' se ofrecen herramientas de calibración estéreo.

Se realizó una búsqueda de ejemplos de programas en Mathworks que permitan calibrar dos cámaras. Lo ideal sería encontrar un método que haga que cada vez que se ejecute el programa, compruebe si la distancia entre ambas cámaras es la misma, y si no se diera el caso, permitiera moverlas manualmente hasta que se ajusten. Para poder hacer esto se necesitaría poner alguna plantilla de referencia delante de ambas cámaras, como por ejemplo una cuadricula. En nuestro ejemplo no vamos a utilizar grabación en tiempo real, por lo que calibrar las webcams con una cuadricula puede ser muy tedioso ya que habría que grabar, comprobar si está calibrado y si no lo está, grabar de nuevo, y así sucesivamente. Para trabajos futuros esta sería una línea importante sobre la que investigar.

Se encuentra un programa que permite formar imágenes en 3 dimensiones que pueden servir para calibrar el montaje e incluso podría ayudar a detectar la cara, ya que en la reconstrucción 3D se pueden observar las distancias a las que se encuentran los objetos. El programa se llama *'Stereo Matching'*, implementado por Wim Abbeloos [19]. El programa dispone de una interfaz gráfica que permite elegir dos imágenes y formar una imagen estéreo introduciendo ciertos parámetros. Para que funcione es necesario tener instalada la toolbox de Matlab *'Image processing'*. El funcionamiento se basa en la utilización de la función *'disparity'* de la toolbox de Matlab.

La función *'disparity'* permite calcular el mapa de disparidad entre dos imágenes estéreo. El mapa de disparidad no es más que una representación gráfica de la distancia a la que se encuentra cada objeto dentro de una imagen 3D.

El programa tiene la siguiente interfaz gráfica:

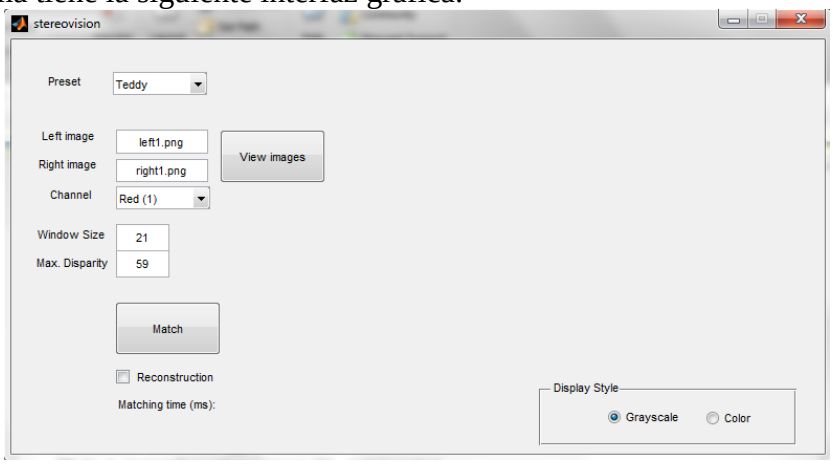


Figura 14: Interfaz gráfica de 'Stereo Matching'.

El programa dispone de diferentes ejemplos y configuraciones para cada caso. Al pulsar en *'view images'*, observamos las imágenes que se van a intentar procesar para calcular la distancia a la que está cada objeto.

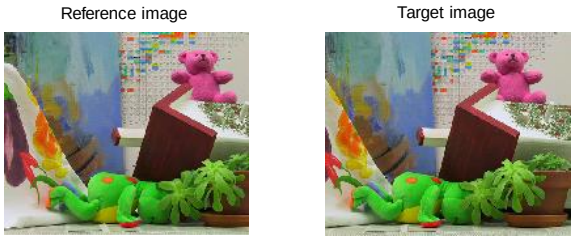


Figura 15: Imágenes de la configuración 'Teddy' de 'Stereo Matching'.

Cuando pulsamos en Match, el programa muestra un mapa de color que representa la distancia a la que estima que está cada punto de la imagen. Siendo rojo lo más cercano a la cámara y azul lo más lejano.

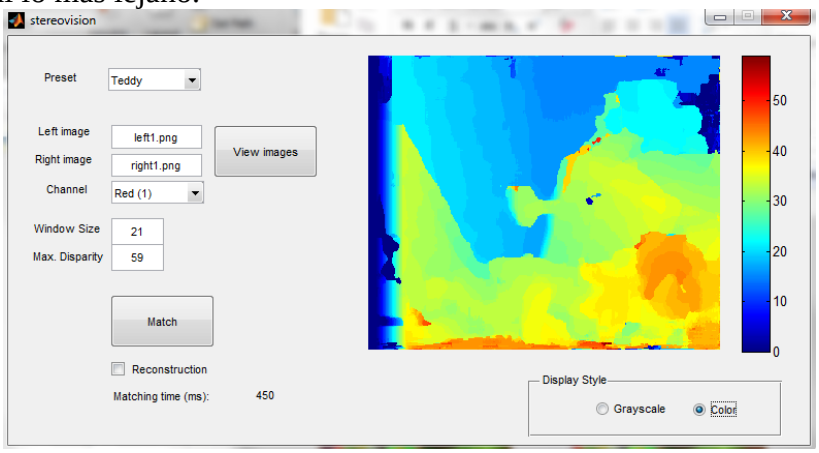


Figura 16: Mapa de color representando disparidad de 'Stereo Matching'

Al marcar la casilla 'Reconstruction', se genera una reconstrucción de ambas imágenes en tres dimensiones.

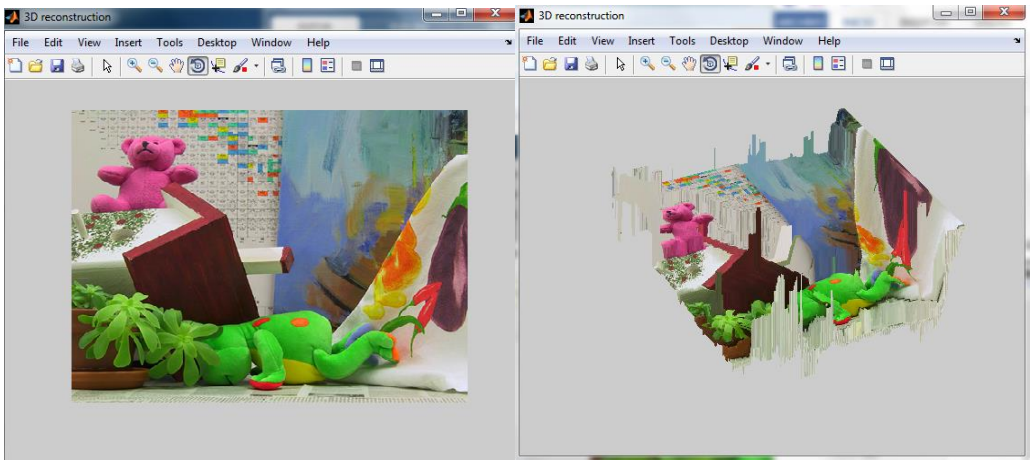


Figura 17: Reconstrucción 3D de 'Teddy' en 'Stereo Matching'.

Al observar los resultados con las imágenes de prueba, se pasa a intentar utilizar el programa para el objetivo del proyecto, con la imagen de un usuario capturada por las dos webcams de baja definición. Se intenta hacer la reconstrucción con la configuración más acertada, a continuación podemos observar el resultado.

Figura 18: Imágenes a reconstruir, caso práctico.

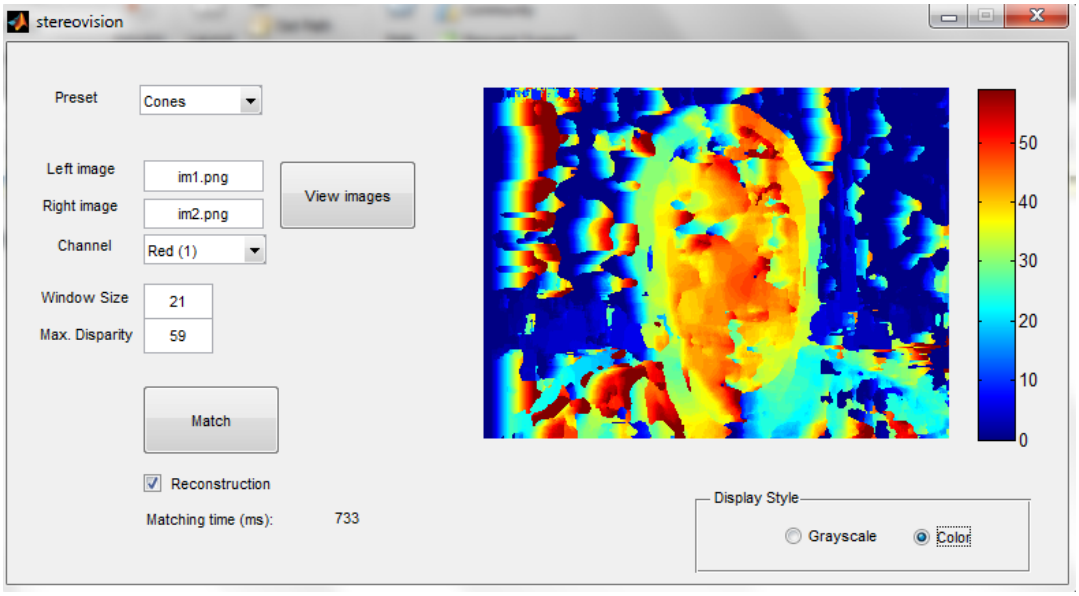


Figura 19: Mapa de disparidad, caso práctico.

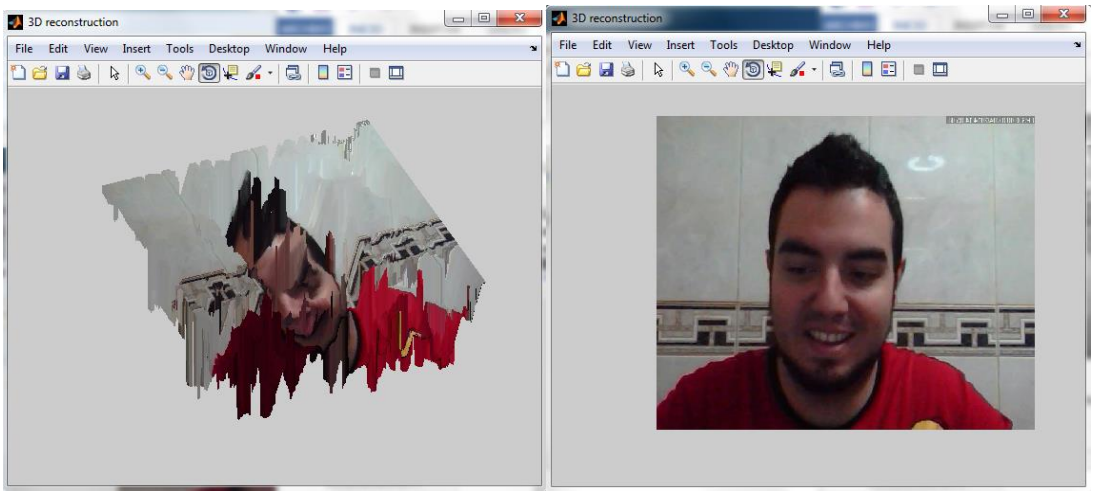


Figura 20: Reconstrucción en 'Stereo Matching' con un caso práctico.

Como podemos observar, la reconstrucción no es demasiado buena, y no tiene demasiada utilidad para la aplicación que se quiere implementar. Por este motivo no se entra en más detalle del funcionamiento de la aplicación 'Stereo Matching', pero se debe tener muy en cuenta para trabajos futuros.

La toolbox de Matlab *'computer vision'* tiene paquetes de *'Camera Calibration'* y *'Stereo vision'*. [20] La mayor parte de herramientas incluidas en esta toolbox pueden servir para hacer un programa que efectue una calibración 3D. Por ejemplo, *'detectFASTFeatures'*, *'detectHarrisFeatures'* y *'detectMinEigenFeatures'* permiten detectar las esquinas en las imágenes estéreo mediante tres metodologías diferentes. *'reconstructScene'* permite reconstruir una escena de tres dimensiones a partir del mapa de disparidad, que para calcularlo existe la función *'disparity'*. Es decir mediante esta toolbox es posible hacer un programa igual que *'Stereo Matching'* sin necesidad de programar arduos desarrollos matemáticos.

MathWorks también proporciona un ejemplo completo de calibración estéreo y reconstrucción de una escena 3D con la utilización de la toolbox anteriormente mencionada.

El ejemplo se puede observar en este link:
http://www.mathworks.es/es/help/vision/ug/stereo-vision.html

Para el programa que se desea implementar en este trabajo fin de grado es suficiente con realizar un montaje fijo que no sea necesario calibrar, sino que las cámaras siempre se encuentren en la misma posición respecto a la pantalla del ordenador. Esto permite que los algoritmos de extracción del eye-gaze se puedan especializar para dicho montaje con las mismas distancias.

El problema es que el usuario puede encontrarse a diferentes distancias entre un ensayo y otro, por lo que la estimación del punto al que se está mirando no sería acertado, debido a que la pantalla se encontraría a distancias diferentes cada vez. Para tener en cuenta las posibles variaciones de posición del usuario se decide reproducir un video de calibración cada vez que se vaya a utilizar la aplicación. Este video de calibración hará que el usuario mire a cinco posiciones distintas de la pantalla, y se registrará para cada posición la ubicación del iris. De esta manera se podrá estimar a donde está mirando la persona, se encuentre a la distancia que se encuentre.

3.4 Interfaz gráfica

Para hacer más sencillo el uso de la aplicación se podría programar una interfaz gráfica. Matlab permite programar interfaces gráficas de manera sencilla, incluso permitiendo reproducir videos o representar figuras en ellas mismas. Para el proyecto se diseña una simple interfaz para poder ejecutar el programa más fácilmente.

En la vista principal de Matlab, en las pestañas superiores, se pulsa en ‘new’, y a continuación en ‘Graphical User Interface’. Se abrirá una ventana que permite abrir una interfaz gráfica existente o crear una nueva. Entre las nuevas existen varias plantillas que pueden servir de base para una interfaz más elaborada.

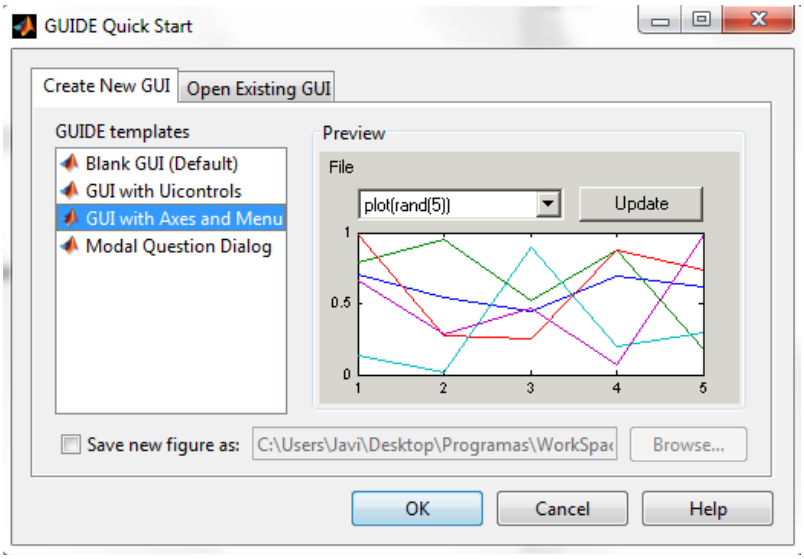


Figura 21: Guía de creación de interfaces gráficas de Matlab.

No es necesario programar la interfaz gráfica mediante comandos, sino que se dispone de un sencillo editor que permite ubicar y configurar los elementos gráficos más comunes, como por ejemplo, botones, texto estático, texto editable, casillas para marcar, etc.

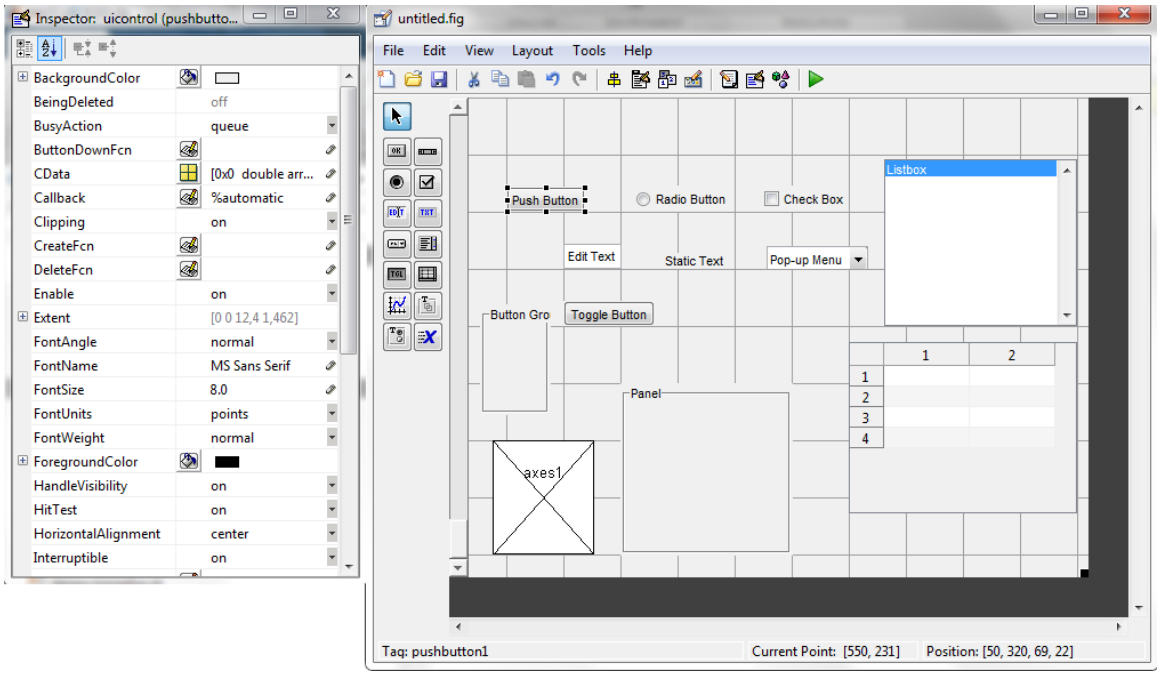


Figura 22: Herramienta de edición de interfaces gráficas de Matlab con los principales elementos gráficos. El botón está seleccionado y en la ventana de la izquierda se pueden seleccionar sus características.

Esta interfaz gráfica genera un archivo .fig que es donde está guardada la disposición de los elementos creados, pero a su vez se autogenera un archivo .m de Matlab que es desde el cual se tienen que programar las acciones que se quiere que realice cada elemento gráfico.

```
function varargout = pruebagui(varargin)
% PRUEBAGUI MATLAB code for pruebagui.fig
%      PRUEBAGUI, by itself, creates a new PRUEBAGUI or raises the existing
%      singleton*.
%
%      H = PRUEBAGUI returns the handle to a new PRUEBAGUI or the handle to
%      the existing singleton*.
%
%      PRUEBAGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PRUEBAGUI.M with the given input arguments.
%
%      PRUEBAGUI('Property','Value',...) creates a new PRUEBAGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before pruebagui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to pruebagui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help pruebagui

% Last Modified by GUIDE v2.5 23-Sep-2014 10:18:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @pruebagui_OpeningFcn, ...
                   'gui_OutputFcn',  @pruebagui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

Figura 23: Código Matlab de interfaz gráfica autogenerado.

3.5 Detección de la cara

Lo primero que se piensa es que para poder detectar los ojos, se debe detectar la cara. Se buscan programas hechos que puedan servir para este fin. En seguida se encuentra el algoritmo de Viola & Jones para compilar en Matlab, implementado en la toolbox *'vision'* dentro de *'object deteccion'* en el detector *'CascadeObjectDetector'*.

Los ingenieros Paul Viola de Mitsubishi Electric Research Labs y Michael Jones de Compaq CRL desarrollaron un algoritmo de detección de rostros en una imagen con un coste computacional muy bajo. Publicaron el día 13 de julio de 2001 un framework que constaba de dos partes principales: un algoritmo de detección de objetos que emplea la clasificación en cascada y un entrenador de clasificadores basado en AdaBoost.

El algoritmo alcanza altas tasas de detección y, a diferencia de otros algoritmos que utilizan información auxiliar, como: el color del pixel o diferencias en la secuencia de video, procesa solamente la información presente en una imagen en escala de grises (formato YUV). El algoritmo para la detección no utiliza directamente la imagen sino una representación de la misma llamada imagen integral. La obtención de esta representación se logra con tan solo unas pocas operaciones por pixel. Hecho esto, buscar características en subregiones de la nueva imagen se transforma en una tarea de tiempo constante, sin importar la escala de la subregión ni posición de la misma. Para determinar si en una

imagen se encuentra un rostro o no, el algoritmo divide la imagen integral en subregiones de tamaño variable y utiliza una serie de clasificadores (etapas), cada uno con un conjunto de características visuales. Este tipo de clasificación es denominada clasificación en cascada.

En cada etapa se determina si la subregión es un rostro o no. Si la subregión es aceptada como rostro entonces es evaluada por la siguiente etapa (más rigurosa) y si no, es discriminada. La clasificación en cascada garantiza la discriminación rápida de subregiones que no sean un rostro. Esto significa un ahorro considerable de tiempo, pues no se procesarán innecesariamente subregiones de la imagen que con certeza no contengan un rostro y solamente se invertirá tiempo en aquellas que posiblemente sí contengan. A la culminación sólo llega la imagen que consigue aprobar todas las etapas. Este método, según [22], tiene una tasa de verdaderos-positivos de 99,9%, mientras que tiene una tasa de falsos-positivos de 33,3% [21].

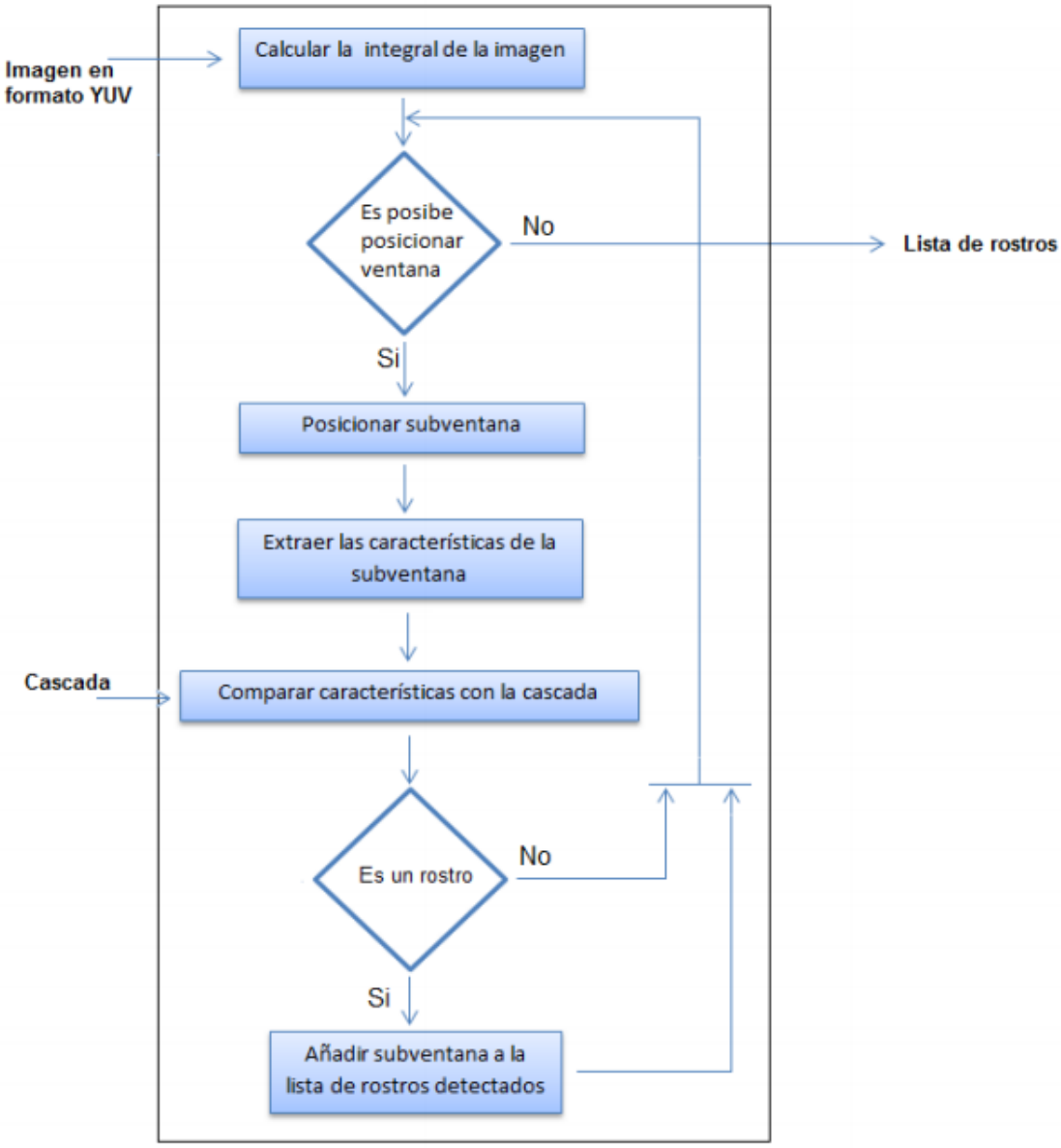


Figura 24: Proceso de detección de rostro en una imagen [21].

Matlab permite detectar únicamente los ojos mediante el detector en cascada, entonces no sería necesario detectar la cara del usuario para poder localizar los ojos, pero sí que sería muy interesante detectar la cara para posteriormente hacer un seguimiento de ésta y así contrarrestar los movimientos de la cabeza para que no afecten al tracking de los ojos.

El programa *'detect face parts'* encontrado en MathWorks permite detectar la cara, ojo derecho, ojo izquierdo, nariz y boca de un individuo. En este caso, hay partes que no se necesitan, pero gracias a este programa se aprende a utilizar la toolbox de detección proporcionada por Matlab. Para ejecutar el programa se requieren las toolboxes *'Computer visión system'* y *'Image Processing'* de Matlab [23]. Los resultados de la demostración que incluye el programa se representan en la figura 25.

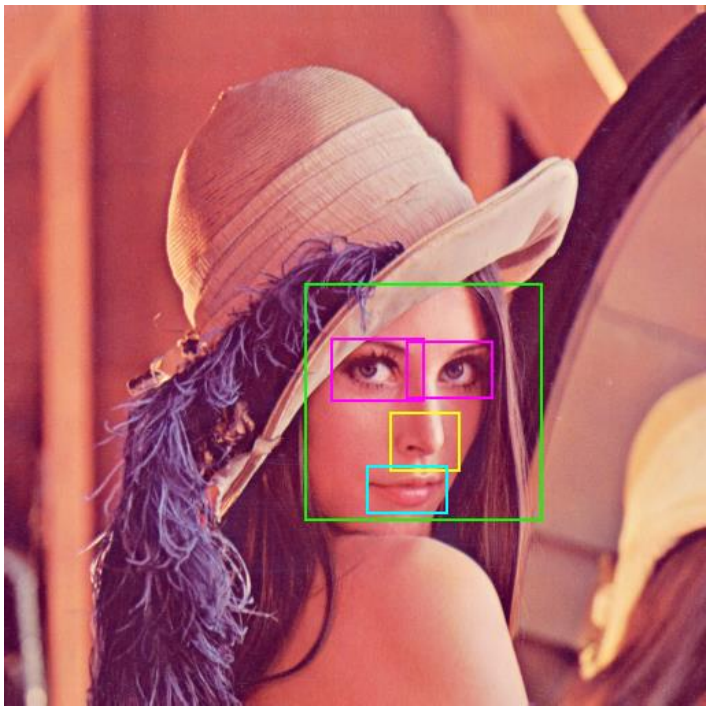


Figura 25: Detección de partes de la cara con imagen de demostración.

A continuación se comprueba cómo se comporta el algoritmo al pasarle dos imágenes capturadas con las webcams de baja definición:

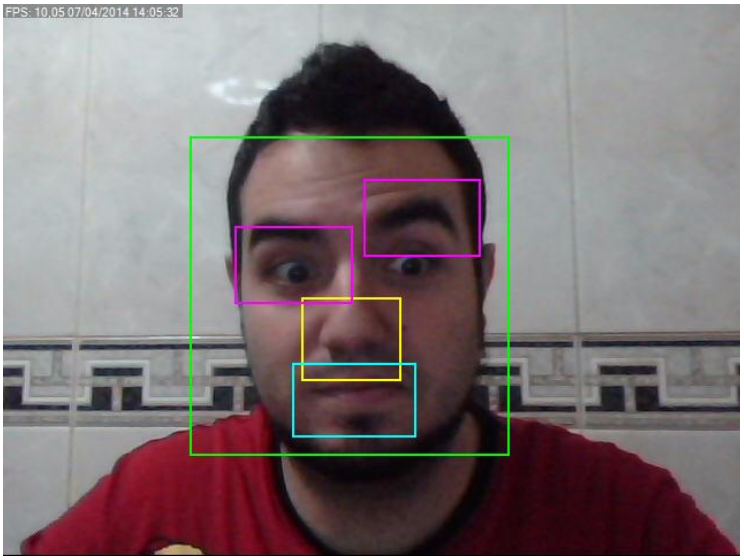


Figura 26: Detección de partes de la cara 1.

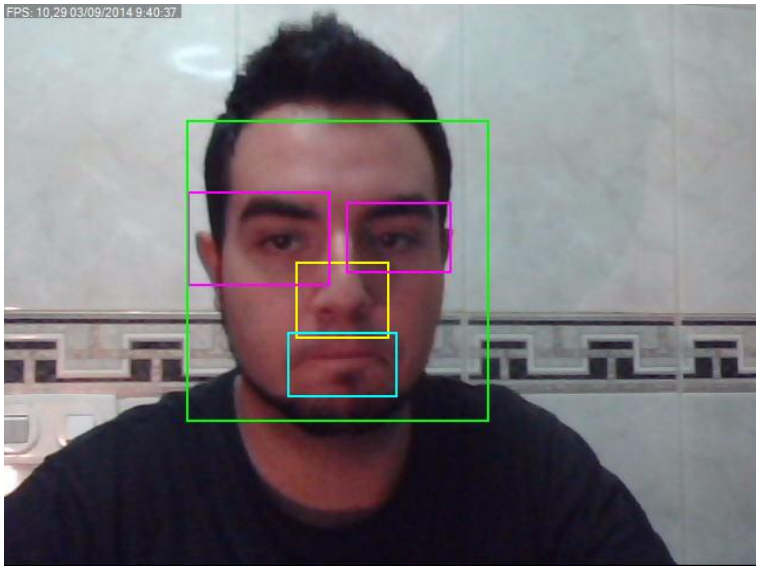


Figura 27: Detección de partes de la cara 2.

En la figura 26 se observa que el ojo derecho no se detecta correctamente. Pero la detección en la figura 27 es bastante acertada. El algoritmo también permite detectar varias caras.

Figura 28: Detección de múltiples caras y las partes de cada una de ellas.

3.6 Detección de los ojos

Una primera idea era capturar una imagen de cada una de las webcams y manualmente que el usuario señalara donde se ubica el centro de sus retinas, y de esta manera no errar a la hora de detectar los ojos. Este método es poco útil para casi todas las posibles aplicaciones de eye tracking, aunque si es verdad que para esta aplicación sería suficiente.

En la toolbox de Matlab *'vision'*, hay un paquete llamado *'object deteccion'* que incluye el detector *'CascadeObjectDetector'* que funciona con el algoritmo de Viola & Jones. Este algoritmo hace posible detectar los ojos de una persona mediante la detección en cascada. Explorando en la ayuda de Matlab se observa que el detector tiene seis posibles parámetros de entrada para poder detectar los ojos:

- EyePairBig. Detecta los dos ojos, este modelo está entrenado con imágenes grandes, tamaño [11 45].

- EyePairSmall. Detecta los dos ojos, este modelo está entrenado con imágenes pequeñas, tamaño [5 22].

- LeftEye o RightEye. Detecta el ojo izquierdo o el derecho por separado, este modelo se compone de clasificadores débiles basados en arboles de decisión.

- LeftEyeCART o RightEyeCART. Detecta el ojo izquierdo o el derecho por separado. Los clasificadores débiles que caracterizan este modelo son los llamados arboles de Cart *'CART-trees'*. Comparados con los arboles de

decisión los clasificadores basados en arboles de Cart son más capaces de modelar las dependencias de orden superior.

Dependiendo del tamaño y calidad de la imagen y del tiempo de procesado y de la precisión que se requiera, se elegirá un detector u otro. Los modelos LeftEye y RigthEye requieren un tiempo de procesado menor que LeftEyeCART y RigthEyeCART, pero por contrapartida son menos precisos. En cuanto a la calidad de la imagen, cuanta menos calidad, menos eficientes son los algoritmos que detectan los ojos por separado, ya que es muy frecuente que detecten el ojo izquierdo como derecho y viceversa. Por eso para imágenes de baja calidad se eligen los detectores que localizan ambos ojos. Cuál de los dos utilizar depende únicamente del tamaño de los ojos que se desean estimar en la imagen. Si la persona se encuentra cerca de la cámara, se debe utilizar EyePairBig, si por el contrario se encuentra lejos, los ojos serán más pequeños y se deberá utilizar EyePairSmall.

Una vez detectados los ojos, se podría guardar una plantilla de ellos en un perfil de usuario. Podrían existir varios perfiles y en cada uno de ellos se guardaría una plantilla con los ojos de esa persona. Con esta metodología podría ser más rápido buscar y encontrar los ojos en próximos ensayos con el programa.

3.7 Detección del iris

Una de las partes más complicadas, la baja resolución de la cámara hace que la detección del iris sea bastante delicada. En esta parte es en la que es más necesario una buena iluminación.

Ya se conocía de otras asignaturas de la carrera el método de la transformada de Hough circular, que comienza obteniendo el mapa de bordes de la imagen, la información obtenida se utiliza para deducir la ubicación de los centros (x, y) y los valores de los radios (r). Por último se lleva a cabo la detección de máximos. La desventaja de este método es la alta demanda de recursos computacionales, por eso esta opción es totalmente inadecuada para procesamientos en tiempo real [24]. Para el caso que se está implementando, que se descarte el procesamiento en tiempo real no es un problema, ya que ya estaba descartado.

Matlab tiene la función ‘imfindcircles’ que usa la transformada de Hough circular. Esta función, según [25], es robusta frente a la presencia de ruido y a las variaciones de iluminación, pero la gran desventaja es que la precisión de esta función está limitada cuando el valor del radio es menor de 10, y en casi todas las imágenes de ojos que procesamos, este valor es inferior.

Se intenta implementar un detector con ‘imfindcircles’ que se base en la transformada de Hough circular, pero los resultados no son para nada los requeridos.

Figura 29: Identificación de círculos mediante la transformada de Hough.

Como observamos, en la imagen completa ningún círculo se aproxima a la posición real del Iris. Se decide intentar detectar círculos con una segmentación de los ojos. (Previamente se detectan ambos ojos y después se segmenta la imagen).

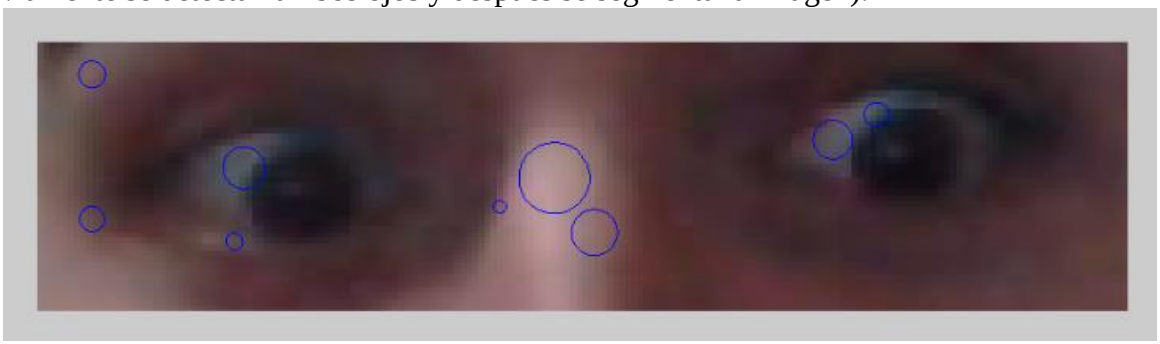


Figura 30: Transformada de Hough en una imagen con los ojos de cerca.

Después de obtener los resultados, se descarta esta técnica para el tipo de imágenes de baja definición con las que se trabaja en este proyecto.

Ahora se intenta detectar el centro del iris localizando espacialmente la mayor concentración de color negro. Este tipo de detección limitaría el uso de la aplicación a personas con los ojos oscuros. Se hace un sumatorio de una versión en blanco y negro de la imagen, primero en vertical, y después en horizontal. A partir de estos dos vectores, se encuentra el mínimo de cada uno de ellos. El mínimo será la fila o la columna de la matriz de la imagen donde más cantidad de negros haya. En el lugar que se crucen el mínimo de ambos vectores debería ser el punto donde se localiza el iris. Este algoritmo no funciona nada bien, ya que es muy susceptible a los cambios de iluminación y sobre todo, en cuanto el recuadro de detección del ojo incluye un fragmento de la ceja del usuario, el algoritmo detecta la mayor concentración de negros en la ceja.

Figura 31: Detección del iris por cantidad de negros.

Por último se buscan programas ya implementados por otras personas, cabe destacar que hay que diferenciar detección del iris, del reconocimiento del iris. La detección únicamente trata de localizar el iris en una imagen, el reconocimiento diferencia el iris de una persona del iris de otra persona. Se encuentran varios ejemplos de detección en MathWorks, pero para todos ellos es necesario tener la imagen de un solo ojo para detectar el iris. Por ello lo primero que hay que hacer es, una vez detectados los ojos, segmentar la zona y separar el ojo derecho del ojo izquierdo. Después pasar estas imágenes por el detector de iris y con el iris detectado, reconstruir la imagen original para poder ubicar las coordenadas de los iris en la imagen completa.

Se encuentra en MathWorks un programa basado en el operador integro-diferencial de Daugman [26]. La aplicación funciona bastante bien, sobre todo a la hora de identificar el iris. También permite identificar la pupila en imágenes de alta calidad, pero en imágenes de baja definición cómo las que se usan en este programa, la identificación de la pupila no es precisa. Esto no supone un problema para nuestro cometido, ya que con identificar el centro del iris es suficiente. Esta es la solución elegida para la implementación del eye-tracker de ejemplo. Se explicará con más detenimiento el funcionamiento del programa en el capítulo 4.

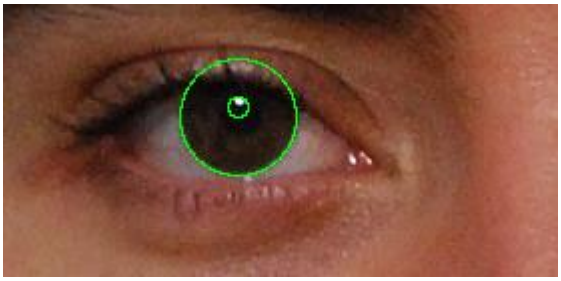


Figura 32: Detección del iris y pupila en imagen de alta calidad mediante el programa 'iris integrodifferential'.

Figura 33: Detección del iris y retina en imagen de baja calidad mediante el programa 'iris integrodifferential'.

El operador integro-diferencial fue propuesto por Daugman [28] en su sistema original de reconocimiento de iris. La base del operador consiste en detectar cambios bruscos en el tono de gris a lo largo de circunferencias en la imagen, con la idea de que un cambio brusco se traduce en un máximo en la derivada de los tonos de gris de las circunferencias centradas en el punto (x0, y0). La expresión del operador para encontrar el círculo de centro (x0, y0) y radio r en la imagen I es la siguiente:

$$\max_{x_0,y_0,r} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{x_o,y_o,r} \frac{I(x,y)}{2\pi r} \mathrm{ds} \right|$$

Figura 34: Fórmula del operador integro-diferencial [27]

Donde:

- *s* representa el contorno sobre el cual se hace la integración, en este caso, un contorno circular
- *Gs(r)* es una función Gaussiana que define el nivel de detalle en el que se trabaja: mientras más precisión se desea, menor será el parámetro de suavización σ.

Este operador funciona iterativamente, buscando primero una aproximación y luego obteniendo resultados más precisos. La cantidad de niveles de detalle puede ser ajustada para que el operador trabaje de forma más veloz. Una propiedad importante de este operador es que se puede generalizar para detectar otros tipos de curvas parametrizables. Cambiando el contorno *s* de una circunferencia a una parábola, es posible construir de manera similar un detector de parábolas, que luego puede ser utilizado para detectar los parámetros de los párpados [27] [28].

Otras posibles alternativas serían programar o encontrar algoritmos que se basen en otras metodologías, por ejemplo en la Transformada de Hough probabilística, que es una variación de la transformada de Hough mucho más rápida y puede permitir la detección de círculos en tiempo real o los nuevos métodos de algoritmos genéticos (GA, por su nombre en inglés: *'Genetic Algorithms'*) y Autómatas de Aprendizaje (LA por su nombre en inglés *'Learning Automata'*). Este último método es muy interesante, puede servir para encontrar la solución de complejos problemas de optimización [24].

3.8 Seguimiento (tracking)

Existen varios programas de seguimiento de los ojos, pero en Matlab no hay ninguno con código abierto que no utilice Opencv como base de programación. De todas maneras se estudia el funcionamiento del programa de Timm & Barth, ya que proporciona unos resultados muy favorables, como demuestra este video:

http://www.youtube.com/watch?v=aGmGyFLQAFM#t=164

El programa trabaja localizando el centro del ojo por medio de gradientes [29]. El tracking se realiza con herramientas de Opencv.

Se observa que la mayor parte de los programas de eye tracking están implementados en Opencv [30]. Opencv es una biblioteca libre, gratuita para el uso académico y comercial. Cuenta con interfaces de diferentes lenguajes de programación como C, C++, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android. Opencv fue diseñado para obtener una eficiencia computacional muy grande para aplicaciones de procesamiento de video y de imágenes en tiempo real.

Matlab permite compilar códigos de otros lenguajes de programación como Opencv o C++. Se puede integrar la librería de *'computer vision'* de Opencv con Matlab utilizando la interfaz *MEX*. De esta manera se pueden usar los algoritmos de Opencv sobre adquisición y tratamiento de imagen, que son muy potentes y fiables [31].

Investigando el algoritmo de Timm & Barth se encuentra un programa llamado *'EyeLike'* realizado por Tristan Hume [32]. Este programa está escrito en Opencv. Se trata de ejecutar desde Matlab mediante las librerías *MEX*, pero la versión instalada de Matlab no es compatible con la versión del programa en Opencv, por lo que no se puede probar. Sería un buen comienzo para futuros trabajos si se consiguen solventar los problemas de compatibilidad.

En MathWorks se encuentra un programa de ejemplo del seguimiento de una cara y gracias a ello se descubre el apartado de tracking en la toolbox *'vision'* de Matlab [33]. El programa detecta la cara y posteriormente hace tracking basado en histogramas.

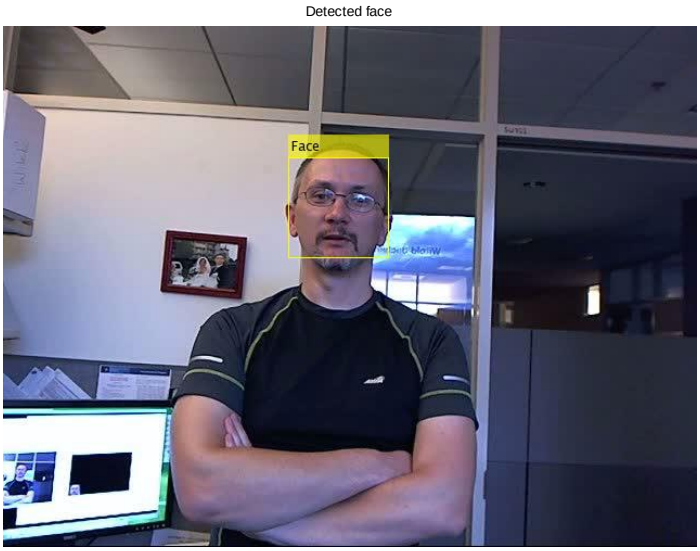


Figura 35: Detección de la cara.

Video del tracking de la cara:
https://www.dropbox.com/s/pl8m36l1mtojoyn/facetrackexamplehistogram.avi?dl=0

El *'HistogramBasedTracker'* utiliza el algoritmo '*CAMShift'*, el cual da la capacidad de seguir un objeto utilizando el histograma de los valores de los pixeles.

En la toolbox de visión de Matlab hay un paquete dedicado a la estimación de movimiento y al tracking [34]. Consultando la ayuda de Matlab podemos ver que opciones hay disponibles, estas son las funcionalidades que implementan:

Análisis del movimiento y seguimiento:

- assignDetectionsToTracks, asigna detecciones para tracking multi objeto.
- vision.BlockMatcher, estima el movimiento entre imágenes o frames de video.
- vision.ForegroundDetector, detecta los objetos en primer plano utilizando modelos gaussianos.
- vision.HistogramBasedTracker, sigue un objeto en un video utilizando técnicas basadas en histogramas.
- configureKalmanFilter, crea un filtro de Kalman para hacer tracking de objetos.
- vision.KalmanFilter, filtro de Kalman.
- vision.OpticalFlow, estima la velocidad de los objetos.
- vision.PointTracker, hace tracking de puntos en un video utilizando el algoritmo de 'Kanade-Lucas-Tomasi (KLT)'.
- vision.TemplateMatcher, localiza una plantilla en una imagen.

El *'PointTracker'* se adapta perfectamente a lo que se busca en este proyecto. Pasándole las coordenadas de donde se encuentran los iris, el 'point tracker' hace un seguimiento de su movimiento. Esta herramienta también permitiría hacer tracking de la cara y así contrarrestar su movimiento para hacer el eye tracking más robusto.

El algoritmo de ‘Kanade-Lucas-Tomasi’, utiliza pirámides de imágenes, su objetivo es generar una imagen piramidal, donde cada nivel tenga una reducción de resolución con factor 2 respecto al nivel previo.

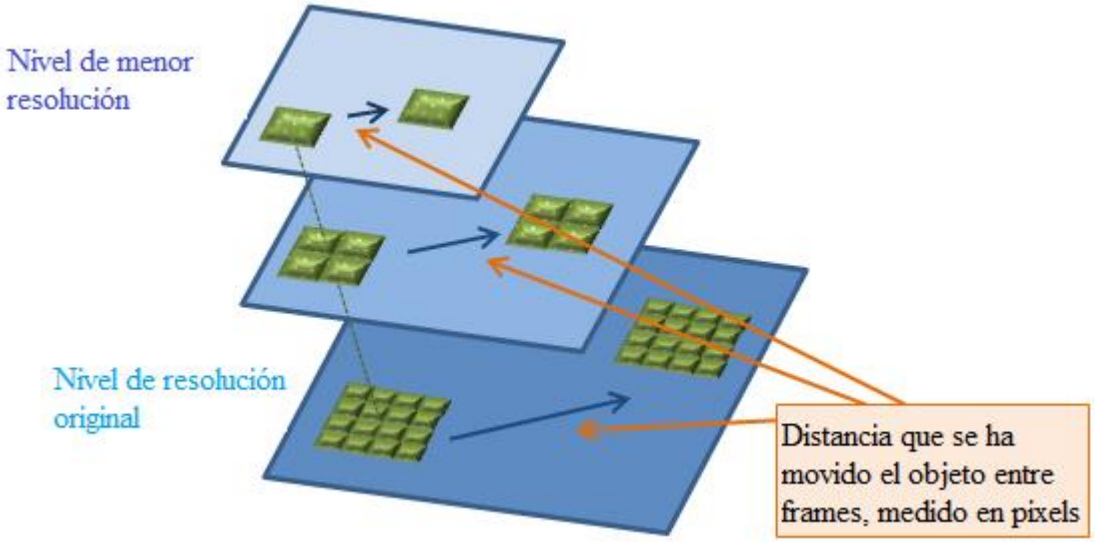


Figura 36: Niveles piramidales [Extraído de la ayuda de Matlab].

El algoritmo KLT desarrolla una búsqueda iterativa para encontrar el nuevo pixel donde se ubica cada punto. Se encontrará cuando se llegue a la convergencia, normalmente el algoritmo converge a las 10 iteraciones. En el capítulo 4 se explicará cómo ejecutar este algoritmo en Matlab para hacer el seguimiento de los ojos.

3.9 Extracción del punto de observación

Por último se quiere extraer el punto de observación o mirada: el eye gaze. Ahora es cuando tenemos que aprovechar que todo el trabajo se haya hecho por duplicado, en la grabación de ambas webcams. Se buscan métodos y estimaciones basadas en técnicas estéreo y métodos de 3 dimensiones. Directamente se explora la toolbox de visión estéreo de Matlab [20].

Por ejemplo, *‘epipolarLine’* computa las líneas epipolares en imágenes estéreo. Las líneas epipolares son las intersecciones de un plano epipolar con un plano de imagen. Un plano epipolar es todo plano que contenga a la línea de base. La línea de base es la línea que une los centros de ambas cámaras.

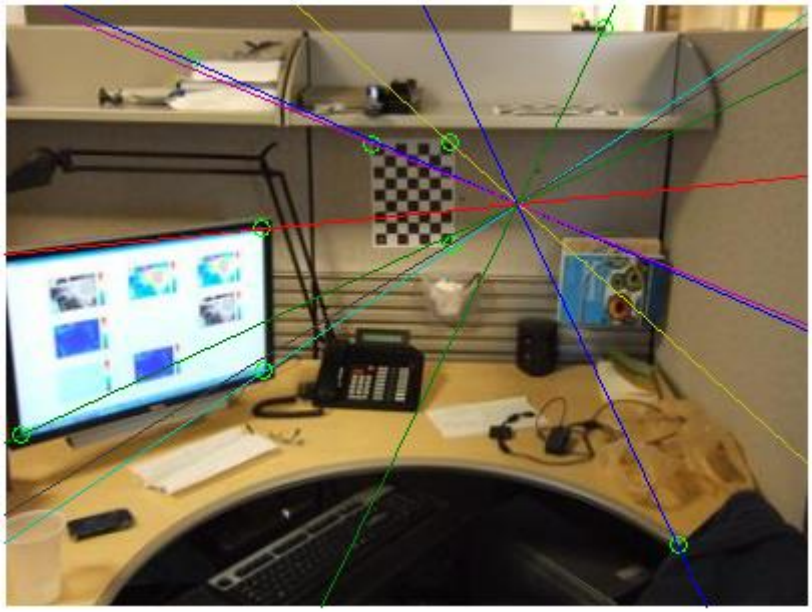


Figura 37: Líneas epipolares en una primera imagen.

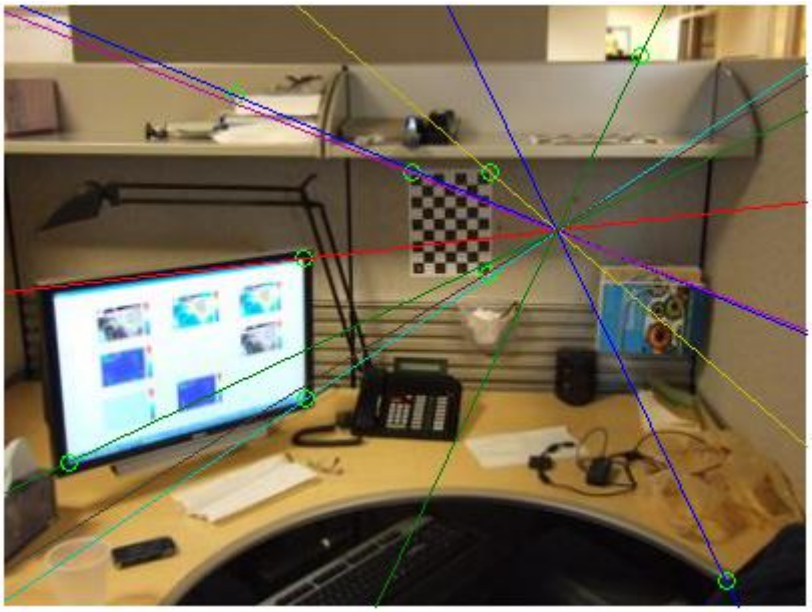


Figura 38: Líneas epipolares en una segunda imagen.

Este método nos permitiría extraer información sobre las distancias entre una cámara y otra y así poder estimar mejor la distancia al monitor y el punto al que está mirando el usuario.

También podría resultar útil *'estimateGeometricTransform'* que estima la transformada geométrica de un par de puntos encontrados o *'extractFeatures'* que extrae de las imágenes los puntos que mejor describen la imagen.

Una vez se ha encontrado una metodología 3D para poder caracterizar la distancia a la que se encuentran los iris del usuario, hay que pensar una forma de estimar a que punto del monitor está mirando a cada momento el usuario.

Se podría hacer tracking de los ojos y a su vez ir estimando el punto de observación. El mayor inconveniente es que el procesado puede ser lento y ralentizará el eye-tracking. Otro método es guardar para cada frame la posición de los ojos en ambas cámaras, y posteriormente ejecutar el algoritmo que detecte el eye-gaze.

El método utilizado para el ejemplo implementado en el trabajo es uno inventado y diseñado por el alumno, con técnicas muy básicas de distancias. Se procede a explicarlo en el capítulo 4.

Capítulo 4

Diseño técnico de la posible solución

4.1 Organización e interfaz gráfica

Se quiere que la organización y la manera de ejecutar el programa sean lo más sencillas posibles, por lo que se decide implementar una interfaz gráfica con una serie de botones que realizan las siguientes funcionalidades:

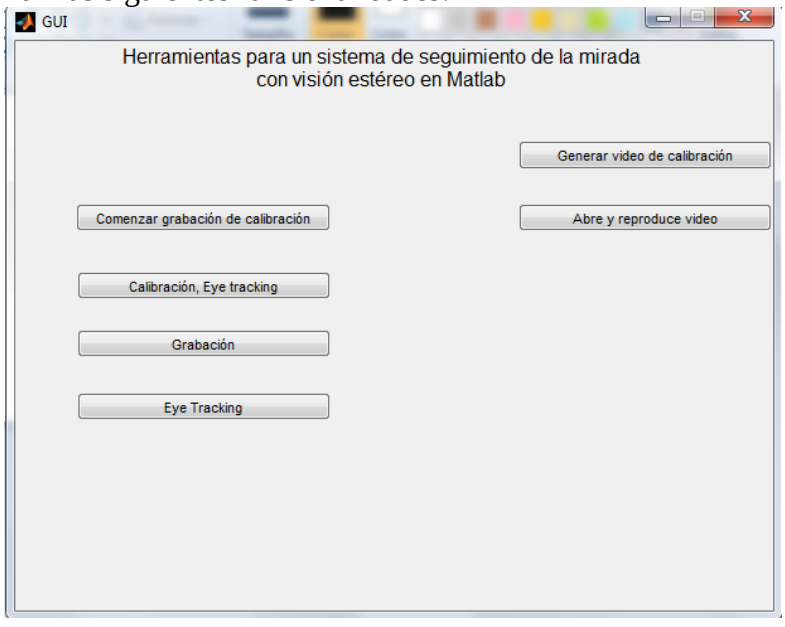


Figura 39: Interfaz gráfica de la aplicación.

- Generar video de calibración: se genera el video de calibración. https://www.dropbox.com/s/hnm6zbulwatyqi9/calibrationvideo.avi?dl=0

- Abre y reproduce video: abre una sencilla interfaz que permite seleccionar un video del directorio local en formato avi o mp4. Una vez seleccionado se reproduce en una figura de Matlab.

- Comenzar grabación de calibración: se graban ambas webcams mientras el usuario sigue con la mirada el video de calibración. video demostrativo: https://www.dropbox.com/s/srj9mgl1k58j613/Grabacion%20de%20video.wmv?dl=0

- Calibración, Eye tracking: Aquí se hace el eye-tracking y el eye gaze para las grabaciones de calibración. Se guarda una configuración con la calibración por si se quiere volver a ejecutar el eye tracking sin necesidad de volver a calibrar el programa. video demostrativo: https://www.dropbox.com/s/fqbuffn1aup8yqs/Ejecuci%C3%B3n%202.avi?dl=0

- Grabación: se graban las webcams desde iSpy durante 20 segundos.

- Eye tracking: se hace el eye-tracking y el eye-gaze con la última configuración de calibración guardada.

Para futuros trabajos podría resultar útil tener el código que permite abrir y reproducir un video, el código es el siguiente:

```
[FileName Path]=uigetfile({'*.avi';'*.mp4';},'Abrir Video');
if isequal(FileName,0)
    return
else
    a=VideoReader(strcat(Path,FileName));

    nFrames = a.NumberOfFrames;
    vidHeight = a.Height;
    vidWidth = a.Width;

    % Crea la estructura del video.
    mov(1:nFrames) = ...
        struct('cdata', zeros(vidHeight, vidWidth, 3, 'uint8'),...
        'colormap', []);

    % Lee un frame cada vez.
    for k = 1 : nFrames
        mov(k).cdata = read(a, k);
    end

      % Ubica el la ventana del video en una posicion y tamaños
      % prefijados.
    hf = figure;
    set(hf, 'position', [150 150 vidWidth vidHeight])

    % Reproduce el video al framerate indicado.
    movie(hf, mov, 1, a.FrameRate);
```

4.2 Generación de video de calibración

Para no usar un programa externo se decide crear el video de calibración desde el propio Matlab.

Lo primero que se hace es crear un video de 120 frames, es decir, a 10 frames por segundo, un video de 12 segundos. Donde los primeros 2 segundos se muestra el texto de la figura 40, y los restantes 10 segundos son frames en negro.

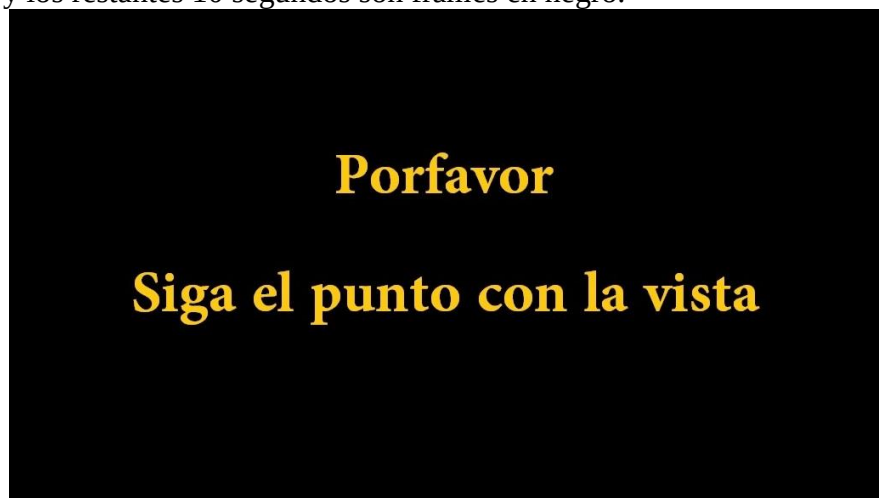


Figura 40: Imagen creada para el video de calibración.

A continuación se genera un punto que recorrerá el video, que será el punto que el usuario deberá seguir con la vista. Se definen las coordenadas donde se va a querer que se ubique el punto, y frame a frame se va insertando el punto mediante la función 'patch' de Matlab.

Estas son las coordenadas de los puntos para los 120 frames del video:

```
for j=first_frame:last_frame
    if j<=20
       pix_pos(j,1)=500;
       pix_pos(j,2)=400;
    end
    if j<=40&&j>20
       pix_pos(j,1)=500;
       pix_pos(j,2)=280;
    end
    if j<=60&&j>40
       pix_pos(j,1)=20;
       pix_pos(j,2)=20;
    end
    if j<=80&&j>60
       pix_pos(j,1)=1000;
       pix_pos(j,2)=20;
    end
    if j<=100&&j>80
       pix_pos(j,1)=20;
       pix_pos(j,2)=560;
```

```
    end
     if j<=120&&j>100
       pix_pos(j,1)=1000;
       pix_pos(j,2)=560;
    end
end
```

Aquí es donde se inserta el punto frame a frame:

```
for n=first_frame:last_frame
      %Leemos el video
      imgRgb1 = read(readerobj,n);
      imshow(imgRgb1);
      %Se inserta el punto con la función patch, el ultimo parámetro es
      %el color del punto (RGB) [1 0 0] es rojo.
      h1=patch(sin(x)*radius+(pix_pos(i,1)/D_rate),(cos(x)*radius+(pix_p
os(i,2)/D_rate)),[1 0 0]);
      F = getframe(gca);
      F.cdata=F.cdata(1:end-1,1:end-1,:);
      mov = addframe(mov,F);
      delete(h1);
      if i<last_frame

            i=i+1;

      end

end
```

4.3 Grabación del video de calibración

El primer paso es ejecutar el comando para encender las cámaras desde iSpy. Para ello hay que utilizar la función 'system' de Matlab, que permite ejecutar comandos de la consola sin salir del programa. Primero, se ejecuta el programa y después se introduce el comando de iSpy.

```
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands ALLON');
```

A continuación, en el caso de grabar las webcams mientras se reproduce el video de calibración, se tiene que preparar el video de calibración para poder reproducirlo a la vez que se graban las cámaras. Una vez cargado, se ejecuta la grabación de iSpy, se reproduce el video de calibración, y una vez terminada su reproducción, se para la grabación de iSpy.

```
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands
RECORD');
movie(hf, mov, 1, a.FrameRate);
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands
RECORDSTOP');
```

Desde que iSpy comienza la grabación, hasta que se empieza a reproducir el video, pasan unos segundos de procesamiento que hay que tener en cuenta para calibrar el seguimiento del punto de la mirada.

En el caso de querer grabar las cámaras sin el video de calibración, simplemente se ejecutan los comandos de iSpy, y en vez de reproducir el video entre la grabación, se incluirá el comando *'pause'* de Matlab que permite esperar ciertos segundos hasta ejecutar el siguiente comando. En el caso de este programa, al pulsar en el botón 'grabación' de la interfaz gráfica, se graban durante 20 segundos ambas webcams.

```
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands ALLON');
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands
RECORD');
pause(20);
system('cd C:\Program Files\iSpy\iSpy (64 bit) && iSpy commands
RECORDSTOP');
```

4.4 Eye tracking

El primer paso es permitir al usuario seleccionar los dos videos desde el directorio local. Primero se seleccionará el video procedente de una webcam, y posteriormente el de la otra webcam. El nombre de los videos que guarda iSpy es la fecha y hora de la grabación.

```
[FileName Path]=uigetfile({'*.mp4';},'Abrir Video 1');
if isequal(FileName,0)
    return
else
    videoFileReader1 = vision.VideoFileReader(strcat(Path,FileName));
end
[FileName Path]=uigetfile({'*.mp4';},'Abrir Video 2');
if isequal(FileName,0)
    return
else
    videoFileReader2 = vision.VideoFileReader(strcat(Path,FileName));
end
%Inicializamos variables
videoPlayer = vision.VideoPlayer('Position', [100, 100, 680, 520]);
shapeInserter = vision.ShapeInserter('BorderColor','Custom',...
    'CustomBorderColor',[1 0 0]);
```

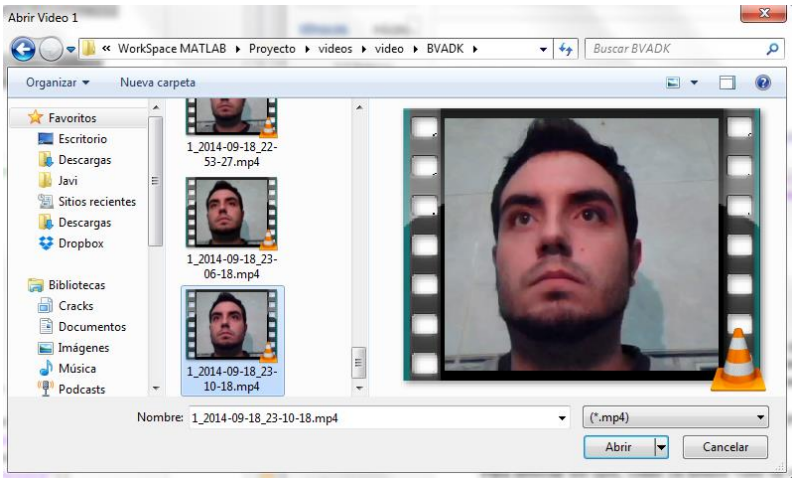


Figura 41: Ventana para seleccionar el primer video.

4.4.1 Detección de los ojos

Ahora debemos extraer un frame de cada video para detectar los ojos en él. El frame elegido no será el primero, sino que se dejarán pasar 3 segundos para obtener el frame. (Muchas veces el usuario está mirando a otro lugar nada más empezar la grabación).

Todo el procedimiento se tiene que hacer por duplicado para los dos videos procedentes de las dos webcams.

Para detectar los ojos, como ya hemos visto en el capítulo 3 se elige el algoritmo de Viola & Jones. Gracias a la toolbox *'vision'* de Matlab, disponemos del detector *'CascadeObjectDetector'*.

Primero debemos elegir con que detector hacer la identificación de los ojos. Mediante la repetición de muchas pruebas, se observa que los detectores RightEye y LeftEye, que detectan los ojos por separado, se confunden mucho más que los detectores que funcionan con ambos ojos. Muchas veces ocurre que se detectan dos ojos izquierdos o dos ojos derechos. Entre los detectores que detectan ambos ojos se elige 'EyePairBig' porque después de hacer varios ensayos, se demuestra que hacen una mejor detección para el tipo de imágenes que se procesan en este programa. (Los ojos en las imágenes se encuentran cerca de la cámara, por lo tanto son más grandes y por eso funciona mejor este detector).

Con los dos frames extraídos de los dos clips de video se sigue el mismo procedimiento:

- Se crea el detector y se ejecuta la detección mediante el comando *'step'*.

```
EyeDetect = vision.CascadeObjectDetector('EyePairBig');
BB=step(EyeDetect,I);
%BB son las coordenadas del recuadro que identifica los ojos de la
siguiente forma [x y anchura altura]
```

- Se comprueba si se ha detectado algún par de ojos en la imagen, si no es así, se crea y se ejecuta una nueva detección, esta vez con 'EyePairSmall'. Si no se encuentran los ojos de nuevo, se devuelve un mensaje de error.

- Existe una variable *booleana* que se pone en verdadero cuando se quieren mostrar las figuras de los pasos de la detección, y que se pone en falso cuando no se quieren mostrar. En la primera detección de los ojos, se muestran las figuras con los ojos detectados. Pero cuando el algoritmo de tracking se pierde y vuelve a llamar a la detección, las figuras no se muestran.

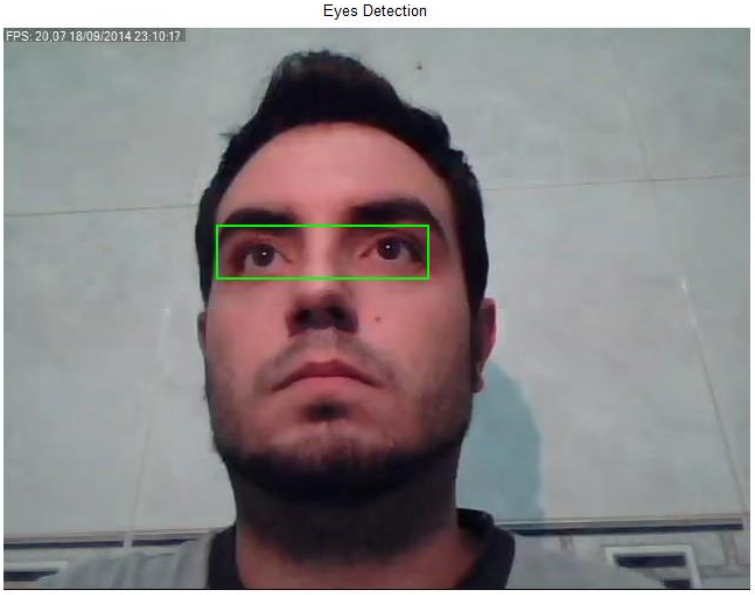


Figura 42: Detección de los ojos del frame de la primera cámara.

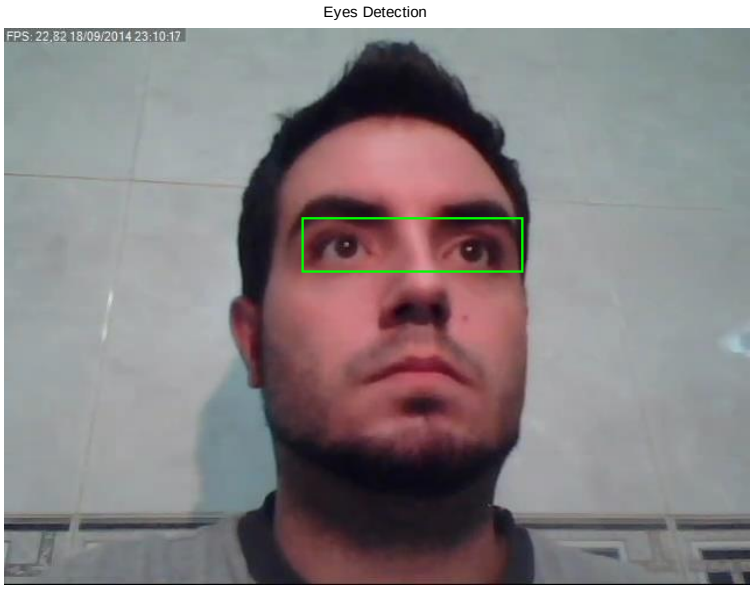


Figura 43: Detección de los ojos del frame de la segunda cámara.

4.4.2 Segmentación de los ojos

Para poder detectar el Iris con el programa elegido, es necesario segmentar ambos ojos por separado. Lo primero que hacemos es segmentar la zona de los ojos con el comando 'imcrop'. (Se elige la primera posición de 'BB' ya que puede ser que se hayan detectado más de un par de ojos)

```
Ieyes=imcrop(I,BB(1,:));
```

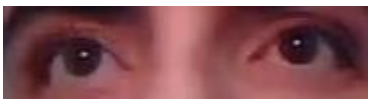


Figura 44: Segmentación de los ojos 1.

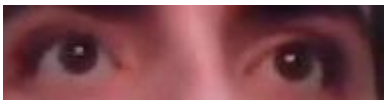
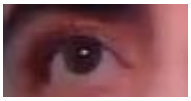
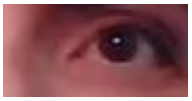


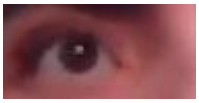
Figura 45: Segmentación de los ojos 2.

Una vez segmentada la zona, se debe separar el ojo izquierdo del derecho, para ello se divide la imagen a la mitad.

```
IeyeL=imcrop(Ieyes,[1,1,round(BB(1,3)/2),BB(1,4)]);
IeyeR=imcrop(Ieyes,[round(BB(1,3)/2),1,BB(1,3),BB(1,4)]);
```







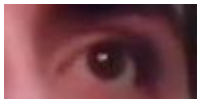


Figura 46: Segmentación de cada uno de los ojos para los dos frames de cada cámara.

Ahora es cuando se hace la detección de iris, que se explicará en el siguiente apartado. La detección de los iris devuelve las coordenadas del punto central del iris, que en cada imagen individual de cada ojo son coordenadas diferentes a las de la imagen

completa. Por lo tanto hay que calcular las coordenadas en la imagen grande a partir de las coordenadas que devuelva la detección del iris. Esto se hace mediante geometría básica de coordenadas utilizando las posiciones del recuadro original que identificaba los ojos (‘BB’). Hay que tener en cuenta que el algoritmo de detección de iris devuelve las coordenadas al revés, es decir (y, x).

```
%CiL son las coordenadas del ojo izquierdo (y,x,radio)
%CiR son las coordenadas del ojo derecho (y,x,radio)
%La función drawcircle dibuja un círculo en una imagen
CoordinatesLeftEye=ciL+[BB(1,2) BB(1,1) 0];
im=drawcircle(I,CoordinatesLeftEye(1:2),CoordinatesLeftEye(3));
CoordinatesRightEye=ciR+[BB(1,2) (BB(1,1)+round(BB(1,3)/2)) 0];
imdef=drawcircle(im,CoordinatesRightEye(1:2),CoordinatesRightEye(3));
```

4.4.3 Detección de los iris

Al programa de detección de iris se le van a pasar cuatro imágenes de 4 ojos. Ojo derecho de la primera cámara, ojo izquierdo de la primera cámara, ojo derecho de la segunda cámara y ojo izquierdo de la segunda cámara. Este algoritmo devolverá las coordenadas del centro del iris y el radio del iris para cada uno de los ojos.

Para implementar esta parte se ha utilizado el programa *‘Iris segmentation using Daugman’s integrodifferential operator’* de Anirudh Sivaraman [26]. El principio de funcionamiento de este programa ya ha sido explicado en el capítulo 3. Ahora procedemos a explicar las entradas y las salidas de dicho programa.

La función principal es la función *‘thresh’* cuyos parámetros de entrada son: la imagen, el radio mínimo del iris a detectar y el radio máximo del iris a detectar. Los parámetros de salida son las coordenadas y radio del iris, las coordenadas y radio de la pupila y la imagen del ojo con ambos círculos dibujados en dichas partes del cuerpo. La pupila no se necesita en este programa.

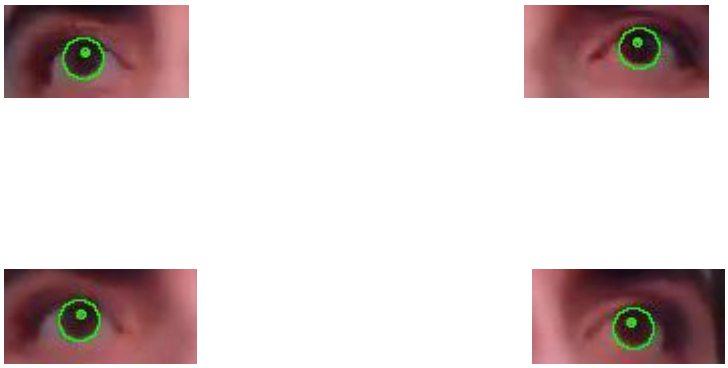


Figura 47: Los ojos con los iris y pupilas detectadas.

Para elegir los máximos y mínimos radios de iris posibles, se estudia poner valores fijos, pero no es algo correcto, ya que en cuanto el usuario esté más lejos de la cámara de lo normal, la detección no funcionará. Es por esto que se eligen radios dependiendo del tamaño de la caja que identifica los ojos. Después de hacer varias pruebas se concluye que los mejores valores para la detección son: que el radio mínimo sea una sexta parte de la altura del recuadro y que el radio máximo sea la mitad de la altura del recuadro (el iris tendría que ocupar todo el recuadro).

```
[ciL,cp,oL]=thresh(IeyeL,round(BB(1,4)/6),round(BB(1,4)/2));
[ciR,cp,oR]=thresh(IeyeR,round(BB(1,4)/6),round(BB(1,4)/2));
```

El detector del iris implementa estas funciones:

- Lineint, función que calcula la línea integral normalizada alrededor de un contorno circular.
- Drawcircle, que dibuja círculos en una imagen.
- Partiald, función que encuentra la derivada parcial.
- Search, función que detecta la pupila.

Figura 48: Iris detectados de la primera imagen.

Figura 49: Iris detectados en la segunda imagen.

4.4.4 Seguimiento de los iris

Una vez tenemos los puntos centrales de los iris en ambas cámaras, necesitamos seguir su movimiento a lo largo de los dos videos. Primero haremos el seguimiento en un video, y posteriormente en el otro.

El método elegido es hacer el seguimiento mediante el *'PointTracker'* que proporciona Matlab en la toolbox de visión. Se le pasan los puntos que debe seguir y se van buscando esos puntos frame a frame y representándolos en una secuencia de video.

Primero se muestran los puntos detectados, los centros de los iris.

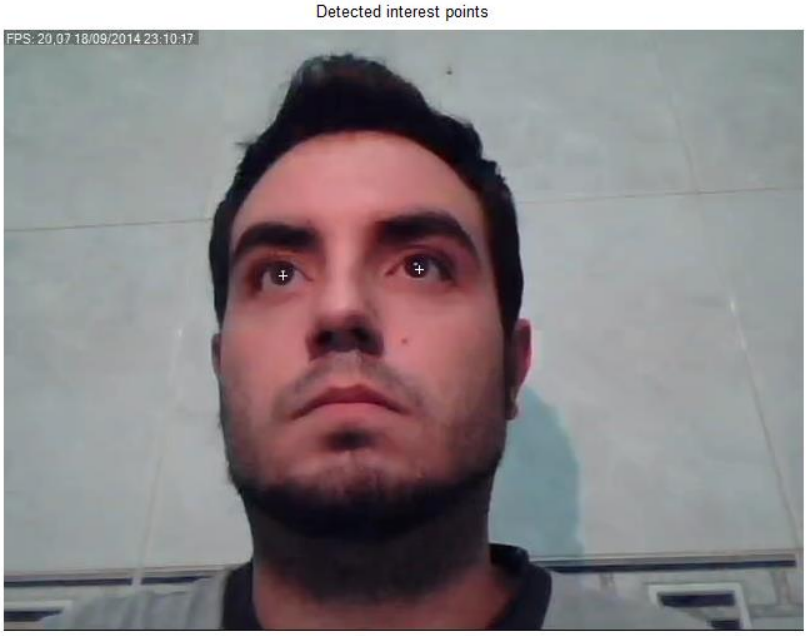


Figura 50: Puntos que identifican el centro del iris en la imagen 1.

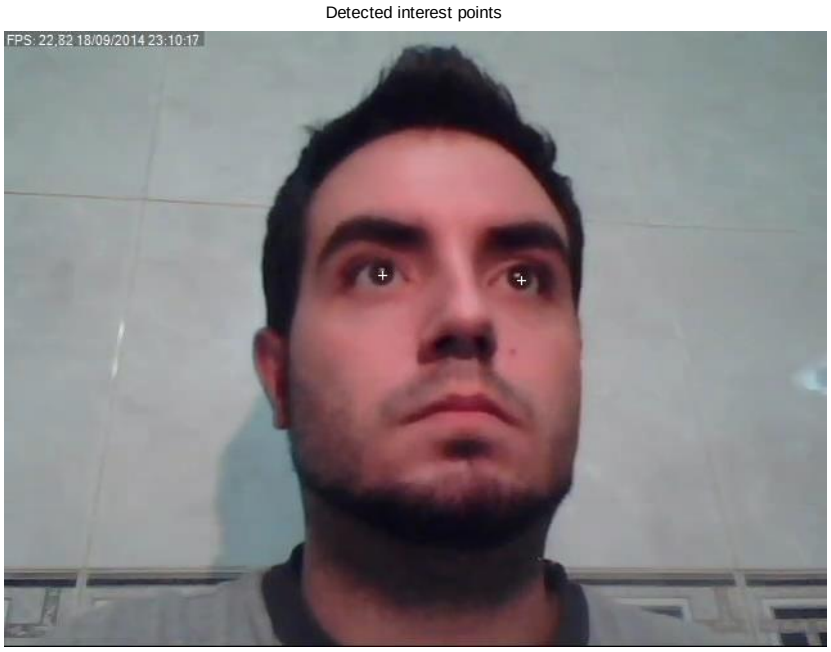


Figura 51: Puntos que identifican el centro del iris en la imagen 2.

A continuación se ha de crear e inicializar el seguidor, el tracker.

```
% tracker es el objeto seguidor, points1 son las coordenadas de los iris
% del primer video, objectFrame1 es la imagen del primer frame donde se
% han detectado los ojos.
tracker = vision.PointTracker('MaxBidirectionalError', 1);
initialize(tracker, points1, objectFrame1);
```

Al tracker se le pasa el máximo error bidireccional. Según la ayuda de Matlab, *'MaxBidirectionalError'* es el umbral de error *'forward-backward'*. El valor de este umbral es la distancia en pixeles desde la localización original de los puntos hasta la localización final después del *'backward tracking'*. Los puntos correspondientes son considerados inválidos cuando el error supera este umbral. Los valores recomendados son entre 0 y 3. El valor por defecto es infinito. Utilizar el error bidireccional es una vía efectiva de eliminar puntos que no se hayan seguido de manera fiable. La contrapartida es que calcular este error requiere computación adicional. Para nuestro caso es posible utilizarlo, por lo que probando diferentes valores, el que mejor resultado da es 1.

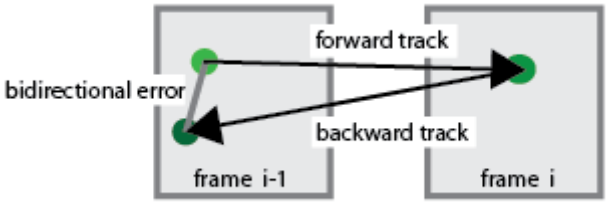


Figura 52: Error bidireccional.

Ahora se debe seguir y representar los puntos en cada frame de video, como siempre, para ambos videos.

Ya se ha comentado que no se controlan los pestañeos de los usuarios, pero si el movimiento del iris es demasiado rápido, el algoritmo se pierde. Por ello ha habido que controlar que si algún punto se pierde, se hace de nuevo la detección de los ojos y de los iris para volver a extraer las coordenadas y así continuar con el seguimiento.

Cada vez que se sigue un punto, éste se almacena en una variable para posteriormente poder extraer el eye gaze de cada instante. El siguiente es el código de seguimiento de la mirada para uno de los videos.

```
while ~isDone(videoFileReader1)

    frame = step(videoFileReader1);             % lee el siguiente frame
    [points1, validity] = step(tracker, frame);  % sigue los puntos
    visiblePoints1=points1(validity,:); % cuantos puntos son visibles
      % variable donde se guardan las   coordenadas de los puntos de
      ambos ojos
    totalpoints1(h,:)=points1(1,:);
    totalpoints1(h+1,:)=points1(2,:);

% comprobamos si se ha perdido algún punto
    if size(visiblePoints1,1)==2
        out = step(markerInserter, frame, points1(validity, :));
% mostramos los puntos

        step(videoPlayer, out);
    else
% En el caso de que hayamos perdido algún punto, debemos detectar de
nuevo los ojos, reincializar el tracker y continuar con el seguimiento
        points1=DetectIris(frame,false);
        tracker = vision.PointTracker('MaxBidirectionalError', 1);
        initialize(tracker, points1, frame);
        validity=logical([1;1]);
```

```
        out = step(markerInserter, frame, points1(validity, :));
        step(videoPlayer, out);
    end
   h=h+2;
end
```

Video de demostración (cuando se ralentiza el video es porque el algoritmo se ha perdido y está volviendo a ejecutar la detección de ojos y de iris):

https://www.dropbox.com/s/e42oy72u72uipj2/eye-tracking1.avi?dl=0

También es importante controlar, en el caso de hacer el eye-tracking en modo calibración, cuales son los instantes de cada video en los que el usuario está mirando a cada punto del video de calibración. Para ello hay que estimar en que momento el video de calibración se muestra al usuario. Haciendo muchas pruebas se logra este objetivo, para comprobarlo se capturan imágenes en los momentos que se estima que el usuario está mirando a un determinado lugar. El código mostrado a continuación es el que se utiliza para guardar y mostrar los frames que corresponden a cada dirección de la mirada del primer video.

```
Init=121;
if(h==Init)%Mirada al centro
    C1=frame;
end
if(h==Init+40)%Mirada al Noroeste
    NO1=frame;
end
if(h==Init+80)%Mirada al Noreste
    NE1=frame;
end
if(h==Init+120)%Mirada al Sudoeste
    SO1=frame;
end
if(h==Init+160)%Mirada al Sudeste
    SE1=frame;
End
```

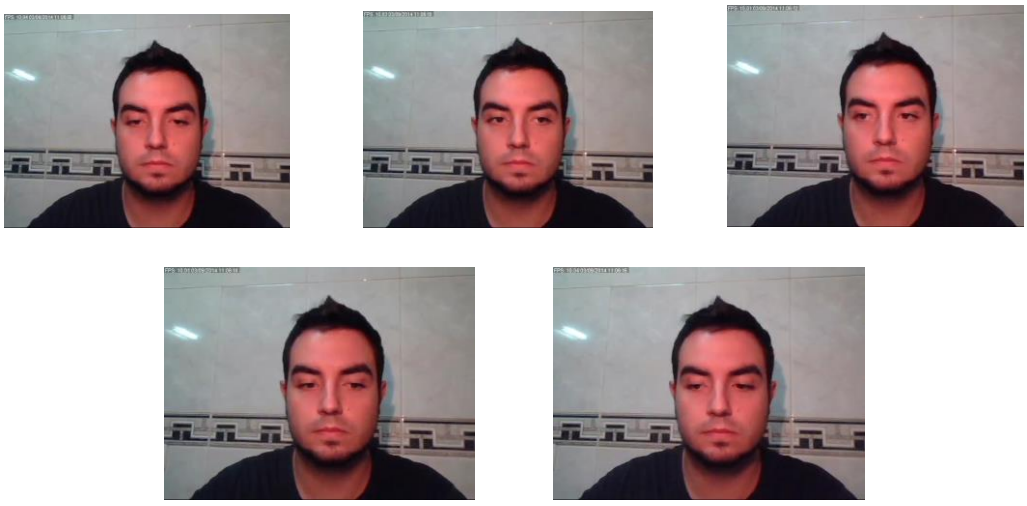


Figura 53: Secuencia de las miradas del primer video.

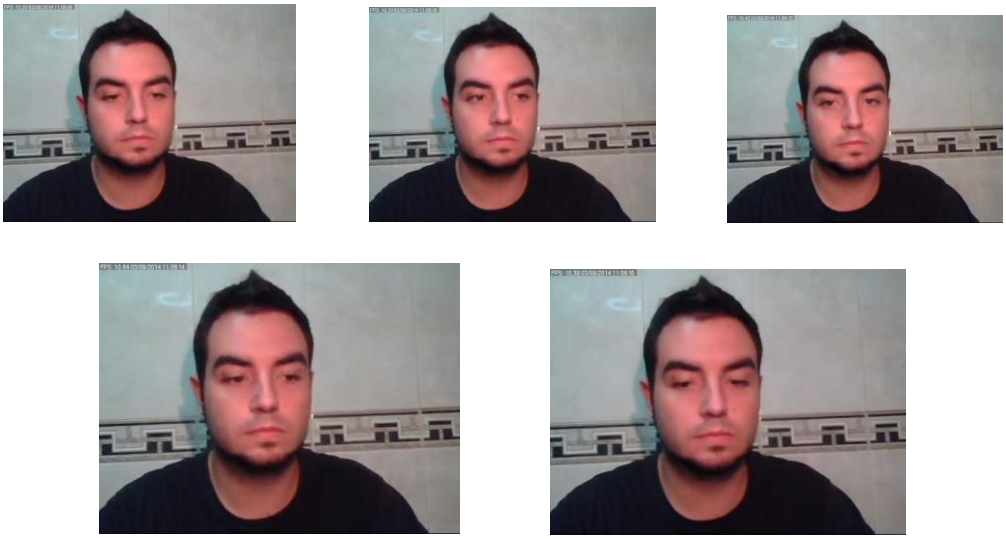


Figura 54: Secuencia de las miradas del segundo video.

4.5 Eye gaze

Para poder estimar el eye-gaze, primero se ha tenido que calibrar el programa. La calibración extrae los puntos extremos a los que puede mirar el usuario, es decir, centro, arriba a la izquierda, arriba a la derecha, abajo a la izquierda y abajo a la derecha.

Actualmente se tienen las coordenadas de todos los puntos estimados en los dos videos y se pueden identificar cuáles de esas coordenadas se corresponden a las 5 posibles posiciones de mirada que se van a estimar.

Se decide estimar sólo 5 posibles posiciones porque se quiere hacer un seguimiento de la mirada sencillo como demostración para trabajos futuros.

Ahora se extraen las coordenadas de cada ojo para cada video en las 5 posiciones extremas. La notación es C=centro, NO=noroeste, NE=noreste, SO=suroeste, SE=sureste, i=ojo izquierdo, d=ojo derecho, 1=video 1, 2=video 2. Se recuerda también que 'init' es la posición inicial que se estima para poder detectar las 5 posiciones extremas de los ojos teniendo en cuenta el tiempo que transcurre desde que se empiezan a grabar las webcams hasta que comienza el video de calibración.

```
Ci1=totalpoints1(init,:);
Cd1=totalpoints1(init+1,:);
Ci2=totalpoints2(init,:);
Cd2=totalpoints2(init+1,:);
NOi1=totalpoints1(init+40,:);
NOd1=totalpoints1(init+40+1,:);
NOi2=totalpoints2(init+40,:);
NOd2=totalpoints2(init+40+1,:);
NEi1=totalpoints1(init+80,:);
NEd1=totalpoints1(init+80+1,:);
```

```
NEi2=totalpoints2(init+80,:);
NEd2=totalpoints2(init+80+1,:);
SOi1=totalpoints1(init+120,:);
SOd1=totalpoints1(init+120+1,:);
SOi2=totalpoints2(init+120,:);
SOd2=totalpoints2(init+120+1,:);
SEi1=totalpoints1(init+160,:);
SEd1=totalpoints1(init+160+1,:);
SEi2=totalpoints2(init+160,:);
SEd2=totalpoints2(init+160+1,:);
```

En este momento es cuando hay que extraer unos descriptores para poder identificar hacia donde se está mirando. Se podrían usar herramientas de Matlab de la toolbox de 3 dimensiones, pero al alumno se le ocurrió un sencillo método que sirve para hacer una estimación.

Para explicar el método se parte de la suposición de que sólo se graba un ojo. Se tienen dos cámaras, cuando un usuario está mirando hacia un lateral, y poco después mira hacia el lateral opuesto, una de las cámaras habrá detectado más desplazamiento del punto central del iris que la otra. Esto es porque una cámara está más cerca del ojo que la otra. Si se está más cerca del ojo, se registrará un mayor desplazamiento, y si se está más lejos, se registrará menor desplazamiento. Utilizando este principio se intentan caracterizar las distancias entre una cámara y otra cuando se está mirando a un determinado punto para cada ojo. Y estas distancias serán las variables numéricas que caracterizarán la calibración del sistema.

Lo que se hace para cada posición extrema y para cada ojo es restar las coordenadas del video 1 a las del video 2. Obtendremos unos valores (x, y), negativos o positivos que caracterizan cuando ese determinado ojo está mirando a esa determinada posición.

```
DifCi=Ci1-Ci2;
DifCd=Cd1-Cd2;
DifNOi=NOi1-NOi2;
DifNOd=NOd1-NOd2;
DifNEi=NEi1-NEi2;
DifNEd=NEd1-NEd2;
DifSOi=SOi1-SOi2;
DifSOd=SOd1-SOd2;
DifSEi=SEi1-SEi2;
DifSEd=SEd1-SEd2;
```

A continuación se guardan en un fichero config.txt.

```
[stat,struc] = fileattrib;
PathCurrent = struc.Name;
NameFile = [PathCurrent '/config.txt'];
fileID = fopen(NameFile,'w');
fprintf(fileID,'%f\n',DifCi);
fprintf(fileID,'%f\n',DifCd);
fprintf(fileID,'%f\n',DifNOi);
fprintf(fileID,'%f\n',DifNOd);
fprintf(fileID,'%f\n',DifNEi);
fprintf(fileID,'%f\n',DifNEd);
fprintf(fileID,'%f\n',DifSOi);
fprintf(fileID,'%f\n',DifSOd);
fprintf(fileID,'%f\n',DifSEi);
```

```
fprintf(fileID,'%f\n',DifSEd);
fclose(fileID);
Config='config.txt';
```

Este será el fichero de calibración.

Ahora viene la parte en la que se estima hacia donde está mirando el usuario en cada momento. Lo primero que se hace es cargar el fichero de configuración.

```
[ Ci, Cd, NOi,NOd,NEi,NEd,SOi,SOd,SEi,SEd ] = loadConfig('config.txt');
```

El siguiente paso es identificar el punto al que se está mirando (eye-gaze). La metodología a seguir es obtener para cada frame de video los puntos a los que se ha estimado que se está mirando. Restar los puntos de la cámara 1 a los de la cámara 2, como se hace al guardar la configuración de calibración, y buscar entre las 5 posiciones cual es la que más se acerca para decidir a cual de las 5 posiciones está mirando el usuario.

Se crean cuatro vectores: *VectXi* es un vector con los cinco descriptores de la coordenada *x* del ojo izquierdo, *VectXd* es un vector con los cinco descriptores de la coordenada *x* del ojo derecho, *VectYi* y *VectYd* son vectores con los cinco descriptores de las coordenadas *y* del ojo izquierdo y del ojo derecho respectivamente.

```
VectXi=[Ci(2) NOi(2) NEi(2) SOi(2) SEi(2)];
VectXd=[Cd(2) NOd(2) NEd(2) SOd(2) SEd(2)];
VectYi=[Ci(1) NOi(2) NEi(1) SOi(1) SEi(1)];
VectYd=[Cd(2) NOi(2) NEd(1) SOd(1) SEd(1)];
```

A continuación se deben recorrer los frames del video, en el caso del eye tracking del video de calibración, serán sólo los frames correspondientes al video de calibración (120 frames). Para cada frame habrá que extraer el descriptor para cada ojo y coordenada.

```
for j=first_frame:numFrames

    Difiy=totalpoints1(h,1)-totalpoints2(h,1);
    Difix=totalpoints1(h,2)-totalpoints2(h,2);
    Difdy=totalpoints1(h+1,1)-totalpoints2(h+1,1);
    Difdx=totalpoints1(h+1,2)-totalpoints2(h+1,2);
    h=h+2;
```

Posteriormente habrá que encontrar, para cada coordenada y ojo, el valor de las 5 posiciones al que más se acercan los descriptores del punto actual. Para ello, en Matlab hay que restar cada vector al descriptor, hacer el valor absoluto de la diferencia y encontrar el mínimo. Cuanto más próxima sea esta diferencia a cero, más fiable será la decisión.

```
tempXi=abs(VectXi-Difix);
    tempXd=abs(VectXd-Difdx);
    tempYi=abs(VectYi-Difiy);
    tempYd=abs(VectYd-Difdy);

    [minxi minxi]=min(tempXi)
    [minxd minxd]=min(tempXd)
    [minyi minyi]=min(tempYi)
    [minyd minyd]=min(tempYd)
```

Ahora se tienen con 4 valores *'minxi, minxd, minyi, minyd'*, que variarán de 1 a 5. Siendo 1 que se detecta que se ha mirado al centro, 2 noroeste, 3 noreste, 4 suroeste y

sureste. (por la posición en la que se han guardado los descriptores en los vectores). Si los 4 valores coincidieran, la detección sería perfecta. Lo normal es que alguno de los valores difiera del resto, porque el algoritmo no es muy preciso y el tracking puede estar mal hecho. Para decidir a donde está mirando el usuario se elige lo que decida la mayoría, o en el caso de que cada uno diga a un lugar diferente, se elige donde señala 'minxi'. Para hacer esto en Matlab se utiliza el operador moda.

```
ModeT=mode([minxi minxd minyi minyd]);
      if (ModeT==1) %Mirada al centro
        pix_pos(j,1)=500;
        pix_pos(j,2)=280;
    elseif(ModeT==2) %Mirada al noroeste
        pix_pos(j,1)=20;
        pix_pos(j,2)=20;

    elseif(ModeT==3) %Mirada al noreste
        pix_pos(j,1)=1000;
        pix_pos(j,2)=20;

    elseif(ModeT==4) %Mirada al suroeste
        pix_pos(j,1)=20;
        pix_pos(j,2)=560;

    elseif(ModeT==5) %Mirada al sureste
        pix_pos(j,1)=1000;
        pix_pos(j,2)=560;
    else %Mirada al centro

        pix_pos(j,1)=500;
        pix_pos(j,2)=400;
    end
```

Por último se debe representar el resultado en un video. En el caso de la calibración, se compara el video de calibración con la estimación del eye-gaze calculado. Para hacerlo, al igual que al crear el video de calibración, se utiliza la función 'patch' de Matlab.

Video de ejemplo del resultado del eye-gaze tracking. (El punto verde es donde se estima que se está mirando).

https://www.dropbox.com/s/nh6ze3a4e0w5td4/eye-gaze1.avi?dl=0

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

Este proyecto fue creado con el objeto de proporcionar las herramientas suficientes para hacer la implementación de un seguidor de la mirada que detecte el punto al que está mirando el usuario de manera precisa y robusta mediante técnicas estéreo. A lo largo del trabajo fin de grado se han mencionado muchas herramientas que se pueden utilizar para tal fin. En el apartado de líneas futuras se resumirán los posibles caminos que se pueden seguir para mejorar el seguidor de la mirada implementado por el alumno.

Para evaluar los resultados del proyecto, se debe investigar si se han cumplido los objetivos parciales propuestos al inicio de la memoria.

5.1.1 Objetivo parcial 1

Localizar y evaluar las herramientas suficientes para hacer una implementación completa de un seguidor de la mirada, en inglés *'eye-tracker'*.

Partiendo del hardware propuesto, dos webcams conectadas a un portátil con sistema operativo Windows 7 y software de programación Matlab, se han identificado y

clasificado las herramientas necesarias en distintos pasos y en el capítulo 3 se han expuesto las ventajas y desventajas de cada una de ellas. A continuación se resumen las herramientas propuestas para cada necesidad:

- Grabación de las dos webcams de manera simultánea. Posible desde el propio Matlab y desde programas externos como iSpy.
- Calibración del montaje. Técnicas de calibración estéreo de las toolboxes de Matlab o montaje fijo.
- Interfaz gráfica de Matlab.
- Detección de la cara y de los ojos. Mediante las toolboxes que proporciona Matlab que trabajan con el algoritmo de Viola & Jones.
- Detección de los iris del usuario. Se pueden detectar mediante programas que funcionan con transformada de Hough Circular, o que funcionan con el operador integrodiferencial de Daugman.
- Tracking de los ojos. Se pueden utilizar programas y utilidades de Opencv o la toolbox de tracking que proporciona Matlab.
- Extraer el eye-gaze. Es posible programar sencillos algoritmos de cálculo de distancias para visión estéreo, o se puede utilizar la toolbox de 3D de Matlab.

Se han conseguido dar herramientas y soluciones para todos los apartados, por lo que este objetivo se da por conseguido.

5.1.2 Objetivo parcial 2

Conseguir implementar un eye-tracker básico que partiendo de unas limitaciones, haga un seguimiento de los ojos aceptable.

A lo largo del capítulo 4 se ha expuesto una solución implementada por el alumno que realiza eye-tracking de manera bastante fiable partiendo de las restricciones que se habían predefinido. La manera de medir cuán preciso es el eye-tracker, tiene que ser objetivamente, observando varias ejecuciones del programa para diferentes videos.

A continuación se incluyen links de varias simulaciones del programa:
https://www.dropbox.com/s/e42oy72u72uipj2/eye-tracking1.avi?dl=0
https://www.dropbox.com/s/0ku44suo3zdqfe3/eye-tracking2.avi?dl=0
https://www.dropbox.com/s/1czxbwcxnxlasxp/eye-tracking3.avi?dl=0
Como podemos observar, en la mayor parte de los videos el tracking se hace de manera bastante efectiva, aunque una vez que el algoritmo se pierde y detecta los ojos en otro lugar, ya no vuelve a reengancharse. Este problema se puede solucionar en el futuro mediante comprobaciones regulares de la posición de los ojos.

Se concluye que el objetivo está cumplido, ya que con ciertas restricciones se consigue hacer un seguimiento de los ojos bastante preciso.

5.1.3 Objetivo parcial 3

Plantear una solución para estimar el punto al que el usuario está mirando en la pantalla mediante técnicas estéreo. Método eye-gaze.

En el apartado 3.9 se plantean diferentes maneras de extraer el punto al que el usuario está mirando, pero no sólo eso, en el apartado 4.4 se implementa un sencillo algoritmo que permite hacer una estimación de dicho punto. La estimación es poco precisa, ya que sólo se pueden detectar 5 posibles posiciones a las que esté mirando un usuario. Pero no se ha profundizado más en realizar una solución más precisa ya que no era objetivo de este proyecto. En estudios futuros esta parte se debe mejorar.

De todas maneras se han creado clips de video con los resultados de este apartado para poder estimar qué porcentaje de tiempo se acierta en la estimación.

Los videos de resultados son los siguientes:
Video 1:https://www.dropbox.com/s/nh6ze3a4e0w5td4/eye-gaze1.avi?dl=0
Video 2:https://www.dropbox.com/s/whr98gb4gh03vap/eye-gaze2.avi?dl=0
Video 3:https://www.dropbox.com/s/70nsvb67pce1k18/eye-gaze3.avi?dl=0
Para calcular los porcentajes, se calcula el tiempo durante el cual el algoritmo acierta frente al que falla.
Video 1: 12.3 segundos de acierto frente a 18 segundos de video. 68.3% de acierto.
Video 2: 11.9 segundos de acierto frente a 18 segundos de video. 66.1% de acierto.
Video 3: 11.8segundos de acierto frente a 18 segundos de video. 65.5% de acierto.

Los resultados, aunque no son muy favorables, si son prometedores, ya que se consigue acertar en bastantes ocasiones. Consideramos, por tanto, cumplido el objetivo inicial planteado.

5.1.4 Presupuesto

Costes directos del personal				
Apellidos y Nombre	Puesto	Dedicación (horas)	Precio Unitario (€/hora)	Total (€)
Alba de Viana-Cárdenas, Javier	Ingeniero Alumno	300	10	3.000
Peláez Moreno, Carmen	Ingeniero Tutor	50	30	1.500
			Total	4.500

Tabla 2: Presupuesto, costes directos del personal

El coste imputable se calcula mediante la amortización de la siguiente manera:
Coste imputable= (Dedicación/Depreciación) * Coste* Uso

Costes de software y hardware					
Producto	Coste	Uso (%)	Dedicación (meses)	Depreciación (meses)	Coste imputable(€)
Ordenador Toshiba Satellite	700	100	8	36	155.55
Software Matlab	69	100			69
2 Webcams	15	100	8	12	10
				Total	234.55

Tabla 3: Presupuesto, costes de software y hardware

Costes Totales	€
Costes directos del personal	4.500
Coste imputable de software y hardware	234.55
Total sin I.V.A	4.734,55
I.V.A (18%)	852.21
Total	5.586.76

Tabla 4: Presupuesto total

"El presupuesto total de este proyecto asciende a la cantidad de CINCO MIL QUINIENTOS OCHENTA Y SEIS EUROS CON SETENTA Y SEIS CÉNTIMOS DE EURO."

5.2 Líneas futuras

Se puede seguir trabajando en este proyecto por diferentes vías. Una posibilidad sería buscar nuevas herramientas para cualquiera de las fases de construcción propuestas para programar un eye-tracker. Según avance la tecnología, nuevas herramientas y posibilidades surgirán para hacer un buen seguimiento de la mirada con visión estéreo.

La parte que más se puede explotar es la visión estéreo, las técnicas en tres dimensiones. Matlab dispone de muchas herramientas que combinadas pueden servir no sólo para calibrar las cámaras y extraer el punto de observación, como se ha visto en esta memoria, sino también para cualquier otra parte, como el eye-tracking. Se podrían investigar métodos que sirvan para que el eye-tracker funcionara de principio a fin con técnicas 3D. Esto sería para no tener que hacer la detección de los ojos, de los iris y el tracking para cada una de las dos cámaras, es decir, tratar de reducir gasto computacional al no trabajar por duplicado.

A partir de las herramientas propuestas se sugieren las siguientes líneas de acción:

- Se podría intentar realizar la captura de las grabaciones desde el propio Matlab para intentar hacer el eye tracking en tiempo real. Una posibilidad sería encontrar un algoritmo de detección de los ojos muy rápido, o utilizar el

mismo algoritmo utilizado para el ejemplo de esta memoria con una computadora más potente.

- Cambiar la metodología de la detección, es decir, se podría capturar unos ciertos fotogramas al principio del video y estimar la posición de los ojos, y después comenzar la grabación de las dos webcams y únicamente hacer el seguimiento de los ojos. Así se reduciría considerablemente la carga computacional y se podría intentar hacer un eye-tracker en tiempo real.

- Otro punto a mejorar sería controlar el movimiento de la cabeza para que no afecte al seguimiento de los ojos, y así hacer un algoritmo robusto frente a los movimientos del usuario. Esto se podría hacer mediante la detección de la cara con el algoritmo de Viola & Jones que proporciona el detector de Matlab 'CascadeObjectDetector'. Después se haría tracking de la cara, posible con el 'Point Tracker' de Matlab, y por último se contrarrestaría el movimiento de la cabeza con el de los ojos.

- Controlar los pestañeos debería ser otra prioridad, en el programa implementado, los pestañeos hacen que el programa se pierda durante todo el tiempo que dura el pestañeo, y a veces no se recupera. Se podría controlar haciendo comprobaciones de la posición de los ojos cada cierto tiempo o implementando un detector que identifique los pestañeos.

- Sería muy interesante investigar metodologías de detección de los ojos y del iris que sean robustas frente a los cambios de iluminación. Un posible punto de partida sería la investigación de algoritmos basados en Autómatas de Aprendizaje. El problema es la grabación en baja definición que efectúan las webcams que limita las posibilidades a la hora de utilizar algoritmos para la detección.

Capítulo 6

Conclusion and further works

The purpose of this project was to provide a variety of tools to implement an eye-tracker able to detect the eye-gaze as accurate and robust as possible by using stereo techniques. Along the report of the project several tools are mentioned, including their advantages and disadvantages. In the further work section ways to improve the eye-tracker implemented by student are outlined.

To evaluate the project results, the accomplishment of the partial objectives stated at the beginning of the report should be revisited.

6.1 Conclusions

6.1.1 Partial objective 1

Find and select a variety of software tools to implement an eye-tracker.

Taking into account the constraints of the proposed hardware (two web cams connected to a computer with Windows 7 operative system and Matlab programming software) necessary tools have been identified. Chapter 3 explains their advantages and disadvantages. The following list summarizes the chosen software for each module.

- Recording from two webcams simultaneously: Matlab and third-party programs like iSpy.
- Camera calibration: there are tools for stereo calibration in Matlab toolboxes but we opted for the immobilization of the two cameras in a fixed setting.
- Matlab Graphical Interface development facilities.
- Face and eyes detection: the well-known Viola & Jones as implemented in Matlab's vision toolbox.
- Iris detection: Hough circular transform or based on Daugman's integro-differential operator.
- Tracking: both Matlab and Opencv offer utilities.
- Eye-gaze tracking: the 3D Matlab toolbox offers a set of tools for locating points in an image but also simple ad-hoc programs can be employed.

Given the previous enumeration, we can consider the objective achieved.

6.1.2 Partial objective 2

Implementation of a simple eye-tracker given a set of constraints.

The achieved implementation is described in chapter 4. Its reliability is limited by the choice of hardware. A formal evaluation should include a large number of videos and subjects but in this project only an informal evaluation has been carried out.

Links to examples of its functionality and performance are includes in the following links:

https://www.dropbox.com/s/e42oy72u72uipj2/eye-tracking1.avi?dl=0
https://www.dropbox.com/s/0ku44suo3zdqfe3/eye-tracking2.avi?dl=0
https://www.dropbox.com/s/1czxbwcxnxlasxp/eye-tracking3.avi?dl=0

It can be observed how, in the majority of the videos, the tracking is effective, though if the algorithm gets lost and erroneously detects the eyes in wrong places, it does not have the capability of recovering from it. This problem can be solved in the future by regular checkings of eyes positions.

We must conclude then that the goal has been accomplished.

6.1.3 Partial objective 3

A solution proposal for the estimation of the point at which a user is looking in a screen (i.e. the eye gaze) by means of stereo techniques.

In section 3.9 several ways to extract the point which users is looking at are reviewed. A simple algorithm devised ad-hoc for this task is described in section 4.4. The estimation so obtained is limited to 5 possible positions at which the user is looking. Since accuracy was not one of the goals of this project this must be left for further work.

A video clip for visual inspection of the results has been included in the following link:

Video 1:https://www.dropbox.com/s/nh6ze3a4e0w5td4/eye-gaze1.avi?dl=0
Video 2:https://www.dropbox.com/s/whr98gb4gh03vap/eye-gaze2.avi?dl=0
Video 3:https://www.dropbox.com/s/70nsvb67pce1k18/eye-gaze3.avi?dl=0

Video 1: 12.3 success seconds versus 18 seconds of video. 68.3%.
Video 2: 11.9 success seconds versus 18 seconds of video. 66.1%.
Video 3: 11.8 success seconds versus 18 seconds of video. 65.5%.

In spite of its limitations the results are considered promising since for long periods of time the target is hit. Therefore we can conclude that the goal is accomplished.

6.2 Further Work

Work on this project could be continued in several different ways. First, the continuous review of new available tools should be of use since technology is progressing in this field very fast. Therefore, new possibilities and alternatives could emerge.

Second, the least developed part of the project is the exploration of 3D techniques in stereo vision. Matlab already offers a number of 3D based tools that could be combined to calibrate the cameras, extract the eye-gaze (this has been studied in this report) and others functions as eye tracking. The employment of 3D techniques from the start for all the modules (face, eye and iris detection and tracking) could reduce the computational cost of computing them separately for each of the eyes.

Departing from the proposed tools suggests the following lines of action:

- The recording of the video sequences could be integrated in Matlab to try to do the eye-tracking in real time. A possibility would be to find an algorithm of fast eye detection, or use of more powerful computing device.

- Perform an initialization phase with a small number of initial frames for the detection of the eyes position. Then, with the two cameras recordings only eye tracking should be necessary. This would reduce the computational cost and ease the implementation of a real-time eye-tracker.

- Another critical issue is the compensation of head position changes to produce a more robust eye tracking. This should be possible by using face detection routines based on Viola & Jones algorithm.

- Another priority should be to control the effects of blinks. In the implemented program blinks produce irrecoverable tracking loses. Proposals for solving this problem include the implementation of periodic checks of eyes positions or the inclusion of a detector that identifies blinks.

- Techniques for illumination robustness able to cope with the low resolution recordings from the cameras employed in the project.

Referencias

[1] Duchowski, Andrew T. '*Eye tracking methodology: theory and practice*' (Springer, 2007, Second Edition).

[2] Bergstrom, Jennifer Romano; Schall, Andrew Jonathan. '*Eye tracking in user experience design'* 2014. Libro electrónico: http://proquest.safaribooksonline.com.strauss.uc3m.es:8080/book/web-development/usability/9780124081383.

[3] González, R; Woods R; and Eddins S '*Digital image processing using MATLAB'.* (Pearson Education India, 2004).

[4] Tobii Technologies. http://www.tobii.com/ Accedido en agosto del 2014.

[5] Allied vision technologies. http://www.alliedvisiontec.com/us/products/application-markets/application-case-studies/application-case-study/article/an-insight-into-eye-tracking-applications.html Accedido en septiembre del 2014.

[6] Duchowski, Andrew T. *'Eye-Based Interaction in Graphical Systems: Theory & Practice, Part III Potential Gaze-Contingent Applications'* Articulo en linea. http://www.ergoestudio.com/descargas/gaze_applications.pdf.

[7] Elias Daniel Guestrin *'General Theory of Remote Gaze Estimation Using the Pupil Center and Corneal Reflections'* Articulo con licencia de IEEEXplore de la universidad Carlos III de madrid. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1634506 .

[8] Especificaciones técnicas de las dos Webcams. http://logitech-en-amr.custhelp.com/app/answers/detail/a_id/18251/kw/c160%20technical%20specifications Accedido en septiembre del 2014.

[9] Página principal de Mathworks: http://www.mathworks.es/.

[10] Página principal de iSpy: http://www.ispyconnect.com/ .

[11] Triangulo de oro: http://www.mediative.com/research/golden-triangle Accedido en septiembre del 2014.

[12] Javier San Agustín Lopez, *'Off-the-Shelf Gaze Interaction'* Tesis de septiembre del 2009, Universidad de copenague.: https://www.itu.dk/en/Research/PhD-Programme/PhD-Defences/~/media/A2CE39A9F78C42368648E8C9D6F56ACE.ashx .

[13] Página principal de la empresa MyGaze. http://www.mygaze.com/ .

[14] Foro sobre 3D en el cual se habla del software onuprova: http://3dvision-blog.com/6551-onuprova-3d-camera-helps-you-record-in-3d-from-two-webcams/.

[15] Página de descarga de AMCap. http://amcap.en.softonic.com/

[16] Fitts, P. M.; Jones, R.E.; Milton, J.L. (1950). '*Eye movements of aircraft pilots during instrument-landing approaches*'. En: Aeronautical Engineering Review, 9 (2), pp. 24-29. http://books.google.es/books?hl=es&lr=&id=WthALsrg_f4C&oi=fnd&pg=PA56&dq=Aeronautical+Engineering+Review+Eye+movements+of+aircraft+pilots+during+instrument-landing+approaches&ots=Omg7CFjlXB&sig=Cv6d8BTKUe1GOh9Dx96QkddUt5g#v=onepage&q=Aeronautical%20Engineering%20Review%20Eye%20movements%20of%20aircraft%20pilots%20during%20instrument-landing%20approaches&f=false .

[17] Hassan Montero, Yusef; Herrero Solana, Víctor. *'Eye-Tracking en Interacción Persona-Ordenador'* Articulo del 28 de Octubre de 2007. http://www.nosolousabilidad.com/articulos/eye-tracking.htm#fitts .

[18] Como grabar un video desde webcam en matlab. http://www.mathworks.com/matlabcentral/fileexchange/46149-record-video-from-webcam-using-matlab/content/recording_video.m Accedido en junio del 2014.

[19] Copyright (c) 2010, Wim Abbeloos, *'Stereo Matching'*. Programa en Matlab. http://www.mathworks.com/matlabcentral/fileexchange/28522-stereo-matching .

[20] Toolbox de Matlab *'camera calibration and 3D vision'*. http://www.mathworks.es/es/help/vision/stereo-vision.html. Último acceso septiembre del 2014.

[21] Díaz Miranda, Tonathiuh; Viloria Rodríguez, José Luis. *'Algoritmo de Viola-Jones para detección de rostros en procesadores gráficos'* Articulo del 2011 publicado en la Revista estudiantil nacional de ingeniería y arquitectura. http://renia.cujae.edu.cu/index.php/revistacientifica/article/viewFile/151/pdf_56 .

[22] Viola, Paul; Jones, Michael J. *'Robust Real-Time Face Detection'*. 11 de Julio del 2003. http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf

[23] Programa en Matlab '*Face Parts Detection'* por Masayuki Tanaka. 25 de mayo del 2012. Proporcionado por MathWorks. http://www.mathworks.com/matlabcentral/fileexchange/36855-face-parts-detection .

[24] Cuevas, Erik; Wario, Fernando; Zaldivar, Daniel; Pérez-Cisneros, Marco. *'Detección de círculos usando Autómatas de aprendizaje'*. Artículo de la Universidad de Guadalajara (México). http://posec.cucei.udg.mx/Tesis/SOMI_modificado.pdf .

[25] Función de Matlab de la transformada de Hough. http://www.mathworks.es/es/help/images/ref/imfindcircles.html. Último acceso septiembre del 2014.

[26] Programa en Matlab *'Iris segmentation using Daugman's integrodifferential operator'* por Anirudh Sivaraman. 6 de julio del 2011. Proporcionado por MathWorks. http://www.mathworks.com/matlabcentral/fileexchange/15652-iris-segmentation-using-daugman-s-integrodifferential-operator .

[27] Luis Mottali, Marcelo. *'Implementación de un sistema de identificación de personas en tiempo real por reconocimiento de iris'*. Tesis final de la Universidad de Buenos Aires. http://www.dc.uba.ar/inv/tesis/licenciatura/2008/mottalli .

[28] Daugman, John G. *'High confidence visual recognition of persons by a test of statistical independence'*, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (1993), no. 11, pp 1148–1161. http://www.cl.cam.ac.uk/~jgd1000/PAMI93.pdf .

[29] Timm, Fabian; Barth, Erhardt. *'Accurate eye centre localization by means of gradients'* University of Lübeck. http://cjee.lakeheadu.ca/public/journals/22/TiBa11b.pdf .

[30] Página principal de Opencv. http://opencv.org/ .

[31] Unión de Matlab y Opencv. Proporcionado por MathWorks http://www.mathworks.es/discovery/matlab-opencv.html .

[32] Programa en Opencv de tracking de los ojos *'EyeLike'*. http://thume.ca/projects/2012/11/04/simple-accurate-eye-center-tracking-in-opencv/

[33] Programa en Matlab que permite hacer tracking de la cara de un usuario mediante tracking de histogramas.
http://www.mathworks.es/es/help/vision/examples/face-detection-and-tracking-using-camshift.html

[34] Toolbox de Matlab *'Object Tracking and Motion Estimation'*.
http://www.mathworks.es/products/computer-vision/features.html#object-tracking-and-motion-estimation .