UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

Diseño e implementación de una ayuda electrónica para pacientes con baja visión periférica (I)

Autor: Jonathan Pajares Redondo
Titulación: Grado en Ingeniería Electrónica Industrial y Automática
Profesor Tutor: Juan Carlos Torres Zafra (Departamento de Tecnología Electrónica)
Codirector: Ricardo Vergaz Benito (Departamento de Tecnología Electrónica)

Título: Diseño e implementación de una ayuda electrónica para pacientes con baja visión periférica (I)

Autor: Jonathan Pajares Redondo
Tutor: Juan Carlos Torres Zafra
Codirector: Ricardo Vergaz Benito

EL TRIBUNAL

Presidente: Virginia Urruchi del Pozo
Vocal: Plinio Jesús Pinzón Castillo
Secretario: Jonathan Crespo Herrero

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 8 de Julio de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO **PRESIDENTE**

Índice

Índice de figuras

Agradecimientos

Es muy difícil acordase, en el momento de escribir estas líneas, de todas las personas que han sido una pieza clave, no solo durante este último año, si no durante todo el desarrollo de mi etapa universitaria.

Primero, tengo que dar las gracias sin lugar a dudas a mi madre y mi hermano. Sin ninguna duda son las dos personas más importantes de mi vida y posiblemente de las que más he aprendido, de mi madre el coraje y el tener la fuerza y la voluntad de sobreponerse a cualquier situación, por muy negra que se ponga la vida siempre hay una solución. De mi hermano es más difícil decir que he aprendido y que no, pues creo que gran parte de lo que soy a día de hoy se lo debo a él, su forma de ver el mundo me ha dado siempre una perspectiva única e irremplazable. A lo largo de este año me han apoyado muchísimo con este trabajo y han entendido perfectamente los días que llegaba a las tantas de la universidad, y que me quedaba toda la noche encerrado en mi habitación. Viviendo en la misma casa y había días que ni les veía, pero siempre están ahí para echarme una mano cuando la necesito.

Tengo que dedicarle unas líneas a “mi burrito”, durante estos 4 años ha estado siempre hay, cuando más lo he necesitado y cuando estaba cansado. Durante estos últimos meses debido a que no ha estado bien de salud no ha podido acompañarme en la recta final, pero sin ninguna duda gran parte se la debo a él.

Hay tres personas que han sido mi bálsamo durante este año, cada dia que salía mosqueado del laboratorio o tan cansado que solo quería dormir, me convencían de sentarnos en una terraza y arreglábamos el mundo con una jarra en la mano. Raquel, para la que yo no soy Joni, soy Ionis y siempre tiente una palabra de ánimo que darme, es la persona más positiva/negativa que conozco. Patry, nunca nadie en tan poco se había convertido en tanto, desde nuestras risas y nuestros llantos hasta sus monólogos sobre la comida, amor y lo que se tercie ese día, sin ninguna duda, una de las personas más importantes no solo este año, si no de los que vienen. David, en este caso un par de líneas creo que no es suficiente para agradecerle todo, no lo digo por sus apoyo durante el trabajo, sus ideas geniales que me sacaban de quicio porque me hacían rehacer el código o sus interminables correcciones de este documento, lo digo por esas horas interminables hablando, esos días que íbamos a estudiar y acabábamos solucionando nuestro futuro y filosofando por todo en vez de estudiar. Este año hemos vivido muchísimo y creo que eso nos ha unido de una manera que aún no somos conscientes. Te debo una dedicatoria en tu corcho, ahí va “Dicen que en la época universitaria todo el mundo hace un gran amigo que le marcara el resto de su vida, todo el mundo cree que esa amistad surge el primer o segundo año, yo he aprendido que cuando menos lo esperes y de la manera más estúpida aparecerá”.

Mis amigos de toda la vida, colegio, instituto, salsa… Aunque este año haya sido imposible cuadrar horarios, ya haya sido para tomar algo, salir a bailar o simplemente vernos 5 minutos, siempre cuando lo conseguíamos era como si el tiempo no pasara. En todo momento interesándose por mi trabajo, dándome ideas, apoyándome, pero sobre todo dando ese punto de alegría y buen rollo cuando más agobiado/estresado estaba.

No puedo olvidarme de todos mis amigos de la universidad, esas horas interminables en la biblioteca para sacar el laboratorio o trabajo de turno adelante, esos ratitos de risas en la cafetería o en el césped de relax. Sin ninguna duda estos años no habrían sido iguales sin ellos. En este apartado tengo que acordarme de mi compañero de proyecto, Ruben, sin él este trabajo no hubiera sido igual, y pasar tantísimas horas riendo, discutiendo, trabajando hacen que conozcas a una persona de verdad.

Tengo que agradecerle enormemente tanto a la gente del CMRF como a los médicos de IOBA, sus consejos y mejoras en este proyecto han sido importantísimas para mi y mi compañero, en concreto querria agradecérselo a Begoña Coco, ya que ha tenido que aguantar mil documentos y videos mios para terminar de perfilar el programa de diagnostico.

Es imposible acabar este apartado sin acordarme de mi Tutor y mi Codirector, Juan Carlos y Ricardo, a lo largo de este trabajo me han dado muchísima confianza para seguir trabajando y espero haberles devuelto esa confianza recibida y que podamos seguir trabajando en otros proyectos. Este agradecimiento se hace extendible a todos los miembros del GDAF, que desde el primer dia en el laboratorio tenían una palabra amable y te ayudaban siempre que era necesario.

Muchas gracias

“Somos quien somos gracias a la gente que nos rodea.”

Resumen

El principal objetivo del proyecto es crear un sistema de procesamiento de imágenes de un tamaño reducido, capaz de enviar la información de los contornos y las profundidades del entorno a personas con baja visión periférica a través de unas gafas de realidad virtual, creando para ello un sistema que satisfaga dichas necesidades con el menor coste posible.

Se van a separar el cálculo de distancias y el cálculo de bordes como dos bloques independientes, desarrollando así dos Trabajos Fin de Grado diferentes. En esta memoria en concreto se tratará el cálculo de los contornos incluyendo: El diseño de los algoritmos y su implementación en un sistema de bajo coste.

Por tanto, tras un diagnóstico realizado con un programa que se desarrolla en la siguiente memoria, se localizará la zona óptima o remanente de visión del paciente. Para ello no se seguirán métodos médicos de diagnóstico, sino que será el propio paciente el que elegirá esa zona.

En esa zona se enviará a través de las micropantallas de las gafas la imagen de los contornos, en el color que el paciente seleccione, del entorno que rodea al paciente en tiempo real. Esa imagen será capturada por una cámara situada en las gafas.

El sistema de procesamiento que se encargará de transformar las imágenes capturadas por la cámara y calcular los bordes de todos los objetos representados en dichas imágenes para después trasladarlas a las gafas, será de un tamaño reducido capaz de entrar en un bolsillo. En este proyecto se valorarán varias alternativas, buscando así la que mejores características tenga.

De esta manera se tendrá un sistema de bajo presupuesto, el cual podrá dar información del entorno a personas que debido a diversas afecciones han perdido parte de su visión periférica y por tanto han visto reducido parte de su visión, su resto visual.

Abstract

The main objective of the project consists of creating an small-sized image processing system. This system is intended to be able to send information about the edges and depths of the environment to people with low peripheral vision through virtual reality glasses, by creating a system that meets these needs at the lowest possible cost.

The calculation of the distance and the calculation of the edges have been divided into two different sections. As a result, two different End-of-Degree Projects have been developed. This report specifically addresses the calculation of the edges, including the design of the algorithms and their implementation on a low-cost system.

Therefore, the patient's optimal zone of vision or residual vision will be located after performing a diagnosis with a software which is developed in the following report. For this purpose, medical diagnostic methods should not be followed: the patient himself will choose this zone.

In this specific zone, the image of the edges which describe the patient's surroundings will be sent in real time. This image, whose color will be the patient's choice, will be sent through the microscreens of the glasses and will be captured by a camera located in the glasses.

The processing system will transform the images captured by the camera and calculate the edges of all the objects represented in these images. In addition, the system, which will be poket-sized, will transfer these images to the camera. Several alternatives will be assessed along this project in order to find the one with the best features.

This way, a low-budget system able to provide people with information about the environment and surroundings will be developed. The system focuses on people who suffer from different visual impairments and have lost part of their peripheral vision. For this reason, they have experienced a reduction on their residual vision.

Índice de acrónimos

ABS Acrilonitrilo Butadieno Estireno
API Application Programming Interface (Interfaz de programación de aplicaciones)
CMRF Centro de Recuperación de Personas con Discapacidad Física
CUDA Compute Unified Device Architecture (Arquitectura unificada de dispositivos de computo)
DSP Digital Signal Processor (Procesador digital de señales)
FPGA Field Programmable Gate Array
FPS Frames Per Second (Images por segundo)
GDAF Grupo de Displays y Aplicaciones Fotónicas
GPIO General Purpose Input/Output (Entradas/Salidas de propósito general)
GPU Graphics Processing Unit (Unidad gráfica de procesamiento)
HDMI High-Definition Multimedia Interface (Interfaz multimedia de alta definición)
IOBA Instituto Universitario de Oftalmobiologia Aplicada
TFG Trabajo Fin de Grado
UC3M Universidad Carlos III de Madrid
VGA Video Graphics Array (Adaptador gráfico de video)

Capítulo 1. Introducción y objetivos

1.1. Introducción

Antes de comenzar, es importante destacar que al haber sido este Trabajo Fin de Grado realizado en colaboración con [1], hay partes redactadas por ambos que guardan gran similitud. A lo largo del documento se irán indicando cómo se relacionan ambos trabajos para poder facilitar una visión global del proyecto.

Otro detalle que hay que remarcar es que este proyecto se ha realizado bajo el apoyo y la supervisión del Instituto de Oftalmobiologia Aplicada de la Universidad de Valladolid (IOBA). El IOBA es un instituto de la Universidad de Valladolid encargado de desarrollar tanto trabajo asistencial como participación en proyectos de investigación. Desde hace varios años existe un acuerdo de colaboración entre IOBA y el GDAF-UC3M.

Este proyecto se basa en los trabajos previos de Carlos Barranco [2] y Francisco Collado [3].

1.1.1. El Diseño para Todos

El Diseño para Todos es una filosofía que tiene como objetivo acercar al mayor número posible de gente cualquier producto o servicio. Este concepto se basa en los principios de la igualdad, la inclusión social y la diversidad humana entre otros.

Esta filosofía se enfocó inicialmente en el ámbito de la accesibilidad para personas con discapacidad, pero finalmente se ha extendido a toda la población independientemente de sus características o habilidades, tanto físicas como psíquicas.

Este diseño está definido a través de la Ley 51/2003 del 2 de diciembre de igualdad de oportunidades, no discriminación y accesibilidad universal de las personas con discapacidad como:

"La actividad por la que se concibe o proyecta, desde el origen, y siempre que ello sea posible, entornos, procesos, bienes, productos, servicios, objetos, instrumentos, dispositivos o herramientas, de tal forma que puedan ser utilizados por todas las personas, en la mayor extensión posible."

Dentro del marco Europeo, se han desarrollado varias iniciativas relacionadas con el Diseño para Todos.

"El Diseño Universal es un método efectivo para mejorar la accesibilidad y la calidad del entorno, servicios y productos. Se centra en la importancia de asegurar que el entorno, edificios y los productos cotidianos se diseñan para todos desde las primeras fases, sin necesidad de adaptarlos en fases posteriores. (...) Promover el principio del Diseño Universal, su aplicación y la participación del usuario en todas las fases del diseño es de vital importancia para mejorar la accesibilidad del entorno construido, el transporte, los sistemas de comunicación y la usabilidad de los productos." [4]

- **Principios del diseño para todos**

1. *Igualdad de uso: El diseño debe ser fácil de usar y adecuado para todas las personas.*
2. *Flexibilidad: El diseño debe adecuarse a un amplio rango de preferencias.*
3. *Simple e intuitivo: El diseño debe ser simple y fácil de entender.*
4. *Información fácil de percibir: El diseño debe ser capaz de transmitir información sin tener en cuenta la capacidad de percepción del usuario.*
5. *Tolerante a errores: El diseño debe evitar las acciones accidentales que puedan provocar fallos fatales.*
6. *Escaso esfuerzo físico: El diseño no ha de requerir un gran esfuerzo físico para su uso*
7. *Dimensiones apropiadas: El diseño debe tener las dimensiones apropiadas para el manejo de cualquier usuario.*

El siguiente trabajo se realiza completamente de acuerdo a esta filosofía de diseño, intentando en cada fase del proyecto estar en contacto con los potenciales usuarios y expertos oftalmólogos, para de esta manera evitar desagradables sorpresas en la implementación final del producto en el mercado.

1.1.2. Definición de Baja Visión.

La baja visión es un concepto muy amplio el cual engloba cualquier privación parcial de la vista que no puede ser corregida adecuadamente con gafas, lentes de contacto o cirugía.

Según la Organización Mundial de la Salud, globalmente, en 2002, más de 161 millones de personas eran discapacitadas visuales, de los cuales 124 millones padecían baja visión y 37 cegueras. En todo el mundo, por cada persona ciega, hay un promedio de 3,4 personas con baja visión.

El porcentaje de visión que conserva cada persona o resto visual, contiene dos parámetros de definición visual: la agudeza y el campo de visión.

La agudeza visual permite distinguir las formas de los objetos que nos rodean. Se mide con el test de Snellen (Fig. 1) de decreciente tamaño, estableciendo cada línea, vista o no, un 10% de agudeza visual.

Figura 1. Test de Snellen

El campo visual es el ángulo que el ojo es capaz de percibir (Fig. 2), correspondiendo 90º a cada ojo (total 180º). El campo se reduce por los costados o de forma aleatoria, a causa de manchas que se forman en el ojo.

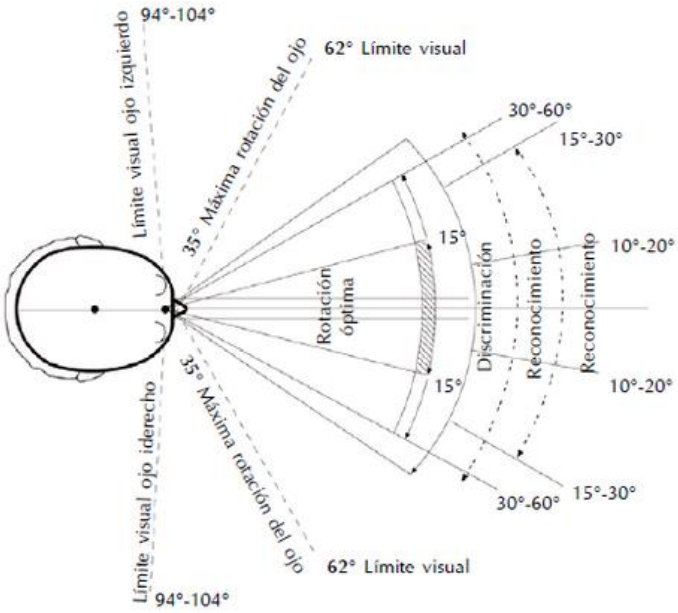


Figura 2. Campo visual

Clasificación del grado de disminución visual

Existen tres niveles para clasificar la disminución visual [5]:

- B3: Las personas que se encuentran en esta categoría no consiguen llegar al 100% de agudeza visual pero supera el 50%. Llevan una vida normal y posiblemente no tengan conciencia de su disminución.

- B2: La gente está en este nivel cuando la visión se encuentra por debajo del 50%, empiezan a existir dificultades notables, a partir del 20% se vuelven imprescindibles ayudas de adaptación. Por debajo del 10% y/o 20^{o} de campo visual se considera que el paciente padece ceguera.

- B1: En esta categoría se encuentran aquellas personas que no ven nada (ceguera total) o solo distinguen zonas difusas de iluminación y sombras.

Principales enfermedades de la visión [6]

- Glaucoma:

  El glaucoma se caracteriza por el aumento de la presión intraocular, por falta de drenaje del humor acuoso, y tiene como condición final la pérdida progresiva de las fibras nerviosas del nervio óptico.

  La pérdida de visión del glaucoma afecta primero a la parte periférica del campo de visión. La pérdida de visión moderada o severa puede ser constatada por el paciente al poner a prueba su visión periférica. Muy frecuentemente, el paciente no detecta pérdida de visión hasta que sufre "visión túnel" (Fig. 3).

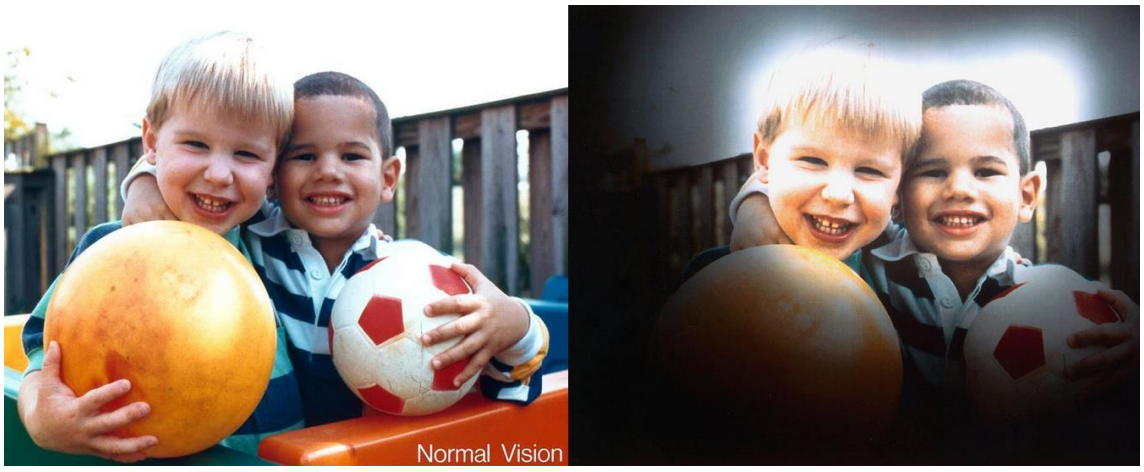


Figura 3. Visión con glaucoma

- Degeneración macular

  Se produce por daños o deterioro de la macula, este área proporciona la agudeza visual, la que permite al ojo percibir detalles finos y pequeños. Cuando la mácula no funciona correctamente, las áreas del centro del campo visual empiezan a perder nitidez, volviéndose turbias y borrosas.

  La rápida pérdida de la visión central es síntoma frecuente de que la persona padece de degeneración macular en ambos ojos. Son indicios de esta enfermedad ver las líneas rectas distorsionadas, ver palabras borrosas, tener problemas para ver con detalle, o tener áreas oscuras o vacías en el centro de la visión (Fig. 4).

Figura 4. Visión con degeneración macular

- Retinopatía diabética

  Es una complicación derivada de la diabetes, causada por el deterioro de los vasos sanguíneos que irrigan la retina. Este daño puede conllevar una fuga de fluido o sangre. Si la enfermedad avanza se forman nuevos vasos sanguíneos y prolifera el tejido fibroso en la retina, lo que tiene como consecuencia que la visión se deteriore, pues la imagen enviada al cerebro se hace borrosa (Fig. 5).

Figura 5. Visión con retinopatía diabética

- Retinosis pigmentaria

  Se trata de un conjunto de enfermedades crónicas de origen genético. Se caracteriza por una progresión degenerativa de la parte sensible a la luz del ojo, la retina, los bastones y los conos.

  Produce como síntomas principales una disminución lenta pero progresiva de la agudeza visual que en las primeras etapas afecta predominantemente a la visión nocturna y al campo periférico, manteniéndose sin embargo la visión central (Fig. 6).

Figura 6. Visión con retinosis pigmentaria

1.1.3. Ayudas ópticas prescritas en la consulta

Las ayudas técnicas de baja visión son aparatos técnicos específicos que mejoran la visión, aprovechando el resto de visión útil.

Existen tres tipos de ayudas [7] [8]:

- Ayudas ópticas

  Utilizan lentes para obtener magnificación en el resto visual. Entre ellas se encuentran filtros ópticos para facilitar la adaptación a la luz o lupas (Fig. 7). Estas últimas son la ayuda a la visión más conocida, con una amplia variedad de diseños dependiendo del fin que persigan.

Figura 7. Lupa

- Ayudas no ópticas

  Son aquellas empleadas para mejorar la postura, el contraste, el deslumbramiento o la distancia de trabajo. Por ejemplo: bastones, texto magnificado, atriles o sistemas de iluminación.

- Ayudas electrónicas

  Entre estas se encuentran gafas de realidad virtual, relojes parlantes u otros aparatos que tratan de dar un estímulo sonoro para contrarrestar la información que se pierde a través del sentido de la vista.

  Un ejemplo de esto es Touch Memo (Fig. 8), es un dispositivo de etiquetado que reproduce pistas de audios pregrabadas en función de la etiqueta que está analizando. Este dispositivo es muy útil para la gente con baja visión ya que le proporciona la información que no pueden obtener mediante la visión a través de la pista sonora.

Figura 8. Touch Memo

El problema de estas ayudas son principalmente dos: existen muy pocos sistemas y son todos muy similares, además el precio de estas ayudas es bastante alto.

1.2. Objetivos

El principal objetivo de este proyecto es desarrollar un sistema de ayuda de bajo coste, para gente con problemas de baja visión como los que se han explicado anteriormente.

Lo que se pretende no es crear un sistema que sustituya los productos de apoyo tradicionales (bastones, perros lazarillo...) a la discapacidad visual, si no crear un sistema complementario que aumente la seguridad en los desplazamientos de sus usuarios.

Más concretamente se va a desarrollar un sistema de realidad aumentada que ofrezca al usuario la información de los contornos de todos los objetos que le rodean, y situar toda esta información en el resto visual del paciente. Debido a la complejidad del sistema se van a desarrollar dos sub-proyectos, uno se encargará de obtener los contornos del entorno y el otro de hallar la distancia a través de un cálculo de profundidades. El proyecto se basará en el desarrollo de los algoritmos de procesamiento utilizando así, un sistema de adquisición y unas gafas de realidad aumentada (Fig. 9).

Tambien se pretende que el sistema sea capaz de ajustar la imagen al campo visual, de tal manera que toda la información se encuentre localizada en el resto visual del paciente, y si asi fuese necesario, que este pudiese modificarlo en función de sus necesidades.

Figura 9. Ejemplo sistema completo

1.3. Especificaciones del sistema

Existen tres aspectos clave a partir de los cuales se van a desarrollar todas las especificaciones y características del sistema.

1.3.1. Bajo coste

Es la base de este proyecto, por esa razón se va a buscar que todos los elementos del sistema sean los más baratos posibles, en concreto, tanto la cámara para la adquisición de imágenes como el sistema de procesamiento.

1.3.2. Versatilidad

Debido a que se trata de un sistema que pretende ayudar a múltiples pacientes, con diferentes tipos de patologías, es necesario un sistema versátil, que se ajuste tanto a cada tipo de enfermedad, como a los deseos personales del paciente.

El sistema pretende ayudar a los pacientes en los desplazamientos tanto en exteriores como en interiores, por lo tanto el sistema tendrá que ser capaz de dar la posibilidad al usuario de especificar el grado de detalle de los contornos que desea ver, e incluso variarlos de color debido a la iluminación del entorno. Por ejemplo, en espacios muy iluminados si los contornos son blancos no se apreciarán correctamente.

1.3.3. Sistema portátil

Al tratarse de un sistema para ayudar a los desplazamientos es necesario que este sea lo más portátil posible, es decir, habrá que buscar un sistema de procesamiento de tamaño reducido, capaz de poder llevarse en el bolsillo o un pequeño bolso.

También se ha de intentar integrar la cámara en el propio soporte de las gafas, de esta manera se podrá reducir al máximo la longitud de los cables buscando así que el sistema sea lo más cómodo posible para el usuario.

Respecto a las gafas, según avanza la tecnología se están desarrollando en el mercado sistemas cada vez más compactos y por tanto más cómodos para utilizar.

No solo hay que tener en cuenta la comodidad del usuario a nivel físico, si no que este se sienta cómodo con su apariencia, hay que recordar que se está buscando un sistema que pretende ayudar a la integración total de las personas en el día a día, pero el usuario rechazara el sistema si ve que este por su apariencia va a provocar un distanciamiento con las demás personas.

1.4. Estado Previo del sistema

Como se ha indicado al comienzo del, todo este trabajo viene precedido por, entre otros, los proyectos de Carlos Barranco [2] y Francisco Collado [3].

En ambos trabajos se desarrollaron ayudas para personas con baja visión, concretamente el proyecto de Carlos Barranco planteó una solución similar a la que se va a desarrollar en el siguiente trabajo, utilizando un sistema que requería gran capacidad de procesamiento, y por tanto, no era portátil. Teniendo en cuenta esto como punto de partida, este proyecto trata de implementar una solución similar de un mayor bajo coste y portátil.

1.5. Medios utilizados

Para la realización del proyecto, se ha hecho uso principalmente de las herramientas de las que se dispone en el laboratorio 1.2.C12 perteneciente al Grupo de Displays y Aplicaciones Fotónicas (GDAF), entre las que se destacan:

- Puesto de soldadura: soldador, estaño, esponjas, des-soldador.
- Útiles de corte y sujeción: alicates, tenacillas de corte, pinzas.
- Material básico: Placa por puntos, pulsadores y cables.
- Software: Visual Studio para desarrollar la programación y Google SketchUp para el diseño 3D.
- Plástico ABS en varios colores.
- Impresora 3D (MakerBot – The Replicator).

Todas las cámaras y las gafas utilizadas a lo largo del proyecto también han sido aportadas por el Grupo de Displays y Aplicaciones Fotónicas.

También se han utilizado conversores HDMI a VGA y cables adaptadores que se han ido adquiriendo a medida que se iban requiriendo.

1.6. Fases del Proyecto

En la siguiente figura (Fig.10) se muestra el diagrama de Gantt con todas las fases de este proyecto. En el diagrama se puede ver que se han diferenciado en 3 colores distintos la parte común que está marcada en verde, el cálculo de contornos de color rojo, que incluye lo desarrollado en este TFG y el cálculo de distancias en color azul, correspondiente al TFG de Rubén Núñez.

En este diagrama se puede observar que se han incluido dos visitas: La primera visita se realizó al Centro de recuperación de personas con discapacidad (CMRF), en el cual se pudo probar el primer prototipo con usuarios potenciales, y la segunda visita se realizó en el Instituto de Oftalmobiologia Aplicada de Valladolid (IOBA), en esta visita se pudo mostrar todas las aplicaciones a oftalmólogos especializados que recalcaron las especificaciones finales del dispositivo.

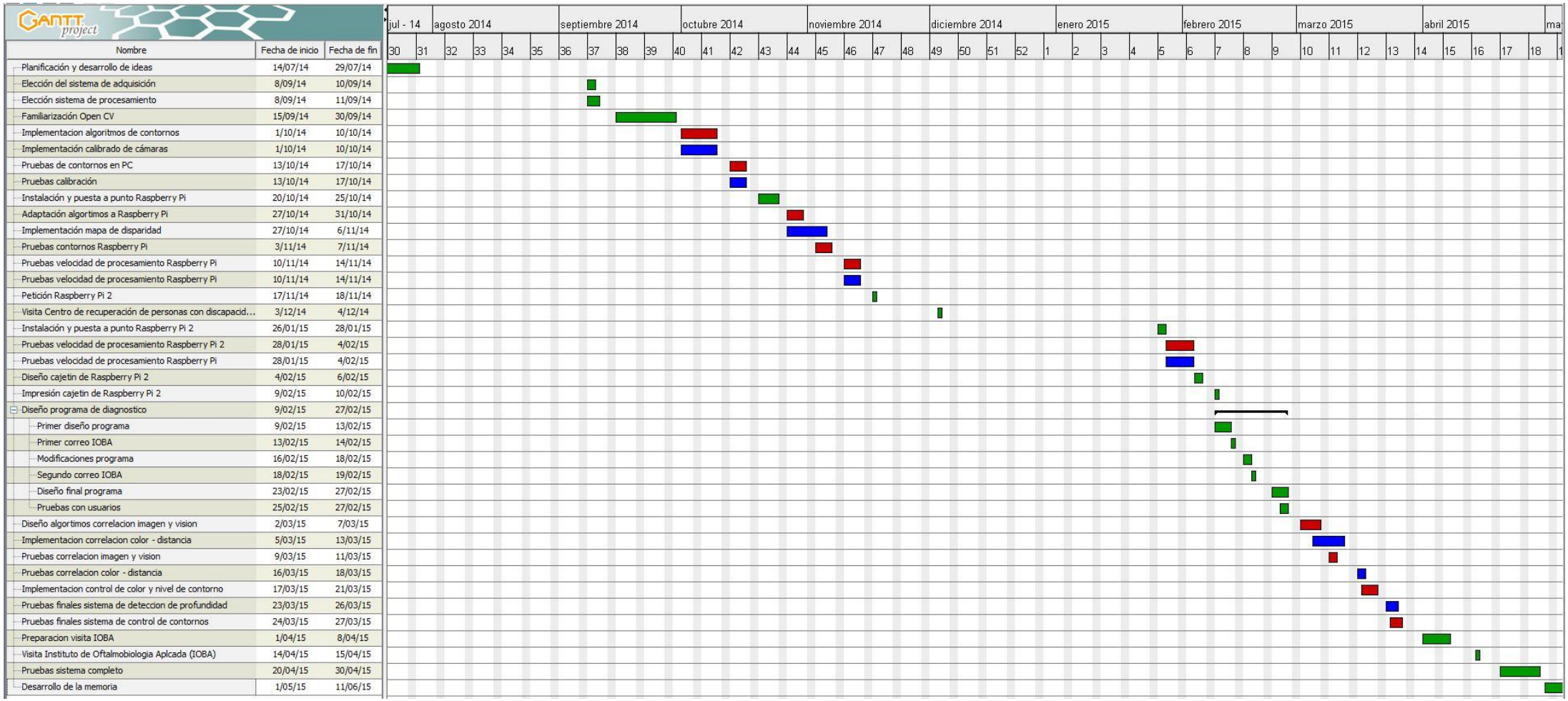


Figura 10. Diagrama de Gantt

Capítulo 2. Diseño del sistema

El sistema descrito a continuación va a estar formado por un sistema de adquisición, el cual capturará las imágenes del entorno, un sistema de procesamiento, que tratará estas imágenes para obtener sus contornos y un sistema de salida que mostrará la imagen ya procesada. Todo esto irá alimentado a través de un batería (Fig. 11).

Figura 11. Sistema propuesto

2.1. Sistema de adquisición de imágenes

Lo primero que ha de hacerse es elegir la cámara o WebCam que se va a utilizar para capturar la imagen y enviarla al procesador que se encargará de procesarla. El Grupo de Displays y Aplicaciones Fotónicas del Departamento de Tecnologías Electrónicas ya cuenta con varios modelos de WebCam, y por tanto se han analizado éstos para elegir la más adecuada en función de los siguientes criterios:

- Resolución: Debido a que el paciente tendrá una pérdida de agudeza visual notable, no es necesario que tenga una gran resolución.

- Velocidad de captura: Este es uno de los puntos clave, debido a que se quiere una aplicación en tiempo real, es necesaria una velocidad de captura muy alta.

- Tamaño y peso: Al tratarse de un sistema portátil, será necesario que la cámara no tenga un tamaño y un peso excesivamente grandes.

- Precio: Se quiere diseñar un sistema de bajo coste, y por tanto hay que tener esta variable muy en cuenta a la hora de elegir la cámara que se va a utilizar.

Aunque en este trabajo solo va a ser necesaria la utilización de una cámara, uno de los objetivos es conseguir un cálculo de distancias, para ello tendremos que utilizar visión estéreo lo cual nos obligará a tener dos cámaras, este también será un factor importante en la elección de la cámara.

2.1.1. Medida experimental de la velocidad de captura

La mayoría de las marcas dan valores de velocidad de captura para resoluciones estándar. Dado que se va a trabajar con resoluciones más bajas de la estándar, se mide la velocidad de captura de manera experimental.

Para realizar esta medida se diseñará un programa muy simple, el cual capturará la imagen de la cámara y lo mostrará por pantalla. Finalmente se medirá el tiempo que pasa entre que se captura la imagen y se muestra cada *frame*.

Dado que estos tiempos son muy bajos y pueden variar por muchos factores, se va a desarrollar una rutina que mida un total de 5000 ciclos, después se tomará la media de todos los tiempos para encontrar un valor aproximado de los procesos por segundo que realiza.

Tanto este programa como los resultados obtenidos vienen especificados en el Capítulo 3, apartado 3.1.

2.1.2. Sony Eye-Camera. PS3 – Webcam

Esta Webcam (Fig. 12) diseñada por Sony está pensada para su uso en la videoconsola Play Station 3. A pesar de estar diseñada para esa plataforma, puede utilizarse perfectamente en un ordenador de sobremesa [9]. Las características principales de esta cámara son:

- Comunicación: USB 2.0.
- Resoluciones: 640x480 pixeles y 320x240 pixeles
- Velocidad de captura (Según el fabricante): 60Hz, 75 fps a resoluciones de 640x480 y 120Hz, 187 fps a resoluciones de 320x240.
- Dimensiones: 80mm × 55mm × 65mm
- Precio: 10 euros

Figura 12. Sony Eye-Camera. PS3 – Webcam

2.1.3. Microsoft Camera 1407

Webcam (Fig. 13) diseña por Microsoft. Las características principales de esta cámara son:

- Comunicación: USB 2.0.
- Resoluciones: 640x480 pixeles
- Velocidad de captura (Según el fabricante): El fabricante no da datos sobre la velocidad de captura
- Dimensiones: 52.1mm × 62.4mm
- Precio: 27,95 euros

Figura 13. Microsoft Camera 1407

2.1.4. Philips SPZ5000

Webcam (Fig.14) diseña por Philips. Las características principales de esta cámara son [10]:

- Comunicación: USB 2.0.
- Resoluciones: de 160x120 pixeles hasta 640x480 pixeles
- Velocidad de captura (Según el fabricante): 60 fps como máximo.
- Dimensiones: 37mm x 56mm x 12 mm
- Precio: 46,28 euros

Figura 14. Philips SPZ5000

2.1.5. Raspberry Pi Camera Module

Módulo diseñado especialmente para el sistema de procesamiento Raspberry Pi. Debido a que uno de los sistemas de procesamiento que se va a probar es la Raspberry Pi se han valorado alternativas adaptadas a esos mini ordenadores. Las características principales de esta cámara (Fig. 15) son [11]:

- Comunicación: CSI (Camera Serial Interface)
- Resoluciones: 640x480 pixeles
- Velocidad de captura (Según el fabricante): 60-90 fps
- Dimensiones: 25mm x 20mm x 9mm
- Precio: 28,95 euros

Figura 15. Raspberry Pi Camera Module

Este módulo cuenta con un problema añadido y es que tiene un protocolo de adquisición de imágenes muy particular, lo que hace que se haga más complejo el uso de las librerías que se van a utilizar en este proyecto.

2.1.6. Minoru

Esta Webcam tiene una ventaja añadida, y es que incorpora dos cámaras ya alineadas, lo que de cara al cálculo de distancias es una ventaja. Además, el GDAF-UC3M ya cuenta con una cámara Minoru (Fig. 16) adaptada para minimizar el tamaño y ser colocados en las gafas [12]:

- Comunicación: USB 2.0.
- Resoluciones: 800x600 pixeles como máximo
- Velocidad de captura (Según el fabricante): 30 fps
- Dimensiones (sin soporte): 93mm x 31mm x 20mm
- Precio: 46,09 euros

Figura 16. Minoru Camera

Habiendo analizado todas las posibilidades podemos realizar un pequeño cuadro resumen para elegir cual será la cámara que se utilizara a lo largo del proyecto. Los valores que se van a dar en la siguiente tabla son los medidos en el Capítulo 3.

Para facilitar la elección, se ha desarrollado una valoración con un baremo de 1 a 5, el criterio que se ha utilizado es el siguiente:

- 2 puntos si es capaz de capturar más de 48 fps (el doble que en tiempo real)
- 2 puntos si el precio de comprar dos cámaras (para poder realizar visión estereoscópica) es inferior a 60 euros y 1 si es inferior a 100 euros.
- 1 punto si ofrece un aporte concreto a este proyecto.

	fps (640x480)	**fps (320x240)**	**Precio**	**Valoración (1 a 5)**
Sony Eye-Camera PS3	60	78	10€ (1 cámara)	4
Microsoft 1407	74	84	27,95€ (1 cámara)	4
Philips SPZ5000	57	75	46,28€ (1 cámara)	3
Raspberry Pi Camera	No medido	No medido	28,95€ (1 cámara)	2
Minoru	60	75	46,09€ (2 cámaras)	5

Desde el punto de vista económico, hay que tener en cuenta que aunque uno de los sistemas más caros es la Minoru Camera, esta incluye dos cámaras, mientras que en las otras alternativas habría que comprar dos cámaras de cara a poder medir distancias en un futuro.

La cámara Philips, se puede considerar demasiado cara teniendo en cuenta que se está buscando un sistema de bajo coste, habida cuenta de que se necesitarían dos. Aunque los resultados de la velocidad de captura son buenos, se asemejan mucho a los de la Minoru Camera, siendo casi el doble de cara.

Como ya se ha dicho anteriormente, la Raspberry Pi Camera tiene un protocolo de adquisición de imágenes muy particular, lo que hace que se haga muy complicado el uso de las librerías que se van a utilizar en este proyecto en comparación con las alternativas de precio similar.

Por tanto, viendo la velocidad de captura y teniendo en cuenta que la cámara de Microsoft esta descatalogada, se va a utilizar para tomar medidas la Sony Eye Camera y la Minoru Camera.

Los resultados se tomarán con ambas cámaras por si finalmente se viera una diferencia al procesar la imagen mucho mejor, en caso de que esto no sea así como se podrá observar en el capítulo 3, se utilizará la Minoru Camera. Además, como ya se explicó el departamento cuenta con unas gafas con la cámara ya incorporada.

2.2. Sistema de procesamiento

Todas las pruebas se harán primero en un ordenador portátil cuyas características se especificarán a continuación, para posteriormente implementar los algoritmos en un sistema de procesamiento portátil, el cual será el que se utilizará en este proyecto.

Como se podrá ver en el Capítulo 3, en un principio se han hecho pruebas con la Raspberry Pi B, y tras ver que los resultados son insuficientes se valoraron otras alternativas.

Características del Portátil

- Procesador: AMD Quad-Core 3.2 GHz
- Disco duro: 1TB
- Memoria RAM: 4GB
- Tipo de sistema: 64 bits
- Sistema Operativo: 1 Partición con Windows 8 y otra con Ubuntu 14.2.

Se ha de buscar un sistema de procesamiento con la potencia suficiente para ser capaz de procesar las imágenes obtenidas por la WebCam en tiempo real, además, el sistema ha de ser de un tamaño reducido para facilitar su transporte. El precio también será un factor limitante en la elección del sistema de procesamiento.

Además de la potencia de procesador y el tamaño, el sistema de procesamiento tiene que contar con puerto USB para poder conectar la cámara. Sobre la conexión del sistema de salida, la mayoría de los sistemas de procesamiento suelen contar con puerto HDMI, en todo caso, no consideraremos limitante el sistema de salida, y si fuese necesario se añadirán conversores dependiendo del tipo de conector que utilicen las gafas.

2.2.1. Raspberry Pi B

Raspberry Pi B(Fig. 17) es un ordenador de placa reducida de bajo coste, desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas[13].

- Puertos USB: 2 USB 2.0.
- Memoria RAM: 512MBytes
- Salida de video: HDMI
- CPU: ARM 1176JZF-S a 700 MHz
- Dimensiones: 85 x 56 x 17 mm
- Precio: 29,95 euros

Figura 17. Raspberry Pi B

2.2.2. Banana Pi

Banana Pi (Fig.18) es una pequeña placa similar a la Raspberry Pi diseñada por la empresa china SinoVoIP, sus características son muy similares aunque esta es algo más potente [14].

- Puertos USB: 2 USB 2.0.
- Memoria RAM: 1 GByte
- Salida de video: HDMI
- CPU: A20 ARM Cortex-A7 Dual-Core a 1GHz
- Dimensiones: 92 mm x 60 mm
- Precio: 65 euros

Figura 18. Banana Pi

2.2.3. PCDuino

PCDuino (Fig. 19) es una placa de mucha potencia capaz de funcionar tanto con Linux como con Android, además, tiene una API completamente desarrollada para poder utilizar la sintaxis de Arduino [15]. Las características principales son:

- Puertos USB: 2 USB 2.0.
- Memoria RAM: 1 GByte
- Salida de video: HDMI
- CPU: AllWinner A20 SoC ARM Cortex A7 Dual Core a 1GHz
- Dimensiones: 121mm X 65mm
- Precio: 59,95 euros

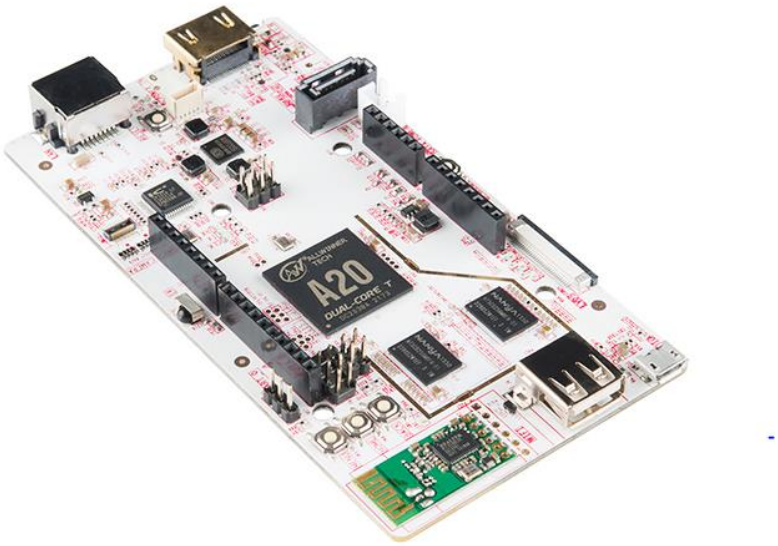


Figura 19. PCDuino

2.2.4. Odroid-XU3

Odroid (Fig. 20) es una potentísima placa capaz de funcionar con Ubuntu o Android, lo único negativo de esta placa es el precio [16]. Las características principales son:

- Puertos USB: 4 USB 2.0. + 1 USB 3.0.
- Memoria RAM: 2 GByte
- Salida de video: HDMI
- CPU: 2 procesadores Samsung Exynos5422 Cortex™-A15 2.0Ghz quad core y Cortex™-A7 quad core
- Dimensiones: 94 x 70 x 18 mm
- Precio: 179 euros

Figura 20. Odroid - XU3

2.2.5. CubieBoard

La placa CubierBoard (Fig.21) al igual que Banana Pi es una pequeña placa fabricada en China por la empresa VIA Technologies [17]. Las características principales son:

- Puertos USB: 2 USB 2.0.
- Memoria RAM: 1 GByte
- Salida de video: HDMI
- CPU: A20 ARM Cortex-A8 a 1GHz
- Dimensiones: 100 x 60 mm
- Precio: 64,90 euros

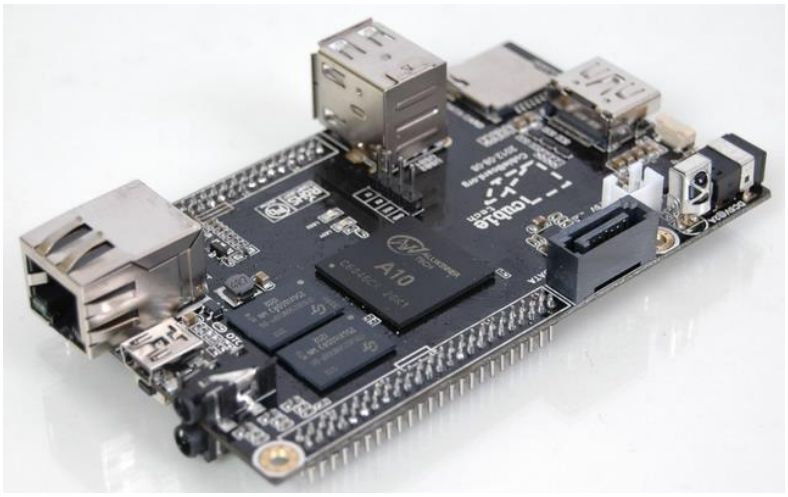


Figura 21. Cubieboard

2.2.6. Udoo Neo

Udoo Neo (Fig. 22) es una placa que pretende unir Raspberry Pi con Arduino. Además a diferencia del resto cuenta con conexión WiFi[18]. Las características principales son:

- Puertos USB: 1 USB 2.0.
- Memoria RAM: 512 MByte
- Salida de video: HDMI
- CPU: ARM Cortex-A9 a 1 GHz
- Dimensiones: 59,3 x 85 mm
- Precio: 49 euros

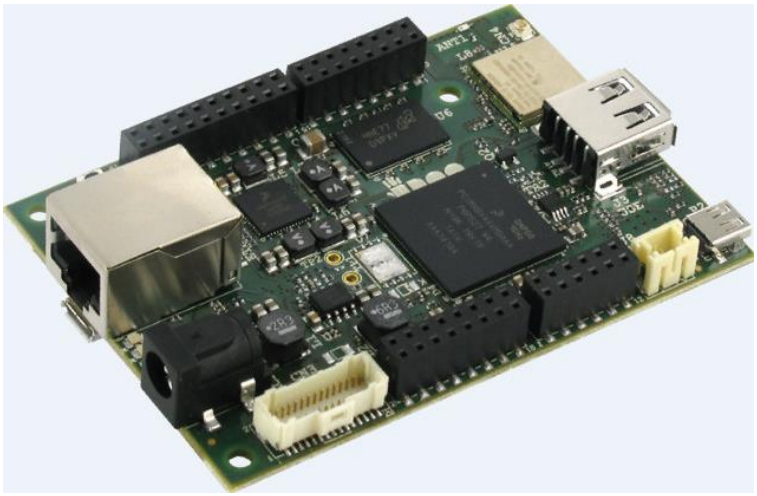


Figura 22. Udoo Neo

2.2.7. Raspberry Pi 2 Modelo B

Raspberry Pi 2 Modelo B (Fig. 23) es una evolución de la placa Raspberry Pi B, la diferencia está en la mejora del procesador pasando a ser 900MHz [13]. Las características principales son:

- Puertos USB: 4 USB 2.0.
- Memoria RAM: 1GByte
- Salida de video: HDMI
- CPU: quad-core ARM Cortex-A7 a 900MHz
- Dimensiones: 85 x 56 x 17 mm
- Precio: 35,30 euros

Figura 23. Raspberry Pi 2 Modelo B

Habiendo analizado todas las posibilidades se puede realizar un pequeño cuadro resumen para elegir cual será el procesador portátil que se utilizará. El criterio que se utilizará para valorar los sistemas de procesamiento es el siguiente:

- 1 punto si la memoria RAM es igual o superior a 1GB.
- 2 puntos si el procesador es superior a 800MHz
- 1 punto si tiene un precio inferior a 50 euros
- 1 punto si tiene, al menos, 2 puertos USB.

	Procesador	RAM	Numero de USB	Tamaño	Precio	Valoración (1 a 5)
Raspberry Pi B	700MHz	512MB	2 USB2.0.	85 x 56 x 17 mm	29,95€	2
Banana Pi	1GHz	1GB	2 USB2.0.	92 mm x 60 mm	65€	4
PCDuino	1GHz	1GB	2 USB2.0.	121mm X 65mm	59,95€	4
Odroid-XU3	2GHz	2GB	4 USB 2.0. + 1 USB 3.0.	94 x 70 x 18 mm	179€	4
CubieBoard	1GHz	1GB	2 USB2.0.	100 x 60 mm	64,90€	4
Udoo Neo	1GHz	512MB	1 USB 2.0.	59,3 x 85 mm	49€	3
Raspberry Pi 2	900MHZ	1GB	4 USB 2.0.	85 x 56 x 17 mm	35,30€	5

Finalmente se va a elegir la Raspberry Pi 2. Esta cumple perfectamente la especificación del precio, además tiene un tamaño reducido y tiene una ventaja añadida, dado que en un principio se trabajó con su versión anterior ya era la que se encontraba en el laboratorio del GDAF-UC3M, no habría que hacer grandes adaptaciones a la hora de buscar un Sistema Operativo, protocolos de conexión, etc… En cuanto a la potencia, no es la más alta de las alternativas, pero se trata del dispositivo con mejor relación Procesador-precio.

2.3. Algoritmos

2.3.1. Sistema Operativo

Dado que finalmente se va a utilizar la Raspberry Pi 2, hay que seleccionar cuál es el sistema operativo que mejor se adapte.

Los sistemas Operativos más comunes en Raspberry Pi y que mejor funcionan son los siguientes [19]:

- ArchLinux: Es un Sistema Operativo de simple entendimiento, está pensado para la ejecución rápida de programas simples y usuarios sin mucho conocimiento. Este sistema queda descartado debido a que está poco optimizado y por tanto los resultados con programas algo más complejos son bastante malos.

- Openelec: Este Sistema Operativo está creado para utilizar la Raspberry Pi como Media Center, aunque está orientado para la reproducción de video, no tiene grandes resultados a la hora de la adquisición y procesamiento de imágenes.

- Rasplex: Es un sistema basado en XBMC que permite tener múltiples dispositivos 'cliente' conectados a un mismo dispositivo 'servidor'. Este sistema está pensado para utilizar la Raspberry Pi como servidor.

- RaspBMC: Es otra distribución basada en XBMC que permite la reproducción de películas, música y vídeos. Al igual que Openlec tiene muy buena respuesta a la hora de reproducir videos e imágenes, pero sus resultados en adquisición y procesamiento de imágenes son pobres.

- Pidora (Fig. 24) : Es una de los mejores Sistemas Operativos para Raspberry Pi, aunque esta distribución tiene peores resultados que ArchLinux y Raspbian para programas simples y queda por detrás de Raspbian para programas complejos.

Figura 24. Logo pidora

- Raspbian: Es el Sistema Operativo más optimizado para Raspberry Pi, tiene versiones para un uso más simple y distribuciones más complejas cuyos resultados son mejores. Este sistema es el que utilizaremos a lo largo de todo el proyecto.

También se van a realizar tanto pruebas en un ordenador portátil como el sistema de diagnóstico, por tanto también se utilizan Windows 8 y Ubuntu 14.2. para realizar las pruebas en el ordenador portátil.

2.3.1.1. Raspbian

Raspbian (Fig. 25) es una distribución del sistema operativo GNU/Linux basado en Debian Wheezy (Debian 7.0). El sistema esta específicamente diseñado para el procesador (CPU) de Raspberry Pi, con soporte optimizado para cálculos en coma flotante por hardware [20].

Esta distribución está diseñada para facilitar el uso de la Raspberry Pi a la vez que la optimiza al máximo frente a otros sistemas operativos, de esta manera se le puede sacar un mayor rendimiento a la vez que se trata de un entorno simple y de fácil entendimiento.

Figura 25. Logo Raspbian

Este Sistema Operativo es el que se va a utilizar durante todo el proyecto en la Raspberry Pi, tanto trabajando con entorno gráfico como sin él, de esta manera se podrá comprobar si existe una mejora de rendimiento y por tanto una mayor velocidad a la salida de la imagen.

2.3.2. Lenguaje de programación

Aunque la Raspberry Pi está especialmente diseñada para facilitar la programación en Python, durante este proyecto se van a diseñar los programas tanto en Python como en C, de esta manera se podrá comprobar con qué lenguaje de programación son más eficientes de los algoritmos.

2.3.2.1. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable [21].

2.3.2.2. Lenguaje C

C es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos [22].

2.3.3. Librerías

Además de las librerías más comunes utilizadas para la realización de programas básicos, se buscó una librería que facilitara el tratamiento de imágenes. De esta manera se simplifica tanto la adquisición de imágenes, como el procesamiento de las mismas. La librería utilizada finalmente fue OpenCV.

2.3.3.1. OpenCV

OpenCV (Fig.26) es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Esta biblioteca pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica. [23]

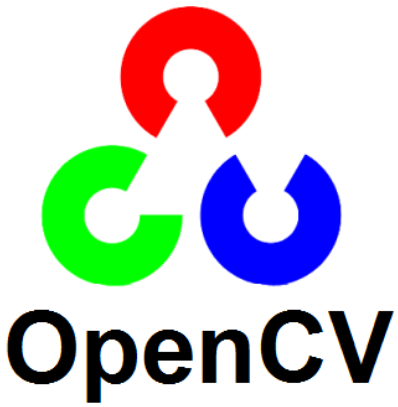


Figura 26. Logo OpenCV

Esta biblioteca se ha utilizado para multitud de aplicaciones, entre otras el sistema de visión de Stanley, el coche no tripulado de la Universidad de Stanford o en el coche inteligente desarrollado en la Universidad Carlos III de Madrid. [24]

2.4. Diseño de los algoritmos

Para el cálculo de los contornos de un objeto existen múltiples algoritmos a utilizar, en este trabajo se van a valorar varios de ellos midiendo su tiempo de ejecución y de esta manera poder determinar cuál es el más adecuado para nuestro sistema. [25]

Vamos a diferenciar entre dos tipos de operadores, operadores basados en el gradiente y operadores más avanzados, en este último caso solo analizaremos el método de detección Canny.

2.4.1. Detectores basados en el gradiente

Las técnicas clásicas de detección de bordes se basan en diferenciar a la imagen, esto es, encontrar la derivada respecto a los ejes x e y, o gradiente. El gradiente de una imagen en un punto indica la variación máxima de la función en ese punto (Fig. 27).

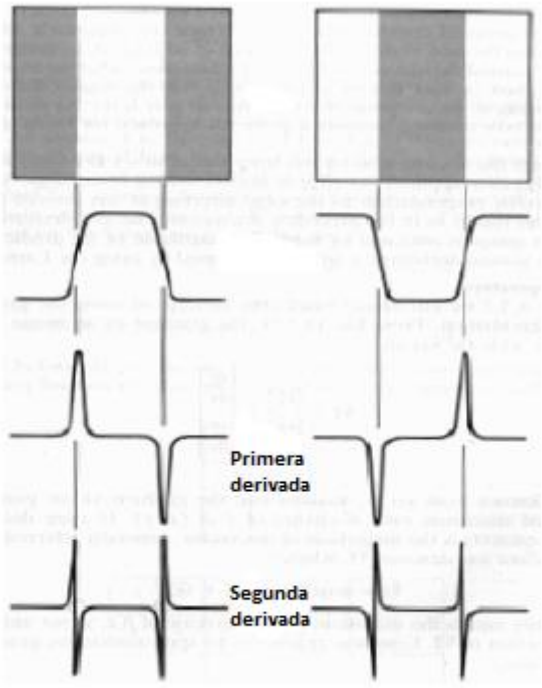


Figura 27. Gradiente [12]

El valor de la magnitud de la primera derivada nos sirve para detectar la presencia de bordes, mientras que el signo de la segunda derivada nos indica si el pixel pertenece a la zona clara o a la zona oscura. Además la segunda derivada presenta siempre un cruce por cero en el punto medio de la transición. Esto puede ser muy útil para localizar bordes en una imagen.

En el caso discreto, se puede aproximar la derivada tomando simplemente la diferencia entre dos valores contiguos. Si se considera una sección de 2x2 de la imagen.

$$\frac{\partial}{\partial x} = I_{1,2} - I_{1,1}$$

$$\frac{\partial}{\partial y} = I_{2,1} - I_{1,1}$$

Roberts

El operador cruzado de Roberts proporciona una aproximación al gradiente usando la técnica de diferencia unidimensional, pero con dos píxeles de diferencia en los ángulos derechos de cada lado como se muestra en la siguiente figura (Fig. 28).

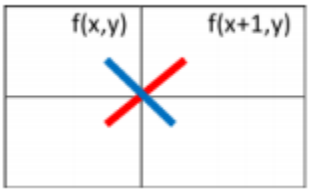


Figura 28. Técnica cálculo del operador Roberts

Presenta como gran desventaja que considera muy pocos píxeles de entrada para hacer la aproximación, lo que provoca que sea muy sensible al ruido y permite solamente marcar los puntos de borde (Fig. 29), es decir, su localización, pero no la orientación de los mismos. No obstante, esta misma desventaja lo convierte en un operador muy simple que trabaja muy bien con imágenes binarias y a una gran velocidad de cómputo.

Figura 29. Operador Roberts

Sobel

Los operadores de gradiente, en general, tienen el efecto de magnificar el ruido subyacente en la imagen, no obstante, el detector de Sobel se puede ver como la combinación de un filtro de suavizado del ruido con un operador de aproximación imprecisa del gradiente. Es decir, la diferencia con el resto de operadores basados en el gradiente es que añade un suavizado previo, de esta manera, se evita ese efecto magnificado del ruido presente en estos filtros.

En este caso particular (Fig. 30) se utiliza una máscara de 3x3, esta máscara se mueve píxel a píxel, calculando el valor del gradiente para cada uno de ellos (el píxel central de la máscara). Una vez obtenido el valor del gradiente se decide si es un borde o no en función de un umbral prefijado.

Figura 30. Operador Sobel

Prewitt

Es un operador similar al de Sobel, pero se diferencia en los coeficientes, ya que Prewitt no enfatiza los píxeles cercanos al centro de la máscara. Se pondera la información de filas y columnas adyacentes para dar mayor inmunidad al ruido.

A diferencia del operador de Sobel, el operador de Prewitt proporciona una mejor detección de los bordes verticales y horizontales en comparación con los bordes diagonales (Fig. 31). No obstante, en la práctica no se aprecia una gran diferencia entre ambos.

Figura 31. Operador Prewitt

2.4.2. Detector Canny

El método de Canny se basa en tres criterios principales:

- El criterio de detección, que expresa el hecho de evitar la eliminación de bordes importantes así como no suministrar falsos bordes.

- El criterio de localización, que establece que la distancia entre la posición real y la posición localizada para el borde debe ser minimizada.

- El criterio de respuesta única, que establece la necesidad de que el detector retorne un único punto por cada punto de borde verdadero. Esto implica que el detector no debe encontrar múltiples píxeles de borde donde solo existe uno.

Uno de los métodos relacionados con la detección de bordes es el uso de la primera derivada, que utiliza el valor cero en todas las regiones donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Por lo tanto, un cambio de intensidad se representa como un cambio brusco en la primera derivada, esta característica es utilizada para detectar un borde, y es la base del algoritmo de Canny.

El algoritmo de Canny consta de tres grandes pasos [24]:

1. Obtención del gradiente: Para la obtención del gradiente, lo primero que se realiza es la aplicación de un filtro gaussiano a la imagen original con el objetivo de suavizar dicha imagen y conseguir la eliminación del ruido que pueda existir.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

2. Supresión no máxima al resultado del gradiente: La salida de este segundo paso es una imagen con los bordes adelgazados después de realizarse la supresión no máxima de puntos de borde.

3. Histéresis de umbral a la supresión no máxima: La imagen que ha sido obtenida en el paso anterior suele contener máximos locales creados por el ruido. Una solución para eliminar dicho ruido es la denominada histéresis del umbral. El proceso de histéresis consiste en tomar la imagen obtenida en el paso anterior, obtener la orientación de los puntos de borde de la imagen y tomar dos umbrales de forma que el primero sea más pequeño que el segundo. Para cada punto de la imagen se debe localizar el siguiente punto de borde no explorado que sea mayor que el segundo valor de umbral.

Figura 32. Operado Canny

OpenCV ofrece funciones optimizadas para varios de estos operadores. Se debe valorar tanto la velocidad de procesamiento como la fiabilidad de los contornos respecto de la realidad. Aunque ahora se van a plantear ventajas e inconvenientes de estos operadores, en el capítulo 3 se pueden ver las pruebas de velocidad de cómputo medidas empíricamente en la Raspberry Pi.

	Ventajas	**Inconvenientes**
Prewitt	Buena respuesta en bordes horizontales y verticales. Poco sensible al ruido. Proporciona la magnitud y dirección del borde	Mala respuesta en bordes diagonales. Lentitud de cálculo. Anchura del borde de varios píxeles.
Sobel	Buena respuesta en bordes horizontales y verticales. Diversidad de tamaños en las máscaras. Proporcionan un suavizado además del efecto de derivación	Mala respuesta en bordes diagonales. Lentitud de cálculo. No da información acerca de la orientación del borde. Anchura del borde de varios píxeles.
Roberts	Buena respuesta en bordes horizontales y verticales. Buena localización. Simpleza y rapidez de cálculo.	Mala respuesta en bordes diagonales. Sensible al ruido. Empleo de máscaras pequeñas. No da información acerca de la orientación del borde. Anchura del borde de varios píxeles.
Canny	Buena respuesta en bordes horizontales, diagonales y verticales. Buena localización. Poco sensible al ruido.	Gran lentitud de cálculo

Los algoritmos que se van a plantear principalmente son Sobel y Canny, en la siguiente figura (Fig. 33) se muestra el diagrama de bloques muy simple de ambos programas. El algoritmo completo puede verse en el Anexo1.

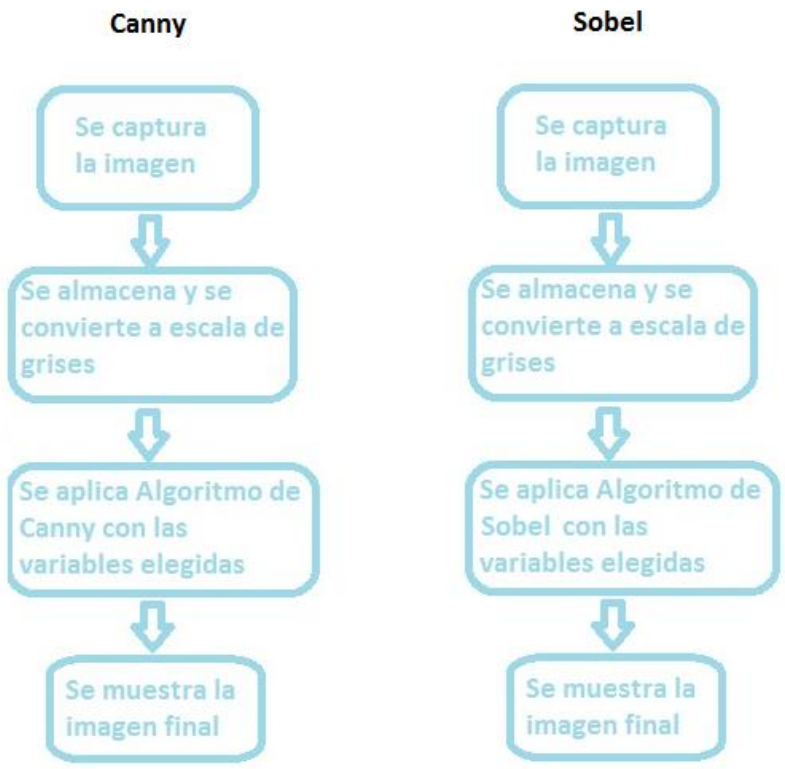


Figura 33. Diagrama bloques Canny y Sobel

2.5. Sistemas de salida

Tras haber sido capturada la imagen por la cámara y procesada por la Rasberry Pi, será necesario seleccionar un sistema de salida para poder ver las imágenes generadas junto con su resto visual. Se está buscando que el sistema completo sea totalmente portable y dado que su uso se va a realizar a diario, las gafas han de ser suficientemente robustas y dentro de las posibilidades de la tecnología lo más cómodas posibles.

Antes de seleccionar las gafas que son más adecuadas, se va a especificar las características principales que deberá cumplir el sistema de salida:

- Peso: Como se ha explicado antes, el peso no ha de ser elevado para facilitar su comodidad, pero en contraposición el sistema ha de ser suficientemente robusto, por tanto habrá que buscar el equilibrio entre estos dos aspectos.

- Número de pantallas: Las gafas podrán ser monoculares o binoculares, se buscarán siempre gafas binoculares, para que de esta manera la imagen vaya a ambos ojos.

- Campo de visión: Las pantallas de las gafas no van a cubrir la totalidad del campo de visión del ojo humano, cuanto mayor campo de visión cubra más información se podrá enviar al usuario debido a que la pantalla podrá llegar a cubrir mayor parte de su resto visual.

- Tecnología: Existen dos tipos de tecnologías, opacas o see-through (Fig. 34). En las primeras el usuario solo podrá ver la imagen generada por las pantallas, mientras que con el sistema see-through el usuario percibirá el entorno con la información de las pantallas y lo que capte con su visión natural.

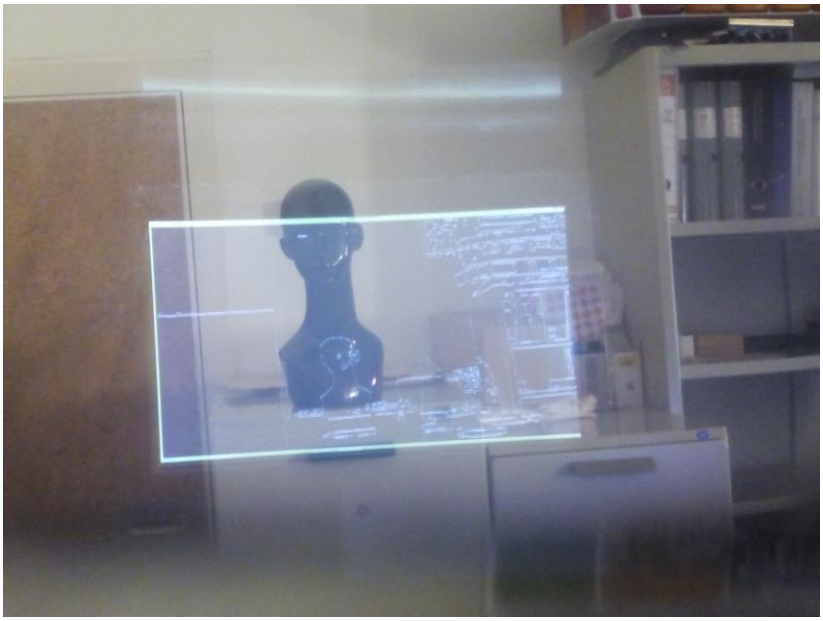


Figura 34. Visión con sistema See-through

Para el proyecto se van a preferir siempre dispositivos con tecnología see-through, esta decisión, tras consultarlo con los expertos del IOBA y con usuarios potenciales, se fundamenta en dos motivos:

1. Para aquellos pacientes cuyo resto visual sea mayor que el campo de visión que cubren las pantallas, utilizar gafas opacas sería contraproducente ya que se está inutilizando esa zona de visión que el usuario aún conserva pero que las pantallas no cubren.

2. El objetivo de este proyecto es facilitar a las personas con baja visión periférica la percepción total de su entorno. El fin de esto es que los pacientes sean más independientes y por tanto se sienta totalmente integrados en el día a día. Al utilizar unas gafas opacas y quitarle la poca percepción directa que tiene, el usuario va a tender a sentirse aislado, ya que solo percibirá lo que ve en una pantalla negra con unos contornos de colores, no podrá percibir, por ejemplo, la luminosidad del ambiente.Esto hará que el usuario rechace el producto ya que considerará que el sistema le va a aislar más.

- Tamaño: Este punto está directamente relacionado con lo expuesto en el punto anterior, cuanto más grandes y aparatosas sean las gafas, más posibilidades hay de que el paciente las rechace por el miedo a “hacer el ridículo” o “ir llamando la atención por la calle”. Hoy en día, la tecnología en estos dispositivos aumenta a gran velocidad, por tanto no es descabellado pensar que en unos cuantos años se podrán contar con unas gafas similares a las que se utilizan para la corrección de problemas leves de visión.

- Conexión: Las gafas van a ir conectadas al sistema de procesamiento, en el caso de la Raspberry Pi la salida de video es HDMI, por tanto sería importante encontrar un dispositivo cuya conexión sea así, igualmente este factor tampoco es importante ya que se pueden colocar conversores entre medias.

- Precio: Al tratarse de un sistema de bajo coste, este factor será de los más importantes. En la actualidad estos sistemas de salida son bastante caros, aun así, como se ha indicado antes la tecnología en este aspecto aumenta muy rápido y por tanto el precio se irá reduciendo progresivamente.

Estas son las alternativas que se han valorado:

MV -1(Fig. 34) [25]

- Monocular
- See-through
- Resolución: 320x240
- Campo de visión: 10-15 grados
- Conexión: VGA
- Peso: 28 gramos
- Precio: 475€

Figura 34. MV - 1

EPSON MOVERIO BT-200 (Fig. 35) [26]

- Binocular
- See-through
- Resolución: 960x540
- Campo de visión: 23 grados
- Conexión: Micro USB
- Peso: 880 gramos
- Precio: 634,77€

Figura 35. Epson Moverio BT-200

Trivisio SXGA61 3D (Fig. 36) [25]:

- Binocular
- Opaca
- Resolución: 1280 x 1024
- Campo de visión: 43-45 grados
- Conexión: HDMI
- Peso: 290 gramos
- Precio: El fabricante no lo indica

Figura 36. Trivisio SXGA61

Vuzix Wrap 1200DXAR (Fig. 37) [27]

- Binocular
- See-through
- Resolución: 852 x 480
- Campo de visión: 35 grados
- Conexión: VGA
- Peso: menos de 1 kg
- Precio: $1499.00

Figura 37. Vuzix Wrap 1200DXAR

Vuzix STAR 1200XLD (Fig. 38) [28]

- Binocular
- See-through
- Resolución: 852 x 480
- Campo de visión: 35 grados
- Conexión: VGA
- Peso: menos de 1 kg
- Precio: $4999.00

Figura 38. Vuzix STAR 1200XLD

Excepto por el campo de visión, la que mejor se adapta a las necesidades del sistema son las gafas fabricadas por Epson. Aun así, y debido al precio de estos sistemas y de la falta del presupuesto para adquirir cualquiera de estos sistemas, solo se van a valorar aquellas gafas que están disponibles en el GDAF-UC3M, es decir Trivisio SXGA61, Vuzix Wrap 1200DXAR y Vuzix STAR 1200 XLD.

De las dos gafas con sistema See-through, dado que las características son similares, finalmente se han elegido las gafas STAR 1200XLD, ya que David Barranco [2] para su proyecto las modificó y llevan incorporada a las gafas la cámara Minoru como se puede ver en la siguiente figura (Fig.39).

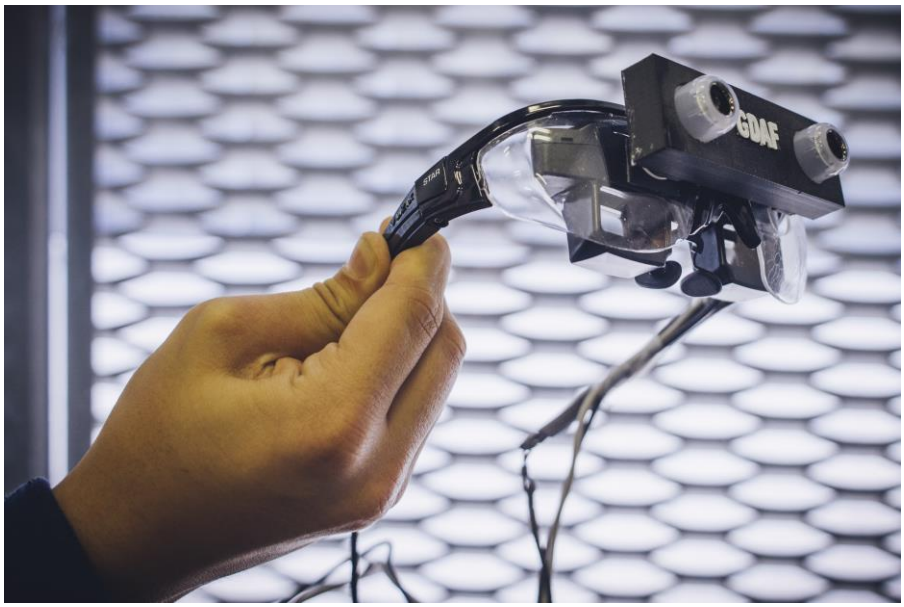


Figura 39. Vuzix STAR con Minoru Camera

2.6. Batería

Tanto las gafas como el sistema de procesamiento han de estar alimentados. Hay que buscar algún sistema portátil que sea capaz de alimentar ambos sistemas. La solución que se ha encontrado es la utilización de unas baterías portátiles usadas actualmente para cargar móviles, están fabricadas por la empresa PowerBank[29], estas baterías tienen una capacidad de 2600mAh, haciendo pruebas experimentales conectando una de estas baterías a los sistemas, se han alcanzado más de 5 horas de autonomía.

Estas baterías (Fig. 40) además de tener un tamaño bastante reducido, tienen un precio de 6 euros, manteniendo así la premisa del bajo coste.

Figura 40. Bateria portatil

2.7. Montaje final del sistema

Tras la elección del sistema de adquisición, sistema de procesamiento y sistema de salida, ya queda montado el sistema final (Fig. 41) con sus correspondientes sistemas de comunicación.

Figura 41. Esquema sistema completo

Aunque en el sistema final se ha añadido como sistema de adquisición a la Minoru camera, las pruebas se van a realizar también con la Sony Eye-Camera. PS3 – Webcam, esto se debe a que los resultados que ofrece el fabricante son bastante buenos y su precio es muy reducido, por tanto como ya se explicó en el punto 2.1. se van a valorar ambos sistemas de adquisición durante el resto del proyecto.

A continuación, se puede ver el sistema completo montado (Fig. 42)

Figura 42. Sistema completo

Por tanto, el sistema final provocará sobre el usuario un entorno como el que se muestra en la figura siguiente (Fig. 43): a través de las gafas el usuario verá los contornos obtenidos por la cámara, superpuestos a la imagen real, pudiendo modificar tanto el tamaño como la posición de la imagen.

Figura 43. Imagen original v.s. imagen modificada por el paciente

Capítulo 3. Pruebas y resultados experimentales

3.1. Pruebas iniciales y resultados experimentales

Antes de nada, es preciso explicar cómo se va a realizar la medición del tiempo de ejecución de cada uno de los programas.Dado que todo el procesamiento ha de ir englobado en un bucle que irá trabajando con cada *frame* según le llegue, lo que se va a medir es el tiempo de ejecución de un bucle. Es decir, se va a medir el tiempo que tarda desde la captura de la imagen, el procesamiento de esta y su envío hacia el sistema de salida.

Esta medición se va a realizar a través de la función time.clock() de la librería time.h. En este caso se utilizará Python. Esta función proporciona el tiempo de ejecución en ese momento desde el inicio del programa. Por tanto, al inicio del bucle mediremos este tiempo y lo guardaremos en una variable, al finalizar el bucle realizaremos el mismo proceso, y la resta de ambas variables nos dará el tiempo que se ha tardado en completar el bucle.

3.1.1. Resultados experimentales del sistema de adquisición de imágenes

Como se ha explicado en el apartado 2.1. para poder ver la velocidad de la cámara a nivel experimental para distintas resoluciones se ha diseñado un pequeño programa.

Las mediciones se van a realizar a 4 resoluciones. Las dos básicas para este tipo de cámaras (640x420 y 320x240) una por debajo (160x120) y la que a priori utiliza nuestro sistema de salida (852x480).

El sistema de procesamiento que se va a utilizar es el portátil mencionado en el capítulo anterior. No se van a tomar medidas de adquisición con los sistemas de procesamiento valorados anteriormente, ya que en este paso solo se quiere medir el tiempo de adquisición para valorar la elección de la cámara, las valoraciones sobre la velocidad de captura con el sistema de procesamiento se harán a posteriori.

El programa diseñado realizara dos funciones:

Primero (Fig. 44), capturará la imagen de la cámara, se establece la resolución y la guardará en una variable de tipo Mat[1] (Este programa ha sido diseñado en Python).

[1] La variable de tipo Mat, es una variable en la cual se puede almacenar matrices en escala de grises o en color, de uno o varios canales. De esta manera podemos almacenar nuestra imagen y trabajar con ella como una matriz de pixeles.

```
from cv2.cv import *
import time

captura = CaptureFromCAM(0)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_WIDTH,640)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_HEIGHT,480)
NamedWindow("Fotograma", 1)
text = open("datos.txt","w")
print ("Entrando en el bucle principal")
i= 0
for i in range(5000):
        start_time = time.clock()
        fotograma = QueryFrame(captura)
        ShowImage("Fotograma", bn2)
        tecla = WaitKey(1)
        print (i)
        elapsed_time = time.clock() - start_time
        elapsed_time = elapsed_time
        text.write(str(elapsed_time))
        text.write('\n')

text.close
```

Figura 44. Primer programa (capturar imagen)

Tras esto se mostrará por pantalla a través de la función ShowImage (Fig. 45), para ello se ha creado una ventana donde se colocará la imagen para mostrarla.

```
from cv2.cv import *
import time

captura = CaptureFromCAM(0)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_WIDTH,640)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_HEIGHT,480)
NamedWindow("Fotograma", 1)
text = open("datos.txt","w")
print ("Entrando en el bucle principal")
i= 0
for i in range(5000):
        start_time = time.clock()
        fotograma = QueryFrame(captura)
        ShowImage("Fotograma", bn2)
        tecla = WaitKey(1)
        print (i)
        elapsed_time = time.clock() - start_time
        elapsed_time = elapsed_time
        text.write(str(elapsed_time))
        text.write('\n')

text.close
```

Figura 45. Primer programa (mostrar imagen)

Para medir los tiempos de procesamiento se va a calcular como se ha explicado anteriormente los “procesos por segundo”. Esta medición se va a realizar durante 5000 ciclos, de esta manera aseguramos no quedarnos con resultados anómalos. Estas 5000 muestras se guardarán en un fichero el cual después se volcarán sus datos en un Excel para que de esta manera podamos realizar la media y los cálculos necesarios para tener la medida necesaria. En azul se muestra cómo se calcula el tiempo y en rojo como se guarda en un fichero (Fig. 46).

```
from cv2.cv import *
import time

captura = CaptureFromCAM(0)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_WIDTH,640)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_HEIGHT,480)
NamedWindow("Fotograma", 1)
text = open("datos.txt","w")
print ("Entrando en el bucle principal")
i= 0
for i in range(5000):
        start_time = time.clock()
        fotograma = QueryFrame(captura)
        ShowImage("Fotograma", bn2)
        tecla = WaitKey(1)
        print (i)
        elapsed_time = time.clock() - start_time
        elapsed_time = elapsed_time
        text.write(str(elapsed_time))
        text.write('\n')

text.close
```

Figura 46. Primer programa (Medir tiempos y guardarlos)

Se ha hecho el análisis con todas las cámaras comentadas en el Capítulo 2. En la siguiente figura (Fig.47) se muestran los resultados experimentales de estas mediciones.

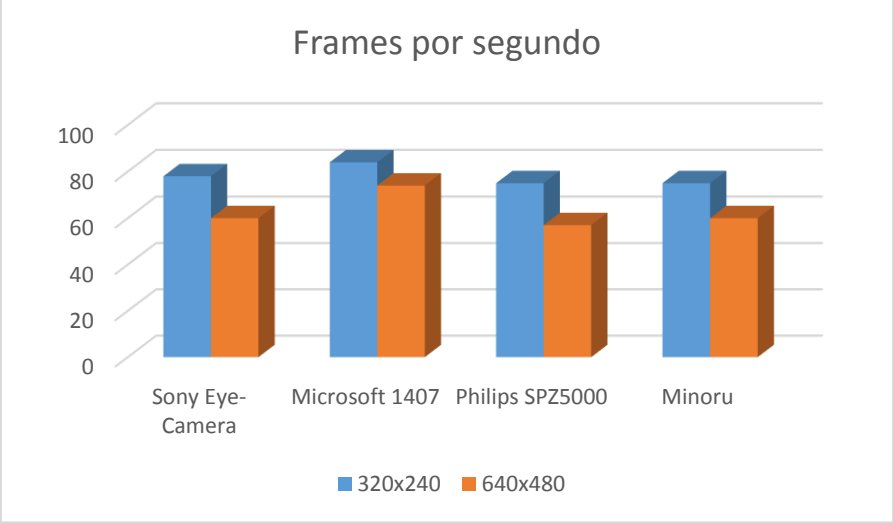


Figura 47. Comparativa fps

Como ya se comentó en el capítulo 2, la cámara que mejor resultado da es la Microsoft 1407, el problema en este caso, es que este modelo se encuentra descatalogado, y su valoración solo se planteó debido a que la cámara se encontraba en el laboratorio, y si hubiese dado resultados muy superiores al resto, se hubiese buscado una alternativa similar del mismo fabricante. Los resultados de las tres cámaras restantes, como se puede apreciar, son muy similares, por tanto, el factor limitante será el precio, descartando de esta manera la cámara de Philips ya que si fuese necesario la compra de dos unidades para realizar la visión estereoscópica, el precio sería muy elevado (aproximadamente 90 euros).

Por tanto, y como se indicó en el capítulo 2 las pruebas se van a realizar con la Minoru y la Sony Eye-Camera.

3.1.2. Resultados experimentales del tiempo de ejecución de los algoritmos

Para calcular la velocidad de ejecución se van a valorar dos operadores con dos cámaras distintas. Concretamente se van a comprobar Sobel y Canny, ya que estos dos operadores están específicamente optimizados con OpenCV.

Para poder comprobar estos algoritmos se repite el proceso explicado anteriormente, para poder ejecutarlos se va a capturar la imagen, almacenar esta y posteriormente tratarla. El tratamiento se basa en dos pasos básicos.

Primero se convierte la imagen a escala de grises, para esto utilizamos la función cvtColor (Fig. 48).

```
// Pasamos el frame a una variable de tipo imagen
 frame=cvQueryFrame(capture);
 // Si esta no existe enviamos mensaje de error
 if(frame.empty()){
     printf("No se puede acceder");
     return -1;
     }
 //Pasamos de BGR a escala de grises
 cvtColor(frame, img, CV_BGR2GRAY);
```

Figura 48. Conversión a escala de grises

Tras esto se aplica un filtro gaussiano para suavizar la imagen y se utiliza uno de los algoritmos de contornos (Fig. 49).

- Sobel

```
//Aplicamos un suavizado Gaussiano
GaussianBlur(img, img, Size(7,7), 1.5, 1.5);
//Aplicamos el operador Sobel
Sobel(img, img, CV_8U,0,1,3);
```

Figura 49. Suavizado y Operador Sobel

Como se puede ver en la imagen del código, la función Sobel tiene varias variables que controlar.

Sobel (imagen de entrada, imagen de salida, profundidad o niveles, orden de la derivada en x, orden de la derivada en y, tamaño de la rejilla utilizada (esta dependerá del nivel de los gradientes utilizados)).

En la siguiente figura (Fig.50) se puede ver cómo si se utiliza una rejilla muy grande (de 5x5 por ejemplo) para gradientes en x e y de orden bajo (1 y 0 en este caso), se tendrán unos contornos muy anchos y mal definidos debido a que existe un alisado desde el centro de la rejilla hasta los puntos más lejanos de la cuadricula.

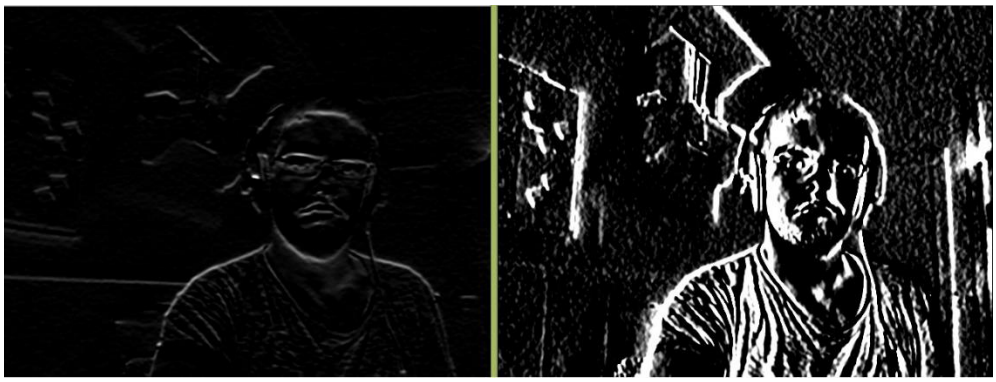


Figura 50. Prueba con diferentes tamaños de rejilla

También hay que comprobar qué orden de las derivadas es el más apropiado, se han realizado pruebas con todas las combinaciones posibles hasta la segunda derivada, es decir, el momento en el que el contorno se representa por un paso por el cero (Fig. 51). Según aparecen, en orden de izquierda a derecha y de arriba abajo se tienen los siguientes órdenes en x e y: (0,1), (1,0), (1,1), (2,1), (1,2), (2,2).

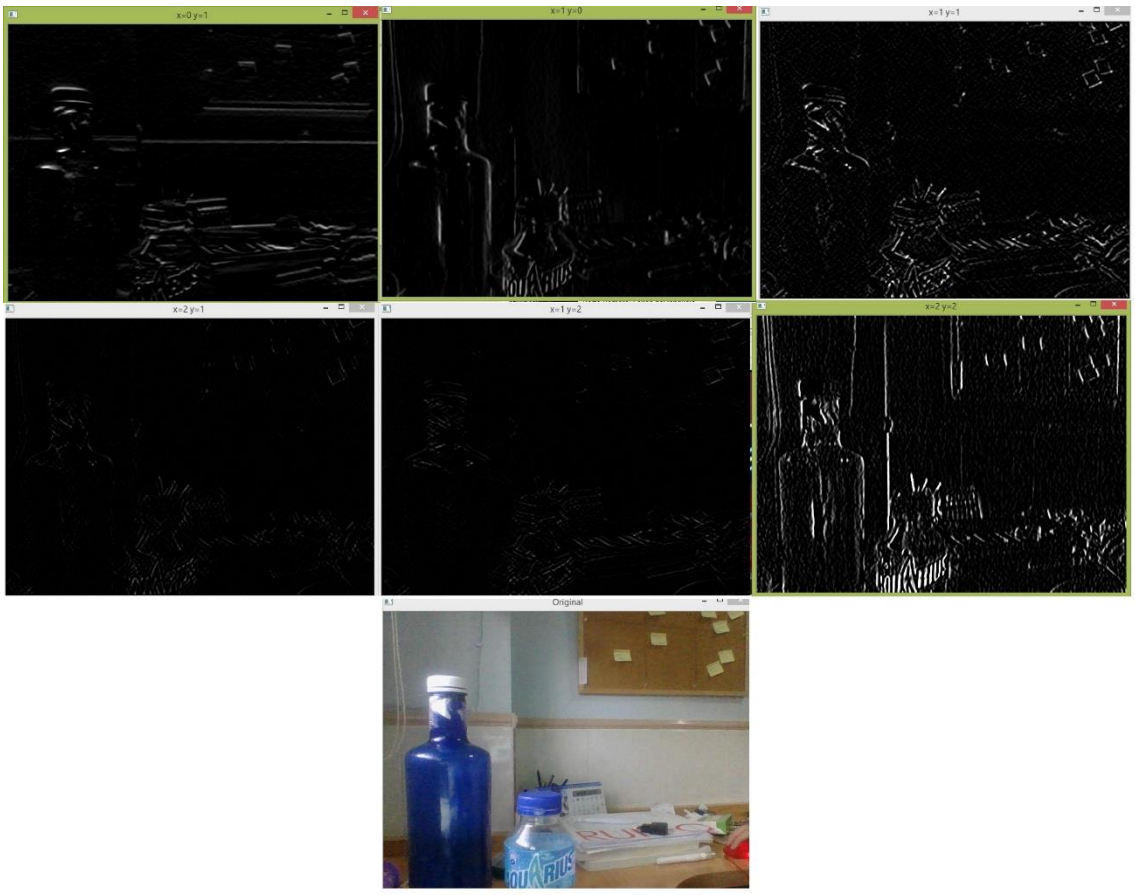


Figura 51. Resultados Sobel

Como se puede observar, a órdenes muy superiores del gradiente, los bordes están mejor definidos pero existe más ruido debido a que es más sensible a variaciones y cualquier oscilación que pase por cero se interpretara como un borde.

- Canny(Fig. 52)

```
//Aplicamos un suavizado Gaussiano
GaussianBlur(img, img, Size(7,7), 1.5, 1.5);
//Aplicamos el operado Canny
Canny(img, img, 50,150, 3);
```

Figura 52 Suavizado y operado Canny

Como se puede ver en la imagen del código, la función Canny tiene varias variables que controlar.

Canny (imagen de entrada, imagen de salida, primer umbral, segundo umbral, tamaño de la rejilla).

A priori, es recomendable utilizar un tamaño de rejilla de 3x3 para el filtro gaussiano, esto se debe a que el objetivo de este primer filtro es el de suavizar la imagen para eliminar el ruido, con rejillas muy superiores el operador se vuelve muy sensible al ruido debido a que los órdenes de las derivadas para el cálculo del gradiente han de ser mayores. Igualmente estos son los resultados aplicando, con los mismos umbrales, distintos tamaños de rejilla (Fig. 53). (De izquierda a derecha de arriba abajo, imagen original, 3x3, 5x5, 7x7).

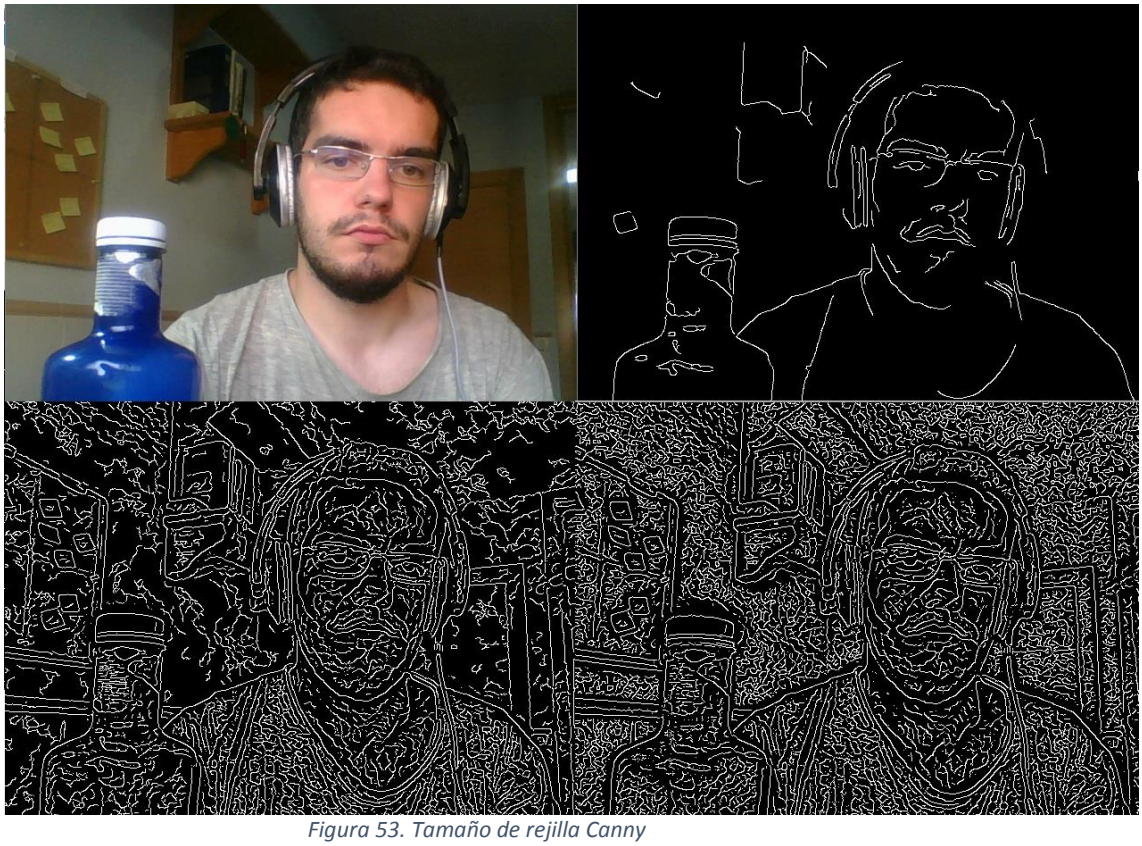


Figura 53. Tamaño de rejilla Canny

Como se ha explicado antes, se puede ver que el efecto del ruido es mayor, y al realizar la histéresis de umbral este ruido se acrecienta.

También hay que comprobar el mejor nivel de umbrales para la histéresis, estos umbrales ayudan a eliminar los posibles máximos locales creados por efecto del ruido (Fig. 54). (En la imagen se muestran umbrales (50 – 100), (50 - 150), (100 - 200), (100 - 250).

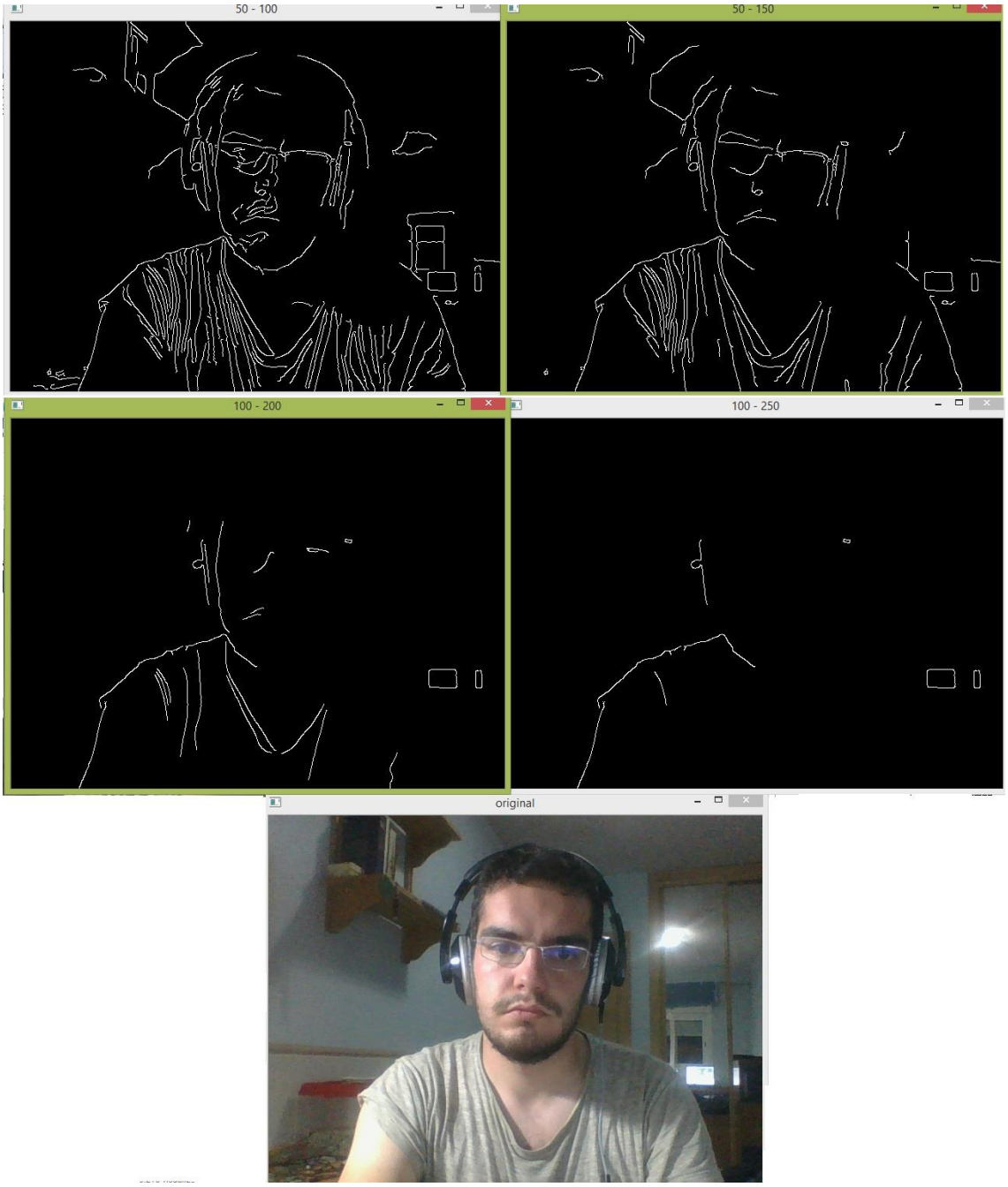


Figura 54. Pruebas de umbral Canny

Como se ve, cuanto mayor sean los umbrales menor detalle se tendrá ya que se filtran a niveles más altos en la escala de grises. El nivel de estos umbrales dependerá mucho del nivel de luminosidad que haya en la habitación, por esta razón, y como se expone en este capítulo, se realizará un control de umbral para que el usuario pueda elegir estos umbrales.

Las pruebas se realizan utilizando como sistema de procesamiento la Raspberry Pi, Raspberry Pi 2 y el ordenador portátil mencionado en el capítulo 2, y como sistema de adquisición la Minoru Camera y la Sony Eye-Camera.

Para que estas pruebas sean veraces de cara a su futura implementación, se realizaron a varias resoluciones y a pantalla completa. Se midieron los tiempos de ejecución un total de 10 veces para evitar tomar datos atípicos debido a procesos internos del sistema que hicieran que se ralentizara el procesamiento.

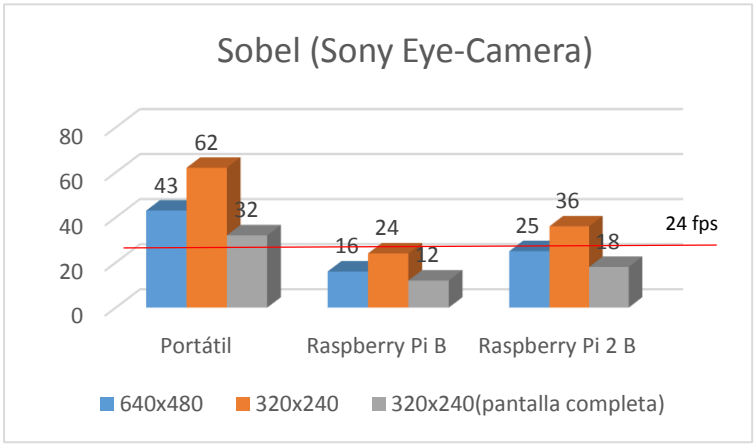
Sobel

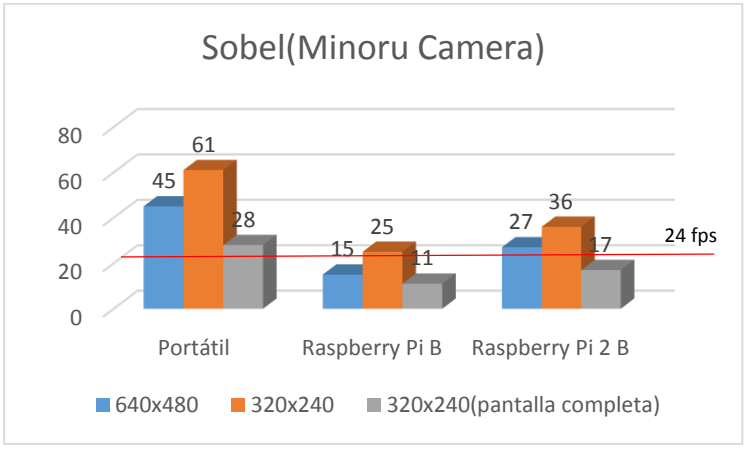
- Sony Eye-Camera

	Portátil	**Raspberry Pi B**	**Raspberry Pi 2 B**
640x480	43 ± 3 *fps*	16 ± 2 *fps*	25 ± 2 *fps*
320x240	62 ± 3 *fps*	24 ± 1 *fps*	36 ± 3 *fps*
320x240(pantalla completa)	32 ± 2 *fps*	12 ± 1 *fps*	18 ± 1 *fps*

- Minoru Camera

	Portátil	**Raspberry Pi B**	**Raspberry Pi 2 B**
640x480	45 ± 2 *fps*	15 ± 1 *fps*	27 ± 4 *fps*
320x240	61 ± 2 *fps*	25 ± 1 *fps*	36 ± 1 *fps*
320x240(pantalla completa)	28 ± 2 *fps*	11 ± 2 *fps*	17 ± 2 *fps*





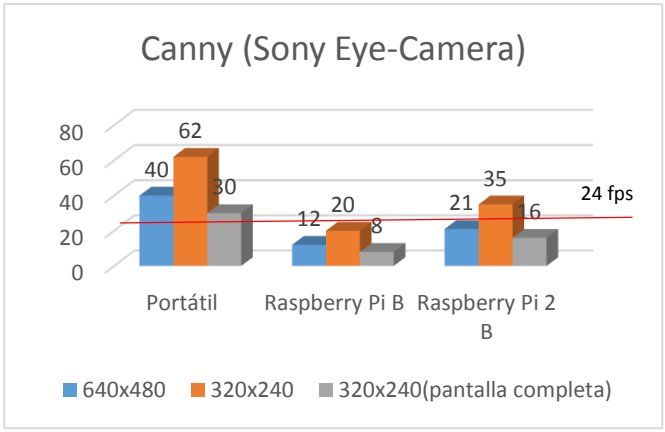
Canny

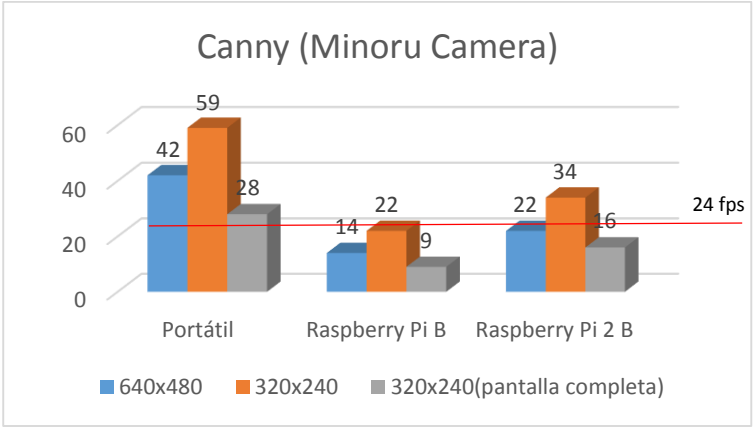
- Sony Eye-Camera

	Portátil	Raspberry Pi B	Raspberry Pi 2 B
640x480	40 ± 1 *fps*	12 ± 2 *fps*	21 ± 2 *fps*
320x240	62 ± 2 *fps*	20 ± 1 *fps*	35 ± 1 *fps*
320x240(pantalla completa)	30 ± 2 *fps*	8 ± 2 *fps*	16 ± 1 *fps*

- Minoru Camera

	Portátil	Raspberry Pi B	Raspberry Pi 2 B
640x480	42 ± 2 *fps*	14 ± 1 *fps*	22 ± 1 *fps*
320x240	59 ± 1 *fps*	22 ± 2 *fps*	34 ± 2 *fps*
320x240(pantalla completa)	28 ± 2 *fps*	9 ± 1 *fps*	16 ± 2 *fps*





Como se puede observar en la tabla anterior, no existe una gran diferencia en el tiempo de ejecución entre los dos tipos de cámaras y aunque sí que el algoritmo Sobel tiene mejores resultados, no existe una gran diferencia en el procesamiento. Según las opiniones recogidas por pacientes y expertos, la información que nos da el operador Sobel puede ser a veces difícil de comprender ya que sus contornos no están tan bien definidos, por tanto dado que la diferencia en ejecución no es muy elevada, se considerará el Operador Canny como la mejor opción ahora mismo.

Como se ve anteriormente, los resultados obtenidos no son suficientes para conseguir tiempo real ni en la Raspberry Pi ni en la Rapberry Pi 2. El mejor resultado obtenido a pantalla completa es de 16 *frames* por segundo, estando este valor muy por debajo de los 24 *fps* que se consideran tiempo real. Por mucho que se baje la resolución de pantalla, el redimensionar la imagen a tamaño completo supone un gran gasto en el sistema de procesamiento.

La Raspberry Pi 2 si consigue valores por encima del tiempo real para resoluciones de 320x240. Pero en resoluciones mayores y en esa resolución en pantalla completa los valores de velocidad no son suficientes. Esto no es suficiente ya que solo el sistema

seria útil para aquellas personas que tenga un angulo de visión muy reducido y un tamaño de pantalla pequeña es suficiente.

Aunque se barajaron alternativas y mejoras, ninguna de ellas permitía ese salto de potencia necesaria. Debido a esto y como se podrá leer en el capítulo 4, se plantea una alternativa mejor de cara a futuros trabajos con el uso de FPGAs.

3.1.3. Alternativas al diseño

Como se ha visto en el punto anterior no se ha conseguido alcanzar una potencia suficiente para obtener una imagen en tiempo real, debido a esto se barajó una alternativa utilizando la GPU de la Raspberry Pi. [30]

La unidad de procesamiento gráfico o GPU (Graphics Processing Unit) es un coprocesador dedicado al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central. De esta manera se puede tener un procesamiento en paralelo, es decir, se podrán ir realizando varias funciones en paralelo al procesamiento del ordenador.

A nivel de programación, cabe destacar CUDA [31]. CUDA es un compilador y un conjunto de herramientas de desarrollo que permite el uso de una variación del lenguaje de programación C para codificar algoritmos en GPU. Respecto a CUDA se han realizado multitud de pruebas procesando imágenes, incluso *OpenCV* incluyo en su paquete una librería para la utilización de alguna de sus funciones a través de la GPU de los procesadores *nVidia*.

Para programar accediendo a la GPU en dispositivos que no son *nVidia*, se ha desarrollado una API como es *OpenGL*, esta API está pensada, entre otras, para ser utilizada en dispositivos ARM, como el que tiene la Raspberry Pi.

Cabe destacar que desde 2012 el código controlador *VideoCore* fue lanzado por la Fundacion Raspberry Pi bajo licencia BSD. Esta licencia no permite desarrollar aplicaciones en *open source*.

Para analizar las posibilidades de la programación en la GPU de la Raspberry Pi, se ha usado como referencia el trabajo de Pete Warden [32] y trabajos similares [33]. Este trabajo se centró en la optimización de Raspberry Pi utilizando su GPU.

Se analizaron los códigos y procesos seguidos en estos trabajos, pero los resultados no se consiguieron obtener. El problema que surgió fue que al procesar dichos algoritmos salían errores de permisos a nivel interno del procesador. Estos errores no se pudieron solucionar y por esta razón y debido a la falta de tiempo, se acabó descartando la idea del uso de la GPU.

Dado que las alternativas, debido al gasto de tiempo en las pruebas con los sistemas de procesamiento y con la GPU (Graphic Process Unit), no daban tiempo a realizarse para su posible ejecución, se optó como se puede ver en el siguiente punto, en mejorar la versatilidad del sistema y realizar pruebas para saber la opinión de los usuarios potenciales. De esta manera tanto las pruebas como las mejoras de los algoritmos podrán ser utilizadas en un futuro cuando se cambie el sistema de procesamiento.

3.2. Implementación final

3.2.1. Ajuste de la imagen procesada al campo visual del paciente

Esta especificación es un requisito que los usuarios han solicitados tras las pruebas que realizó Carlos Barranco durante el desarrollo de su proyecto [2].

Dado que el sistema ha de ser versátil, el ajuste de la pantalla al entorno real tiene dos variantes:

- Baja percepción visual: no existe una zona de visión limitada, es decir, el resto visual engloba la totalidad de lo mostrado por las gafas, en este caso, el usuario podrá desplazar la imagen y aumentarla o reducirla para, de esta manera, ajustar la imagen a su campo de visión.

- Baja visión periférica: en este caso, el resto visual del paciente no cubre la totalidad de las pantallas de las gafas, y por tanto habrá que ajustar la imagen a la zona donde ve. Para ello se ha diseñado un programa de "diagnóstico".

3.2.1.1. Ajuste de la imagen sin pérdida de visión periférica

Como se ha explicado anteriormente, en este caso no se debe ajustar la imagen al resto visual del paciente, ya que su resto visual es mayor que el ángulo de visión de las pantallas de las gafas. Por tanto, lo que se va a realizar es un sistema que permita al usuario colocar la imagen generada por el microprocesador alineada con la imagen real que está viendo a través del sistema see-through.

La primera limitación existente es que se deben tener ambos horizontes alineados, es decir, que tanto la visión normal del usuario como las pantallas estén alineadas al mismo nivel. Para conseguir esto evitando modificaciones a nivel hardware, se ha de dar la oportunidad al usuario de controlar el desplazamiento de la pantalla en horizontal y en vertical. Al realizar esta modificación obviamente se pierde información, ya que si, por ejemplo, subimos la imagen, toda la zona inferior quedará vacía. Como se ha comentado se va a desplazar la pantalla a nivel software, en ningún momento a nivel hardware, por tanto, nos será imposible captar otra información a través de las cámaras por mucho que modifiquemos la posición de la pantalla.

Una vez ajustado el horizonte y la posición total de la imagen, puede ser necesario hacer zoom a la imagen para que de esa manera quede totalmente alineada la imagen real y el contorno correspondiente a este. Para esto se va a determinar una región de interés (Region of interest o ROI), esta guarda los parámetros de lo captado por la cámara pero se centra justo en una zona menor a la imagen global, de esta manera el usuario podrá elegir el nivel de zoom que quiere presionando un botón para hacer zoom y otro para volver al estado anterior. Hay que tener en cuenta, que al hacer zoom de una zona se pierde información de los alrededores, por tanto el usuario también podrá mover la imagen tanto en vertical como en horizontal, para de esta manera colocar el zoom en la zona que desee. En la siguiente figura (Fig. 55) se puede ver de izquierda a derecha y de arriba abajo: Imagen original, imagen modificada, modificada con contornos en verde y modificada con contornos en azul.

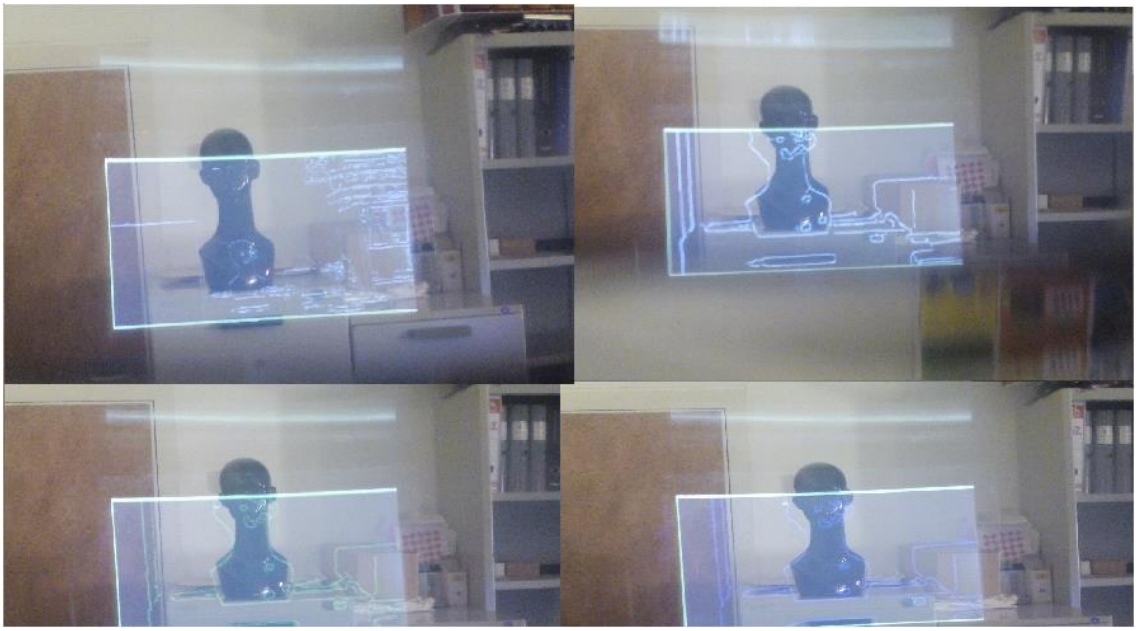


Figura 55. Desplazamiento y cambio de color

Para realizar todo este proceso, se van a añadir una serie de botones, los cuales permitirán modificar los parámetros antes mencionados. Este sistema, va a estar conectado directamente a la Raspberry Pi, por lo que, el usuario podrá tener un acceso directo a la configuración de sus sistema. Para la conexión de los botones en la Raspberry Pi se ha hecho uso de las librerías GPIO de Raspberry [34] [35], la implementación de estas librerías al algoritmo puede verse en el Anexo 1 de este documento.

Este sistema también se ha implementado en un ordenador, en este caso, se han utilizado las teclas para manejar las variables de control.

A continuación, se muestra la distribución de los botones y la función de cada uno de estos. En el caso de la Raspberry Pi, vienen referenciados a los pin de entrada y salida con los que esta cuenta (Fig. 34) y el esquema que se implementa (Fig. 35).

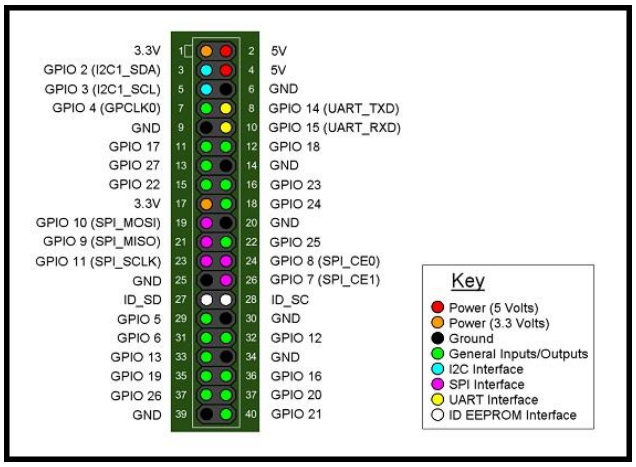


Figura 56. GPIO Raspberry Pi [34]

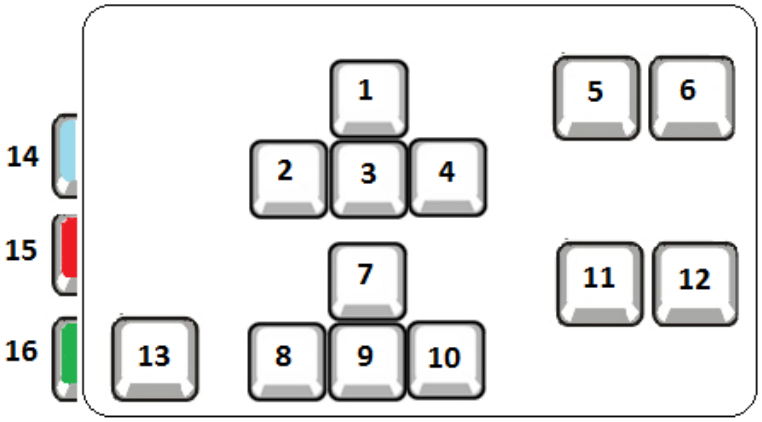


Figura 57. Esquema botones

Boton	Raspberry Pi	Portatil	Función
1	GPIO4	w	Desplazar la ventana hacia arriba
2	GPIO17	a	Desplazar la ventana hacia la izquierda
3	GPIO27	s	Desplazar la ventana hacia abajo
4	GPIO22	d	Desplazar la ventana hacia la derecha
5	GPIO5	+	Aumentar el nivel de los contornos
6	GPIO6	-	Disminuir el nivel de los contronos
7	GPIO20	t	Desplazar la imagen hacia arriba
8	GPIO21	f	Desplazar la imagen hacia la izquierda
9	GPIO16	g	Desplazar la imagen hacia abajo
10	GPIO12	h	Desplazar la imagen hacia la derecha
11	GPIO23	k	Hacer Zoom
12	GPIO24	j	Deshacer Zoom
13	GPIO13	b	Contornos en color blanco
14	GPIO19	c	Contornos en color azul

15	GPIO26	r	Contornos en color rojo
16	GPIO18	v	Contornos en color verde

A continuación (Fig. 58), se muestra el conexionado de uno de los pines, siendo el resto similar:

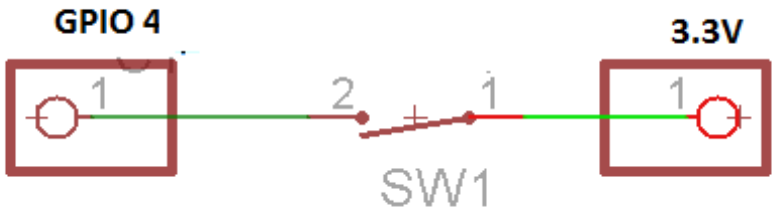


Figura 58. Conexionado botones

El diagrama de flujo del programa se muestra en la siguiente figura (Fig. 59).

Figura 59. Diagrama de flujo

El funcionamiento del sistema final se puede ver en el siguiente video.

https://www.youtube.com/watch?v=jvrwHb3j6dQ
Última vez consultada: 18/6/2015

3.2.1.2. Ajuste de la imagen con pérdida de visión periférica

En el caso de que el paciente tenga una visión por debajo de lo que abarcan las gafas (35 grados de visión periférica), será necesario ajustar la pantalla a su resto visual. Para realizar esto, se ha diseñado un programa, el cual el paciente utilizará la primera vez que vaya a usar las gafas. Este proceso se realizara en un ordenador y con la supervisión de un oftalmólogo.

Para diseñar este programa se ha contado con la ayuda de los expertos del Instituto de Oftalmobiología Aplicada de Valladolid (IOBA), ya que, al estar en contacto directo a diario con pacientes, nos pudieron dar una visión mucho más concreta de las necesidades y requisitos que debía cumplir el programa.

El desarrollo del programa se puede dividir en tres fases:

- Fase 1: En esta fase se le mostrarán varias imágenes al paciente, y con ayuda del ratón deberá dibujar un rectángulo en la zona que el considere que su visión es correcta y por tanto la zona donde el desea que aparezca la pantalla. Para evitar falsos rectángulos al clicar sin querer, una vez dibujado el rectángulo se le va a pedir que confirme dicho rectángulo o lo rechace (Fig. 60). De imagen de fondo se muestra una rejilla de Amsler[1]. Esto les ayudará a tener una referencia en todo momento.

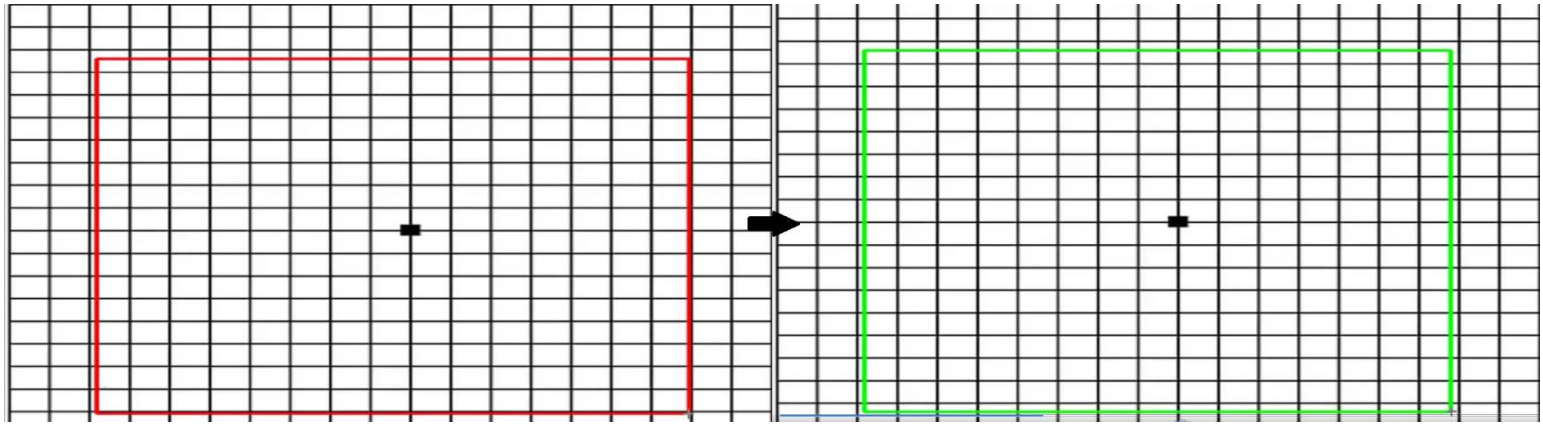


Figura 60. Diagnostico 1

- Fase 2: Una vez dibujados un total de 8 rectángulos, se le irán mostrando uno a uno al paciente. Según aparezca por pantalla el rectángulo el paciente podrá confirmar que las dimensiones le parecen bien o borrarlo si por algún casual considera que es pequeño o demasiado grande.

- Fase 3: En ésta se muestran todos los rectángulos restantes, el usuario podrá, colocando el cursor sobre el rectángulo, destacar uno concreto y si lo desea eliminarlo. De esta manera solo quedarán aquellas zonas que el paciente considere óptimas, ya sea por tamaño o por deseo personal debido a la colocación (Fig. 61).

[1]La rejilla de Amsler, es una imagen muy utilizada para el diagnóstico de patologías relacionadas con la visión, en la prueba se le solicita al paciente que se coloque a una distancia concreta de la imagen y ha de observar que todas las líneas sean rectas, en caso contrario existe alguna deficiencia en la visión.

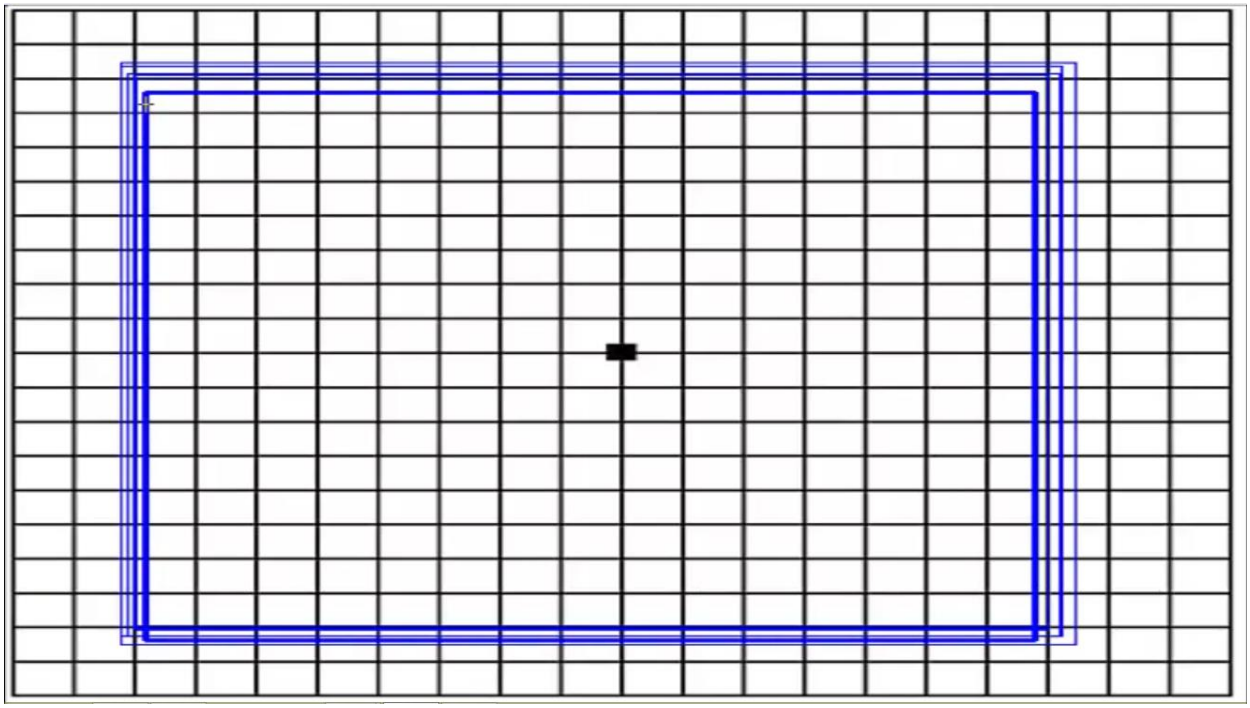


Figura 61. Diagnostico 2

Una vez finalizado este proceso, el programa calcula el área más grande de los rectángulos restantes y guarda en un documento de texto tres valores: La esquina superior, el largo y el ancho del rectangulo.

Al cargar la raspberry pi, esta leerá el documento de texto y mediante la función moveWindow y resizeWindow, colocaremos la ventana en la posición deseada y con el tamaño que se ha seleccionado. La implementación de estas funciones se puede ver en el código, presente en el Anexo 1.

Se puede ver un ejemplo de la ejecución del programa de diagnóstico en el siguiente video.

https://www.youtube.com/watch?v=8T8egk_Zvfc
Última vez consultada: 18/6/2015

3.2.2. Pruebas en entorno real y modificaciones finales

Dado que el sistema no alcanzaba un rendimiento suficiente para alcanzar un procesado en tiempo real (24 *fps*), las pruebas se realizaron utilizando un ordenador portátil. A continuación se muestran un de vídeo del uso del sistema en un entorno real.

https://www.youtube.com/watch?v=9z1-2W3DfPM&feature=youtu.be
Última vez consultada: 18/6/2015

También cabe destacar en este apartado que dado que no había financiación, se decidió desde el GDAF-UC3M intentar un proceso de CrowdFunding[1] apoyado en la convocatoria que lanzó en 2014 la UC3M. De esta manera, además de solicitar financiación, al llegar a más gente, pudimos obtener mucho más *feedback* sobre el proyecto y de esta manera poder realizar varias mejoras. Este proyecto se realizó a través de la web goteo.org [36], y a continuación se muestra un enlace al video de presentación, en el cual se puede ver un pequeño resumen de todo el proyecto desarrollado a lo largo de este TFG y [1].

https://www.youtube.com/watch?v=15i9GaD_CmQ&feature=youtu.be
Última vez consultada: 18/6/2015

De cara a su implementación futura en otro sistema de procesamiento, se diseñó un encapsulado que integrara tanto el sistema de procesamiento como toda la botonería para controlar el sistema. Esta última, fue soldada en una placa por puntos (Fig. 62).

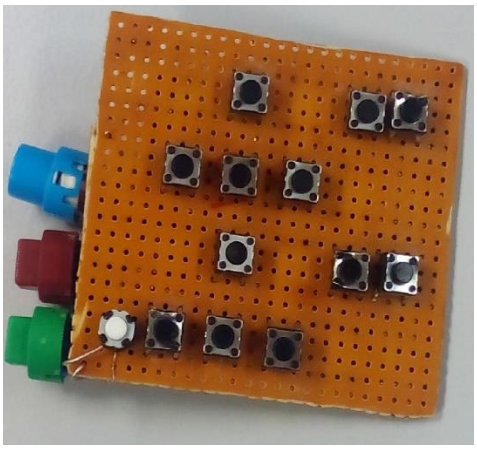


Figura 62. Placa con botones

De esta manera se puede conectar la placa a la Raspberry Pi de la siguiente forma (Fig. 63)

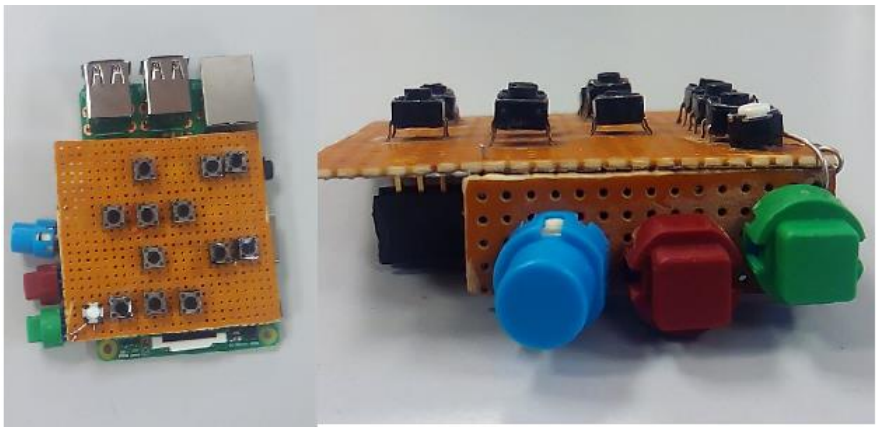


Figura 63.Montaje Raspberry con botones

[1]Crowdfunding es un sistema de financiación colectiva, de uso muy extendido en la actualidad. A través de diversas plataformas, un usuario con una idea o proyecto, presenta este para que gente interesada en él pueda aportar dinero.

El último paso, es la creación de una carcasa de plástico utilizando la impresora 3D del laboratorio 1.2.C12 del GDAF-UC3M. En este caso, aunque los diseños están preparados, no ha sido posible su implementación debido a problemas técnicos con la impresora.

A continuación (Fig.64) y (Fig. 65) se muestran los planos de la caja diseñada, esta se taladraría posteriormente para que el usuario pueda manipular los botones [37].

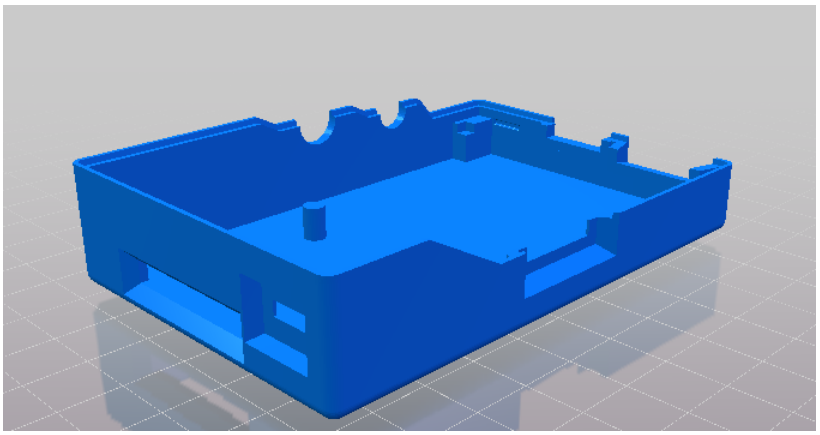


Figura 64. Base carcasa

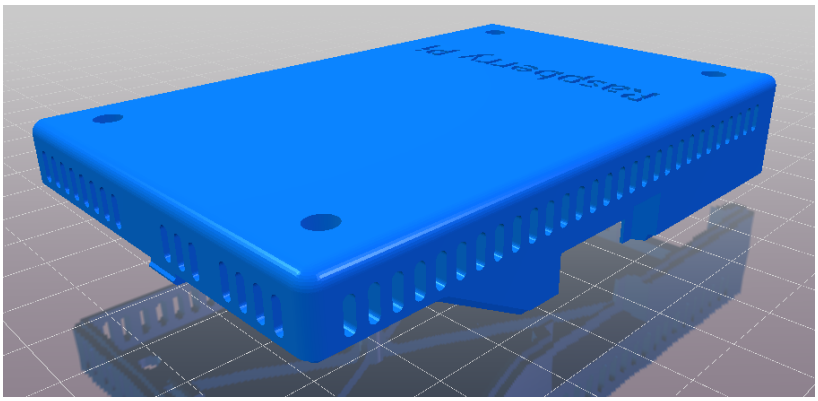


Figura 65. Tapa carcasa

Una vez completado todo el sistema, el montaje del mismo se puede ver en el siguiente video:

https://www.youtube.com/watch?v=bxur_UdWApA&feature=youtu.be
Última vez consultado: 20/06/2015

Toda la información sobre el montaje aparece en el manual de usuario en el apartado de Anexos.

3.3. Presupuesto

Para calcular el coste total tanto del sistema como del proyecto, se van a presentar dos presupuesto: Un primer presupuesto, el cual tendrá en cuenta todos los recursos utilizados en el proyecto, incluyendo todos los elementos que se han dimensionado (Cámaras, sistemas de procesamiento…). El segundo será el presupuesto del sistema, es decir, el coste de la realización de un sistema completo totalmente funcional.

Como se podrá observar en ambos presupuestos, se han añadido los honorarios de los ingenieros del proyecto. Para tener en cuenta este coste se ha establecido un salario medio de 26 €/hora y un total de 240 horas de trabajo distribuidas de la siguiente manera:

Tarea	Número de horas	Coste total
Diseño de los algoritmos y selección de los dispositivos	60	1560€
Pruebas y fabricación	150	3900€
Documentación	30	780€

Por tanto, el coste total de los honorarios ascenderá a 6.240€. Teniendo en cuenta que en el proyecto han trabajado 2 ingenieros, el coste total será 12.480€.

La siguiente tabla muestra el coste total del proyecto. Incluyendo los elementos que no serán parte del sistema final, pero que si han sido probados. En este apartado faltaría añadir la amortización de la impresora 3D, por ejemplo. Los elementos que si han sido nombrados a lo largo del capítulo 2 pero no figuran en este proyecto, no han sido probados y por tanto no ha sido necesaria su adquisición.

CÓDIGO	UNIDAD DE MEDIDA	DESCRIPCIÓN	CANTIDAD	PRECIO (UNITARIO)	IMPORTE TOTAL
		CAPÍTULO 1: Sistema de adquisición			
1.01	unidades	Minoru Web-Camera 3D	1	46,09 €	46,09 €
1.02	unidades	Sony Eye-Camera PS3	1	10,00 €	10,00 €
1.03	unidades	Microsoft Camera 1407	1	27,95 €	27,95 €
1.04	unidades	Philips SPZ5000	1	46,28 €	46,28 €
1.05	unidades	Raspberry Pi Camera Module	1	28,95 €	28,95 €
			TOTAL CAPÍTULO 1: Sistema de adquisición		159,27 €
		CAPÍTULO 2: Sistema de procesamiento			
02.01	unidades	Bobina de plástico ABS Filamento de plástico ABS negro de 1.75mm. 1 Kg	1	17,95 €	17,95 €
02.02	unidades	Interruptores Interruptor basculante de 2 posiciones negro	10	0,83 €	8,30 €

02.03	unidades	Placa de puntos. Placa C.I. 2 cuadrados 100x160MM	1	7,00 €	7,00 €
02.04	unidades	Tarjeta microSDHC Verbatim 16GB Clase 10	1	15,00 €	15,00 €
02.05	unidades	Bateria Power Bank A5 Classic 2600mAh	1	5,99 €	5,99 €
02.06	unidades	Raspberry Pi 2 Modelo B	1	35,30 €	35,30 €
02.07	unidades	Raspberry Pi B	1	29,29 €	29,95 €
			TOTAL CAPÍTULO 2: Sistema de procesamiento		119 €
		CAPÍTULO 3: Sistema de salida			
03.01	unidades	Gafas Vuzix STAR 1200XLD	1	4.999 €	4.999 €
03.02	unidades	Cable adaptador, VGA hembra HDMI macho	1	10,75 €	10,75 €
03.03	unidades	Gafas Vuzix Wrap 1200DXAR	1	1.499 €	1.499 €
			TOTAL CAPÍTULO 3:Sistema de salida		6.498 €
		CAPÍTULO 4: Honorarios Ingenieros			
04.01	horas	Honorario a los dos ingenieros al cargo del proyecto, suponiendo un salario para cada uno de 26 €/hora	240	26 €	6.240 €
			TOTAL CAPÍTULO 4: Honorarios Ingenieros		12.480 €
				TOTAL	19.257 €

La siguiente tabla muestra el coste total del sistema. Es decir, el coste de un sistema formado por cámara, batería, sistema de procesamiento, gafas y todos los elementos necesarios para el correcto funcionamiento del sistema.

CÓDIGO	UNIDAD DE MEDIDA	DESCRIPCIÓN	CANTIDAD	PRECIO (UNITARIO)	IMPORTE TOTAL
		CAPÍTULO 1: Sistema de adquisición			
1.01	unidades	Minoru Web-Camera 3D	1	46,09 €	46,09 €
			TOTAL CAPÍTULO 1: Sistema de adquisición		46,09 €
		CAPÍTULO 2: Sistema de procesamiento			
02.01	unidades	Bobina de plástico ABS Filamento de plástico ABS negro de 1.75mm. 1 Kg	1	17,95 €	17,95 €

02.02	unidades	Interruptores Interruptor basculante de 2 posiciones negro	10	0,83 €	8,30 €
02.03	unidades	Placa de puntos. Placa C.I. 2 cuadrados 100x160MM	1	7,00 €	7,00 €
02.04	unidades	Tarjeta microSDHC Verbatim 16GB Clase 10	1	15,00 €	15,00 €
02.05	unidades	Batería Power Bank A5 Classic 2600mAh	1	5,99 €	5,99 €
02.06	unidades	Raspberry Pi 2 Modelo B	1	35,30 €	35,30 €
			TOTAL CAPÍTULO 2: Sistema de procesamiento		90 €
		CAPÍTULO 3: Sistema de salida			
03.01	unidades	Gafas Vuzix STAR 1200XLD	1	4.999 €	4.999 €
03.02	unidades	Cable adaptador, VGA hembra HDMI macho	1	10,75 €	10,75 €
			TOTAL CAPÍTULO 3: Sistema de salida.		5.010 €
		CAPÍTULO 4: Honorarios Ingenieros			
04.01	horas	Honorario a los dos ingenieros al cargo del proyecto, suponiendo un salario para cada uno de 26 €/hora	240	26 €	6.240 €
			TOTAL CAPÍTULO 4: Honorarios Ingenieros		12.480 €
				TOTAL	17.625 €

Observando ambos presupuesto, se puede ver que el coste total del proyecto es de 19.257 €. Aunque esta cifra es muy elevada hay que tener en cuenta que la mayoría de los recursos presentes en este presupuesto, y por tanto, utilizados a lo largo del proyecto, ya se encontraban disponibles por el GDAF-UC3M.

El coste total del sistema asciende a 17.625€, hay que tener en cuenta, que existen dos elementos que engloban más del 90% del presupuesto.

En primer lugar los honorarios, estos son un 70% del coste total del sistema, en este caso están dimensionados teniendo en cuenta que se realice un solo sistema, es decir, el impacto de los honorarios sobre el presupuesto se verá reducido según vayamos aplicando economía de escala.

El segundo elemento son las gafas, estas engloban un 28% del presupuesto, hay que indicar que la elección de las gafas que finalmente se han utilizado, se ha debido a la disponibilidad en el GDAF-UC3M, por tanto, estos costes se podrían ver reducidos optando por otras de las gafas valoradas en el apartado 2.5., por ejemplo, las EPSON MOVERIO BT-200[15] tienen un coste casi diez veces menor que las de la marca Vuzix, 634€.

Capítulo 4. Conclusiones y posibles líneas futuras

4.1. Conclusiones

Se ha conseguido un sistema capaz de procesar los contornos del entorno y mostrarlos a través de un sistema de salida. El sistema es suficientemente versátil para adaptarse tanto a cada tipo de enfermedad o lo severa que sea esta, como a los deseos personales de cada usuario.

Por otro lado, las pruebas realizadas en entornos reales y la opinión de los médicos del IOBA, han demostrado que el sistema es útil para los pacientes y pueden servir como una mejora de la calidad de vida del usuario en sus desplazamientos.

La principal especificación era la de tener un sistema de bajo coste, si se valora el precio total del sistema (cámara, sistema de procesamiento, gafas y baterías) la cifra asciende a 5.145€. Este precio aún está muy por encima de considerarse como bajo coste, aunque hay que tener en cuenta que la solución aquí planteada ofrece una gran información sobre las profundidades del entorno y posibilita al usuario el control de varios parámetros. También hay que indicar que los costes debido a las gafas se podrían ver reducidos optando por otras de las gafas valoradas en el apartado 2.5. El sistema sin contar las gafas asciende a 135€, un precio que si se aproxima a un sistema de bajo coste.

El sistema es capaz de adaptar la imagen al resto visual del paciente, y permitir a este que realice las modificaciones que desee. Aun así, se ha hablado de la colocación de un total de 16 botones, una cifra muy elevada, esto choca completamente con varios de los principios del diseño para todos. Uno de los trabajos que hay que realizar como línea futura será la reducción al máximo de los botones y controles manteniendo la versatilidad del sistema. En el siguiente punto se podrán ver varias soluciones planteadas para mejorar este problema.

Otro de los puntos que no se ha logrado ha sido la de conseguir que el sistema portátil fuese capaz de trabajar en tiempo real. En el siguiente punto se especificarán como trabajos futuros, las soluciones que se han planteado a este problema.

Desde el punto de vista del conocimiento que se ha adquirido durante el desarrollo del trabajo, se puede decir que se ha conseguido establecer relación entre lo aprendido a nivel teórico de la carrera y la realización de un sistema real con unas especificaciones concretas.

A nivel personal, hay que indicar que el trabajar en algo que puede tener una implicación directa y positiva en la vida de las personas da sentido a todo el esfuerzo desarrollado durante la totalidad del proyecto.

4.2. Líneas Futuras

Como se ha visto en el punto anterior, existen objetivos que no se han completado para la realización de este proyecto. Se van a desarrollar 3 puntos principales en los que se va a focalizar las posibles líneas futuras de trabajo.

4.2.1. Aceleración Hardware

Uno de los principales problemas que existen es que no se ha conseguido obtener procesamiento a tiempo real. Para esto se han planteado dos alternativas:

4.2.1.1. DSP (Digital Signal Processor)

Es un sistema basado en un procesador o microprocesador que posee un conjunto de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad.

Las aplicaciones más habituales en las que se emplean DSP son el procesado de audio y vídeo; y cualquier otra aplicación que requiera el procesado en tiempo real. Con estas aplicaciones se puede eliminar el eco en las líneas de comunicaciones, lograr hacer más claras imágenes de órganos internos en los equipos de diagnóstico médico, cifrar conversaciones en teléfonos celulares para mantener privacidad, analizar datos sísmicos para encontrar nuevas reservas de petróleo y una larga lista de elementos que pueden ser relacionados con el proceso de señales [38].

Se podría decir que este sistema es un paso intermedio entre el sistema de procesamiento que se ha utilizado durante todo el proyecto y el sistema que se va a plantear ahora (FPGA).

4.2.1.2. FPGA (Field Programmable Gate Array)

Es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de descripción especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip[39].

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando ésta disponga de los recursos necesarios. Las aplicaciones de las FPGA cada vez son más altas debido a que dan una velocidad de procesamiento y una paralelización de los procesos que un microprocesador al uso no puede conseguir.

Esta posible línea de trabajo es muy real ya que se ha contactado con una pequeña StartUp que está actualmente en formación (ALCYONE) especializada en aceleración hardware.

A pesar de cambiar el formato del proyecto, eliminando el microprocesador y sustituyéndolo por una FPGA, todo este trabajo sigue teniendo sentido y está justificado dado que todos los algoritmos desarrollados y los estudios realizados para sistemas de entrada y salida son reutilizables para ser utilizados con una FPGA. Incluso Xilinx, empresa especializada en FPGA, ha realizado adaptaciones de las librerías de OpenCV a sus dispositivos.

El precio utilizando una FPGA es muy similar al de utilizar Raspberry Pi 2. Actualmente se pueden encontrar FPGAs a partir de 20 euros.

4.2.2. Reducción de costes

Como se puede observar en el apartado de presupuesto, la gran limitación en cuanto al precio es el sistema de salida.

Actualmente, las gafas de realidad aumentada tienen un precio muy elevado, y esta limitación es difícil de solventar. Como ya se ha indicado en varias ocasiones durante este trabajo, la tecnología en este aspecto está avanzando muy rápidamente y por tanto es muy probable que se reduzca drásticamente el precio en unos años. Igualmente, como se ha explicado en puntos anteriores, existen gafas de mucho menor coste actualmente en el mercado.

Otra de las líneas de trabajo que se pueden desarrollar es la de valorar la posibilidad de crear unas gafas en el propio grupo que integren las cámaras, batería y sistema de salida.

4.2.3. Simplificar el sistema de control

Como se ha comentado en el apartado de conclusiones, el sistema de control tiene 16 botones, esto choca directamente con los principios del diseño para todo. A continuación se van a mostrar algunas alternativas que se han planteado para mejorar el sistema pero que no se han podido llevar a cabo por falta de tiempo.

Hay un total de 8 botones para controlar movimientos de pantalla, 4 para mover la ventana y otros 4 para desplazar la imagen si se ha dado zoom, dado que estos movimientos no se van a realizar simultáneamente, se puede condensar esos 8 botones a 4. De esta manera, si la imagen no ha sufrido zoom los controles desplazaran la ventana, en caso de que se haya aplicado zoom los controles moverán la imagen.

Tanto para controlar el color de los contornos, como para variar los umbrales, se puede utilizar un solo botón para cada tarea, de esta manera según se vaya pulsando el botón, de control de colores por ejemplo, se ira cambiando el color conmutando entre las 4 opciones (rojo, verde, azul y blanco). En el caso del nivel de contornos seria similar.

Con estos cambios, pasaríamos de 16 botones a 8, además se podría sustituir los dos botones que controlan el zoom por un potenciómetro de rueda. De esta manera se tendrían 6 botones y un potenciómetro de rueda u 8 botones. Estas modificaciones alterarían el programa mostrado en los Anexos.

Otra variación a todo lo comentado anteriormente es el uso de un microprocesador, por ejemplo se podría utilizar un ATMega (Arduino) que gestionará una pequeña pantalla incrustada en la carcasa, y 4 botones que servirán para que el usuario navege a través de un menú creado para poder gestionar todas las variables.

Como todos los pasos realizados en este proyecto, habría que consultar previamente con los usuarios potenciales y con los expertos del IOBA, ya que ellos tienen la mejor visión sobre la comodidad de cada una de las alternativas.

Bibliografía

[1] Núñez Martin, Rubén. “Diseño e implementación de una ayuda electrónica para pacientes con baja visión periférica (II)”

[2] Barranco García, Carlos. “Diseño e implementación de algoritmos de tratamiento de imágenes para ayudas técnicas visuales usando HMDS”

[3] Collado Martínez, Francisco. “Sistema de ayuda a la movilidad para personas con baja visión”

[4] Consejo de Europa,Plan de Acción del Consejo de Europa para la promoción de derechos y la plena participación de las personas con discapacidad en la sociedad: mejorar la calidad de vida de las personas con discapacidad en Europa 2006-2015

[5] Web IOBA http://www.web.ioba.es/
Última vez consultada: 18/6/2015

[6]Web Centro español de baja visión http://www.baja-vision.org/patologias.htm
Última vez consultada: 18/6/2015

[7]Tienda Baja Visión
http://www.tiendabajavision.com/tiendaonline/CatalogoFamBajaVision.aspx
Última vez consultada: 18/6/2015

[8]Web Sociedad Española de Especialistas en Baja Visión
http://www.seebv.com/SEEBV-Sociedad-Espanola-Especialistas-Baja-Vision-ayudas_visuales
Última vez consultada: 18/6/2015

[9] https://en.wikipedia.org/wiki/PlayStation_Eye
Última vez consultada: 18/6/2015

[10] Web fabricante Philips http://www.philips.es/c-p/SPZ5000_00/webcam-para-pc
Última vez consultada: 18/6/2015

[11] Web Fundación Raspberry Pi
https://www.raspberrypi.org/products/camera-module/
Última vez consultada: 18/6/2015

[12] Web Minoru 3D http://www.minoru3d.com/
Última vez consultada: 18/6/2015

[13]Articulo sobre las diferencias de las distintas versiones de Rasapberry Pi. Blog lifehacker.
http://lifehacker.com/the-raspberry-pi-b-adds-more-ports-and-features-consu-1604629726

Última vez consultada: 18/6/2015

[14]Web Banana Pi http://www.bananapi.org/p/product.html
Última vez consultada: 18/6/2015

[15]http://tienda.bricogeek.com/raspberry-pi/665-pcduino-v3.html
Última vez consultada: 18/6/2015

[16]Web Odroid http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu3
Última vez consultada: 18/6/2015

[17]Web oficial empresa Cubierboard
http://cubieboard.org/
Última vez consultada: 18/6/2015

[18]Web empresa UDOO http://www.udoo.org/udoo-neo/
Última vez consultada: 18/6/2015

[19]Blog tecnológico fpaez. http://fpaez.com/sistemas-operativos-para-raspberry-pi/
Última vez consultada: 18/6/2015

[20] Web oficial Raspbian http://www.raspbian.org/RaspbianFAQ
Última vez consultada: 18/6/2015

[21]Web Python https://www.python.org
Última vez consultada: 18/6/2015

[22] Articulo en página Monografías
http://www.monografias.com/trabajos4/lenguajec/lenguajec.shtml
Última vez consultada: 18/6/2015

[23] Web oficial OpenCv http://opencv.org/
Última vez consultada: 18/6/2015

[24]Noticia coche inteligente Universidad Carlos III de Madrid
http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/noticias/coche_inteligente_peatones
Última vez consultada: 18/6/2015

[25] Articulo web carpente sobre detección de bordes
http://carpente.es/archivos/fic/opt/va/Teoria/02%20-%20DeteccionBordes.pdf
Última vez consultada: 18/6/2015

[24] De la Escalera Hueso, Arturo. Apuntes "Tema 4. Segmentación de imágenes". Sistemas de percepción. Universidad Carlos III de Madrid (10/12/2014)

[25] Jiménez Dorado, Javier. “Estudio mercado: Dispositivos realidad virtual”.

[26]Web de la empresa Epson
http://www.epson.es/es/es/viewcon/corporatesite/products/mainunits/specs/12411
Última vez consultada: 18/6/2015

[27] Web Vuzix http://www.vuzix.com/augmented-reality/products_wrap1200dxar/
Última vez consultada: 18/6/2015

[28] Web Vuzix http://www.vuzix.com/augmented-reality/products_star1200xld/
Última vez consultada: 18/6/2015

[29] Web empresa Powe Bankhttps://power-bank.es/
Última vez consultada: 18/6/2015

[30]Articulo de la página de la empresa nVidia sobre la GPU
http://www.nvidia.com/object/what-is-gpu-computing.html
Última vez consultada: 18/6/2015

[31]Web empresa nVidia
http://www.nvidia.com/object/cuda_home_new.html
Última vez consultada: 18/6/2015

[32] Blog Pete Warden
http://petewarden.com/2014/08/07/how-to-optimize-raspberry-pi-code-using-its-gpu/
Última vez consultada: 18/6/2015

[33]Articulo sobre la GPIO de raspberry pi
https://rpiplayground.wordpress.com/2014/05/03/hacking-the-gpu-for-fun-and-profit-pt-1/
Última vez consultada: 18/6/2015

[34] Articulo sobre la GPIO de raspberry pi
http://pi.gadgetoid.com/pinout
Última vez consultada: 18/6/2015

[35]Pagina fundación Raspberry Pi
https://www.raspberrypi.org/learning/quick-reaction-game/worksheet/
Última vez consultada: 18/6/2015

[36]Proyecto UC3M en web goteo
https://goteo.org/project/realidad-aumentada-para-baja-vision
Última vez consultada: 18/6/2015

[37]Web de modelos 3D thingiverse

http://www.thingiverse.com/
Última vez consultada: 18/6/2015

[38]Blog sobre DSP.
http://digiday.com/platforms/wtf-demand-side-platform/
Última vez consultada: 18/6/2015

[39]Articulo de Xilinx sobre FPGAs
http://www.xilinx.com/fpga/
Última vez consultada: 18/6/2015

Anexos

A.1. Código medición velocidad de adquisición (Python)

```
from cv2.cv import *
import time

#Se captura la imagen de la cámara

captura = CaptureFromCAM(0)

#Se establecen las propiedades de la captura, aqui es donde se
van variando las resoluciones para tomar medidas

SetCaptureProperty( captura, CV_CAP_PROP_FRAME_WIDTH,640)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_HEIGHT,480)

#Se crea una ventana

NamedWindow("Fotograma", 1)

#Se abre un documento de texto, si no existe se crea

text = open("datos.txt","w")

#Se inicia la variable de control para el bucle for

i= 0

#Se inicia un bucle que recorrera 5000 ciclos

for i in range(5000):

      #Se mide el tiempo de ejecucion y se guarda en una variable

      start_time = time.clock()

      #Se guarda la captura en una imagen y se muestra

      fotograma = QueryFrame(captura)
      ShowImage("Fotograma", fotograma)
      tecla = WaitKey(1)

      #Se mide el tiempo de ejecucion en ese momento y se resta
      al anterior, sacando asi el tiempo de ejecucion del bucle

      elapsed_time = time.clock() - start_time

      #Se guarda el dato del tiempo en el fichero
      text.write(str(elapsed_time))
      text.write('\n')

#Al finalizar el bucle se cierra el archivo
text.close
```

A.2. Código inicial medición tiempo de procesamiento (Python)

```
from cv2.cv import *
import time

#Se captura la imagen de la cámara
captura = CaptureFromCAM(0)

#Se establecen las propiedades de la captura, aqui es donde se
van variando las resoluciones para tomar medidas
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_WIDTH,640)
SetCaptureProperty( captura, CV_CAP_PROP_FRAME_HEIGHT,480)

#Se crea una ventana
NamedWindow("Fotograma", 1)

#Se abre un documento de texto, si no existe se crea
text = open("datos.txt","w")

#Se inicia la variable de control para el bucle for
i= 0

#Se inicia un bucle que recorrera 5000 ciclos
for i in range(10):

      #Se mide el tiempo de ejecucion y se guarda en una variable
      start_time = time.clock()

      #Se guarda la captura en una imagen y se muestra
      fotograma = QueryFrame(captura)

      #Se crea una imagen del tamaño de la de captura
      bn2 = CreateImage(GetSize(fotograma), IPL_DEPTH_8U, 1)

      #Se convierte la imagen a escala de grises y se usa
      operador Canny
      CvtColor(fotograma, bn2, CV_BGR2GRAY)
      Canny (bn2,bn2, 125,75)

      #Se muestra la imagen
      ShowImage("Fotograma", bn2)
      tecla = WaitKey(1)

      #Se mide el tiempo de ejecucion en ese momento y se resta
      al anterior, sacando asi el tiempo de ejecucion del bucle

      elapsed_time = time.clock() - start_time

      #Se guarda el dato del tiempo en el fichero
      text.write(str(elapsed_time))
      text.write('\n')

#Al finalizar el bucle se cierra el archivo
text.close
```

A.3. Código detección de la zona de visión (C)

```
//Se incluyen las librerias
#include "stdafx.h"
#include <Windows.h>
#include<vector>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;

//Se inicializan las variables de los vertices para los ocho rectangulos
int inix[8], iniy[8],finx[8], finy[8], area [8];
int control = 0, a=0, IndicadorDeProceso = 0, G=0;
Scalar Imagen1 = cvScalar (255,0,0,0);
Scalar Imagen2 = cvScalar (255,0,0,0);
Scalar Imagen3 = cvScalar (255,0,0,0);
Scalar Imagen4 = cvScalar (255,0,0,0);
Scalar Imagen5 = cvScalar (255,0,0,0);
Scalar Imagen6 = cvScalar (255,0,0,0);
Scalar Imagen7 = cvScalar (255,0,0,0);
Scalar Imagen8 = cvScalar (255,0,0,0);
Scalar ImagenTest = cvScalar (0,0,255,0);
Scalar ImagenTestConfirm = cvScalar (0,255,0,0);
int InicioX, InicioY, FinX, FinY;
int TamañoRectanguloTest = 2, TamañoRectanguloEstatico = 1,
TamañoRectanguloPulsado = 2;

//Funcion que gestiona la posicion del raton
void CallBackFunc(int event, int x, int y, int flags, void* userdata)
{
	//Primer paso del sistema
	if (IndicadorDeProceso == 0)
	{
	if (G==0)
	{
		IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
		cvShowImage("Original", img);
		G++;
	}
	//Si se pulsa boton derecho se cancela el rectangulo
	if ( event == EVENT_RBUTTONDOWN && a<8)
	 {
	    control = 0;
		inix[a]=0;
		iniy[a]=0;
		finx[a]=0;
		finy[a]=0;
		IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
		cvShowImage("Original", img);
	}
	//Si se pulsa boton izquierdo
	if ( event == EVENT_LBUTTONDOWN && a<8)
	 {
		 //Si no se ha pulsado antes se establece el primer vertice
		 if (control == 0)
		 {
		 control =1;
		 inix[a] = x;
		 iniy[a] = y;
```

```
                }
                //Si ya ha sido pulsado se establece el segundo vertice
                else if (control == 1)
                {
                control =2;
                finx[a] = x;
                finy[a] = y;
                IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                cvRectangle(img,cvPoint(inix[a],iniy[a]), cvPoint(finx[a],
finy[a]),ImagenTestConfirm,TamañoRectanguloTest,4,0);
                cvShowImage("Original", img);
                //Se vuelve al estado inicial para dibujar el siguiente rectangulo
                }else if (control == 2)
                {
                a++;
                IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                cvShowImage("Original", img);
                control = 0;
                //Si ya se han dibujado los 8 rectangulos se borra la ventana
                if ( a == 8)
                {
                        cvDestroyWindow ("Original");
                }}
        }
        //Si se a pinchado el primer vertice, se va dibujando el rectangulo en
funcion del movimiento del raton
        if ( control == 1 && a<8 )
        {
                finx[a] = x;
                finy[a] = y;

                IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                cvRectangle(img,cvPoint(inix[a],iniy[a]), cvPoint(finx[a],
finy[a]),ImagenTest,TamañoRectanguloTest,4,0);
                cvShowImage("Original", img);
        }
        }
        //Segundo paso del sistema
        else if(IndicadorDeProceso == 1)
        {
               //Se van mostrando las imagenes
               IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
               cvSetWindowProperty("Original", CV_WND_PROP_FULLSCREEN,
CV_WINDOW_FULLSCREEN);
               cvRectangle(img,cvPoint(inix[a],iniy[a]), cvPoint(finx[a],
finy[a]),ImagenTestConfirm,TamañoRectanguloPulsado,4,0);
               cvShowImage("Original", img);
               //Si se pulsa boton derecho se borran los datos del rectangulo
               if ( event == EVENT_RBUTTONDOWN && a<8)
         {
            inix[a]=0;
               iniy[a]=0;
               finx[a]=0;
               finy[a]=0;
               IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
               cvShowImage("Original", img);
               a++;
        }
               //Si se pulsa boton izquierdo se pasa al siguiente rectangulo
               if ( event == EVENT_LBUTTONDOWN && a<8)
                      a++;
               //Si ya se han revisado los 8 rectangulos se borra la ventana
```

```
		if (a==8)
			{
				cvDestroyWindow ("Original");
		}}
	//Tercer paso del sistema
	else if (IndicadorDeProceso == 2)
	{
		// Se muestran todos los rectangulos
		int Xraton = x;
		int Yraton = y;
		IplImage *img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
		cvSetWindowProperty("Original", CV_WND_PROP_FULLSCREEN,
CV_WINDOW_FULLSCREEN);
		cvRectangle(img,cvPoint(inix[0],iniy[0]), cvPoint(finx[0],
finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[1],iniy[1]), cvPoint(finx[1],
finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[2],iniy[2]), cvPoint(finx[2],
finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[3],iniy[3]), cvPoint(finx[3],
finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[4],iniy[4]), cvPoint(finx[4],
finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[5],iniy[5]), cvPoint(finx[5],
finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[6],iniy[6]), cvPoint(finx[6],
finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
		cvRectangle(img,cvPoint(inix[7],iniy[7]), cvPoint(finx[7],
finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
		cvShowImage("Original", img);

		if (  event == EVENT_RBUTTONDOWN)
			{
				cvDestroyWindow ("Original");
				IndicadorDeProceso = 3;}
		//A partir de aqui se gestiona si el raton se encuentra sobre cada
uno de los rectangulos, al colocar el raton sobre uno de ellos, se cambia el
tamaño del rectangulo
		// PRIMER RECTANGULO
		 if ((Xraton > inix[0] && Xraton < finx[0] && Yraton == iniy[0])||
(Xraton > inix[0] && Xraton < finx[0] && Yraton == finy[0]) || (Yraton > iniy[0]
&& Yraton < finy[0] && Xraton == inix[0])|| (Yraton > iniy[0] && Yraton < finy[0]
&& Xraton == finx[0]) )
			{
				IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
				cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloPulsado,4,0);
				cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
				cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
```

```
                        cvShowImage("Original", img);
                        if (event == EVENT_LBUTTONDOWN)
                        {
                                inix[0] = 0;
                                iniy[0] = 0;
                                finx[0] = 0;
                                finy[0] = 0;
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                        }
                }

                //SEGUNDO RECTANGULO

                if ((Xraton > inix[1] && Xraton < finx[1] && Yraton == iniy[1])||
(Xraton > inix[1] && Xraton < finx[1] && Yraton == finy[1]) || (Yraton > iniy[1]
&& Yraton < finy[1] && Xraton == inix[1])|| (Yraton > iniy[1] && Yraton < finy[1]
&& Xraton == finx[1]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloPulsado,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                                if (event == EVENT_LBUTTONDOWN)
                                {
                                        inix[1] = 0;
                                        iniy[1] = 0;
                                        finx[1] = 0;
                                        finy[1] = 0;
                                        IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
```

```
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                        }
                }

                //TERCER RECTANGULO

                if ((Xraton > inix[2] && Xraton < finx[2] && Yraton == iniy[2])||
(Xraton > inix[2] && Xraton < finx[2] && Yraton == finy[2]) || (Yraton > iniy[2]
&& Yraton < finy[2] && Xraton == inix[2])|| (Yraton > iniy[2] && Yraton < finy[2]
&& Xraton == finx[2]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloPulsado,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                                if (event == EVENT_LBUTTONDOWN)
                                {
                                        inix[2] = 0;
                                        iniy[2] = 0;
                                        finx[2] = 0;
                                        finy[2] = 0;
                                        IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                        cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
```

```
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                        }
                }

                //CUARTO RECTANGULO

                 if ((Xraton > inix[3] && Xraton < finx[3] && Yraton == iniy[3])||
(Xraton > inix[3] && Xraton < finx[3] && Yraton == finy[3]) || (Yraton > iniy[3]
&& Yraton < finy[3] && Xraton == inix[3])|| (Yraton > iniy[3] && Yraton < finy[3]
&& Xraton == finx[3]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloPulsado,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                                if (event == EVENT_LBUTTONDOWN)
                                {
                                        inix[3] = 0;
                                        iniy[3] = 0;
                                        finx[3] = 0;
                                        finy[3] = 0;
                                        IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                        cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                        cvShowImage("Original", img);
                                }

                }

                 //QUINTO RECTANGULO
```

```
                if ((Xraton > inix[4] && Xraton < finx[4] && Yraton ==
iniy[4])|| (Xraton > inix[4] && Xraton < finx[4] && Yraton == finy[4]) || (Yraton
> iniy[4] && Yraton < finy[4] && Xraton == inix[4])|| (Yraton > iniy[4] && Yraton
< finy[4] && Xraton == finx[4]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloPulsado,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                                if (event == EVENT_LBUTTONDOWN)
                                {
                                        inix[4] = 0;
                                        iniy[4] = 0;
                                        finx[4] = 0;
                                        finy[4] = 0;
                                        IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                        cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                        cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                        cvShowImage("Original", img);
                                }
                }

                  //SEXTO RECTANGULO

                   if ((Xraton > inix[5] && Xraton < finx[5] && Yraton ==
iniy[5])|| (Xraton > inix[5] && Xraton < finx[5] && Yraton == finy[5]) || (Yraton
> iniy[5] && Yraton < finy[5] && Xraton == inix[5])|| (Yraton > iniy[5] && Yraton
< finy[5] && Xraton == finx[5]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
```

```
                        cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                        cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                        cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                        cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                        cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloPulsado,4,0);
                        cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                        cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                        cvShowImage("Original", img);
                        if (event == EVENT_LBUTTONDOWN)
                        {
                                inix[5] = 0;
                                iniy[5] = 0;
                                finx[5] = 0;
                                finy[5] = 0;
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                        }
                }

                        //SEPTIMO RECTANGULO

                         if ((Xraton > inix[6] && Xraton < finx[6] && Yraton ==
iniy[6])|| (Xraton > inix[6] && Xraton < finx[6] && Yraton == finy[6]) || (Yraton
> iniy[6] && Yraton < finy[6] && Xraton == inix[6])|| (Yraton > iniy[6] && Yraton
< finy[6] && Xraton == finx[6]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
```

```
                        cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloPulsado,4,0);
                        cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                        cvShowImage("Original", img);
                        if (event == EVENT_LBUTTONDOWN)
                        {
                                inix[6] = 0;
                                iniy[6] = 0;
                                finx[6] = 0;
                                finy[6] = 0;
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                        }
                }

                        //OCTAVO RECTANGULO

                         if ((Xraton > inix[7] && Xraton < finx[7] && Yraton ==
iniy[7])|| (Xraton > inix[7] && Xraton < finx[7] && Yraton == finy[7]) || (Yraton
> iniy[7] && Yraton < finy[7] && Xraton == inix[7])|| (Yraton > iniy[7] && Yraton
< finy[7] && Xraton == finx[7]) )
                        {
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[7],iniy[7]),
cvPoint(finx[7], finy[7]),Imagen8,TamañoRectanguloPulsado,4,0);
                                cvShowImage("Original", img);
                                if (event == EVENT_LBUTTONDOWN)
                                {
                                        inix[7] = 0;
                                        iniy[7] = 0;
                                        finx[7] = 0;
```

```
                                finy[7] = 0;
                                IplImage
*img=cvLoadImage("imagen.jpg",CV_LOAD_IMAGE_COLOR);
                                cvRectangle(img,cvPoint(inix[0],iniy[0]),
cvPoint(finx[0], finy[0]),Imagen1,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[1],iniy[1]),
cvPoint(finx[1], finy[1]),Imagen2,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[2],iniy[2]),
cvPoint(finx[2], finy[2]),Imagen3,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[3],iniy[3]),
cvPoint(finx[3], finy[3]),Imagen4,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[4],iniy[4]),
cvPoint(finx[4], finy[4]),Imagen5,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[5],iniy[5]),
cvPoint(finx[5], finy[5]),Imagen6,TamañoRectanguloEstatico,4,0);
                                cvRectangle(img,cvPoint(inix[6],iniy[6]),
cvPoint(finx[6], finy[6]),Imagen7,TamañoRectanguloEstatico,4,0);
                                cvShowImage("Original", img);
                            }
                 }
                }
}
//Se define la funcion para ordenar los rectangulo
void OrdenarRectangulos()
{
       //Se recorren todos los rectangulos para calcular como vertice inicial el
vertice superior izquierdo y el final el inferior derecho
       for (int j= 0; j<8; j++)
       {
              if (inix[j] > finx [j])
              {
                     int b = inix[j];
                     inix[j] = finx[j];
                     finx[j] = b;
              }
              if (iniy[j] > finy [j])
              {
                     int b = iniy[j];
                     iniy[j] = finy[j];
                     finy[j] = b;
              }
       }}

int main()
{
       //Primer bucle para el Paso 1
       do
       {
       //Se crea la ventana y la hacemos a pantalla completa
       cvNamedWindow ("Original", CV_WINDOW_NORMAL);
       cvSetWindowProperty("Original", CV_WND_PROP_FULLSCREEN,
CV_WINDOW_FULLSCREEN);
       //Se llama a la funcion que gestiona el movimiento del raton
       setMouseCallback("Original", CallBackFunc, NULL);
       waitKey ();
       }while ( a < 8);
       cout << "Generando rectangulo....." << endl;
       //Se llama a la funcion para ordenar los rectangulos
       OrdenarRectangulos();
       //Se modifica la variable para ir al paso 2
       IndicadorDeProceso = 1;
       //Se establece la variable a=0 para comenzar desde el primer triangulo
```

```
        a=0;
        Sleep (3000);
        //Segundo bucle para el paso 2
        do
        {
        cvNamedWindow ("Original", CV_WINDOW_NORMAL);
        setMouseCallback("Original", CallBackFunc, NULL);
        waitKey ();
        }while (a<8);
        cout << "Procesando datos....." << endl;
        //Se establece variable para pasar al paso 3
        IndicadorDeProceso=2;
        Sleep (3000);
        //Tercer bucle para el paso 3
        do
        {
        cvNamedWindow ("Original", CV_WINDOW_NORMAL);
        setMouseCallback("Original", CallBackFunc, NULL);
        waitKey ();
        }while (IndicadorDeProceso<3);
        int mayor = 0;
        //Bucle para calcular el rectangulo con mayor area
        for (int i = 0; i < 8 ; i++)
        {
               int ancho = finx [i] - inix [i] ;
               int alto = finy [i] - iniy[i];
               area [i] = ancho * alto;
               if (area[i] > area [i-1])
                      mayor = i;
        }
        //Se abre un fichero
        FILE *fp;
        fp = fopen ( "fichero.txt", "w" );
        //Se guardan las vaariables en un archivo de texto
        fprintf(fp, "%d", inix[mayor+1]);
        fprintf(fp, "\n");
        fprintf(fp, "%d", iniy[mayor+1]);
        fprintf(fp, "\n");
        fprintf(fp, "%d", finx[mayor+1] - inix[mayor+1]);
        fprintf(fp, "\n");
        fprintf(fp, "%d", finy[mayor+1] - iniy[mayor+1]);
        fprintf(fp, "\n");
        //Se cierra el fichero
        fclose ( fp );
    return 0;
}
```

A.4. Sistema final para PC (C)

```
//Se inician las librerias
#include "stdafx.h"
#include <Windows.h>
#include <conio.h>
#include<vector>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;


int main()
{
	//Se captura la imagen de la cámara
	CvCapture *capture= cvCaptureFromCAM(0);
	//Si no esta conectada la camara se manda mensaje de error
	if(capture==NULL)
	{
		printf("No se puede encontrar camara");
	}
	//Se abre el archivo con los datos obtenidos del programa de diagnostico
	FILE *fp;
	fp = fopen ( "datos.txt", "r" );

	//Se cargan las variables donde se almacenaran los datos
	char ejex[10], ejey[10], largorect[10], altorect[10];

	//Se obtienen las variables y se convierten a una variable entera a traves
de la funcion atoi
	fgets(ejex,10,fp);
	int xrect = atoi(ejex);
	fgets(ejey,10,fp);
	int yrect = atoi(ejey);
	fgets(largorect,10,fp);
	int largo = atoi(largorect);
	fgets(altorect,10,fp);
	int alto = atoi(altorect);

	//Se cierra el fichero
	fclose ( fp );

	//Se inicializan las variables para los umbrales de la histeresis de la
funcion Canny y las variables para el cambio de color
	Scalar color[4] = {cvScalar (0,255,0,0),cvScalar (255,0,0,0),cvScalar
(0,0,255,0), cvScalar (255,255,255,0) };
	int umbrales[3] = {30, 70, 500};
	int seleccion = 3;
	int x = 0, y=0;

	//Se inicializan las variables para el control del zoom
	int umbral = 0, multiplicador = 0, multiplicador1 = 0, factorx = 0,
factory = 0;

	//Se inicializan las imagenes donde se iran almacenando los diversos
procesos
    Mat canny, imageROI;
	vector<vector<Point> > contornos;
```

```
	vector<Vec4i> hierarchy;

	//Se crea una ventana y se redimensiona en funcion de los valores leidos
en el fichero
	namedWindow("Contornos2",0);
	resizeWindow("Contornos2",largo , alto);

	//Se inicia el bucle para desarrollar el procesamiento de la imagen
    while (true)
    {
		//Se mueve la ventana segun los valores del fichero
	    moveWindow("Contornos2", xrect+x, yrect+y);
		Mat frame;

		// Se guarda en una imagen la captura
	    frame=cvQueryFrame(capture);

		// Si esta no existe se envia mensaje de error
		if(frame.empty()){
			printf("No se puede acceder");
			return -1;
			}

		//Se pasa de BGR a escala de grises
	    cvtColor(frame, canny, CV_BGR2GRAY);

		//Se aplica un suavizado Gaussiano
	    GaussianBlur(canny, canny, Size(7,7), 1.5, 1.5);

		//Se realiza el operador Canny
		Canny(canny, canny, umbrales[umbral], umbrales[umbral]*2, 3);

		//Se buscan los contornos de la imagen ya generada tras aplicar el
operado Canny
		findContours( canny, contornos, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
		Mat drawing = Mat::zeros( canny.size(), CV_8UC3 );

		//Recorremos la imagen y se vuelven a dibujar los contornos en el
color seleccionado
		for( int i = 0; i< contornos.size(); i++ )
		{
			 drawContours( drawing, contornos, i, color[seleccion], 1, 8,
hierarchy, 0, Point() );
		}

		if (multiplicador <= 0 || multiplicador >  drawing.rows ||
multiplicador > drawing.cols)
		{
			imageROI = drawing;
		}

		//Se hace zoom si no se esta tratando en valores estandar
		else if (multiplicador >0 || multiplicador-factory <  drawing.rows
|| multiplicador-factorx < drawing.cols )
		{
			imageROI= drawing(Rect((multiplicador-
factorx),(multiplicador-factory),(drawing.cols-multiplicador-
factorx),(drawing.rows-multiplicador-factory)));
		}

		//Se comprueba si se pulsa alguna tecla
```

```
		int k =  waitKey(1);

		//Dependiendo de la tecla se modifican las variables
		switch (k)
		{
			case 99:
				seleccion = 1;
				break;
			case 118:
				 seleccion = 0;
				 break;
			case 114:
				seleccion = 2;
				break;
			case 98:
				seleccion = 3;
				break;
			case 45:
				if (umbral < 2)
					umbral++;
				break;
			case 43:
				if (umbral > 0)
					umbral--;
				break;
			case 97:
				x= x-10;
				break;
			case 100:
				x=x+10;
				break;
			case 115:
				y=y+10;
				break;
			case 119:
				y=y-10;
				break;
			case 106:
				multiplicador = multiplicador + 10;
				break;
			case 107:
				multiplicador = multiplicador - 10;
				factorx = 0;
				factory =0;
				//Condicion para evitar que la imagen se salga de la
ventana
				if (multiplicador <= 0)
					multiplicador = 0;
				break;
			case 102:
				//Condicion para evitar que la imagen se salga de la
ventana
				if ((multiplicador-factorx)>0 )
					factorx= factorx+10;
				break;
			case 104:
				factorx = factorx-10;
				//Condicion para evitar que la imagen se salga de la
ventana
				if (drawing.cols-factorx-factorx > drawing.cols)
					factorx=factorx+10;
				break;
```

```
                        case 103:
                                factory = factory-10;
                                //Condicion para evitar que la imagen se salga de la
ventana
                                if (drawing.cols-factory-factory > drawing.cols)
                                        factory=factory+10;
                                break;
                        case 116:
                                //Condicion para evitar que la imagen se salga de la
ventana
                                if ( (multiplicador-factory)>0 )
                                        factory=factory+10;
                                break;
                }
                //Si se pulsa escape el programa se acaba
                if (k == 27)
                        break;

                //Se muestra la imagen
                 imshow("Contornos2", imageROI);
        }
    return 0;

}
```

A.5. Sistema final para Raspberry Pi (C)

```
//Se inician las librerias
#include "stdafx.h"
#include <Windows.h>
#include <conio.h>
#include<vector>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;

GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(5, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(6, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(20, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(16, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(12, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(21, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(24, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(13, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(19, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(26, GPIO.IN, GPIO.PUD_UP)

int main()
{
	//Se captura la imagen de la cámara
	CvCapture *capture= cvCaptureFromCAM(0);
	//Si no esta conectada la camara se manda mensaje de error
	if(capture==NULL)
	{
		printf("No se puede encontrar camara");
	}
	//Se abre el archivo con los datos obtenidos del programa de diagnostico
	FILE *fp;
	fp = fopen ( "datos.txt", "r" );

	//Se cargan las variables donde se almacenaran los datos
	char ejex[10], ejey[10], largorect[10], altorect[10];

	//Se obtienen las variables y se convierten a una variable entera a traves
de la funcion atoi
	fgets(ejex,10,fp);
	int xrect = atoi(ejex);
	fgets(ejey,10,fp);
	int yrect = atoi(ejey);
	fgets(largorect,10,fp);
	int largo = atoi(largorect);
	fgets(altorect,10,fp);
	int alto = atoi(altorect);

	//Se cierra el fichero
	fclose ( fp );
```

```
	//Se inicializan las variables para los umbrales de la histeresis de la
funcion Canny y las variables para el cambio de color
	Scalar color[4] = {cvScalar (0,255,0,0),cvScalar (255,0,0,0),cvScalar
(0,0,255,0), cvScalar (255,255,255,0) };
	int umbrales[3] = {30, 70, 500};
	int seleccion = 3;
	int x = 0, y=0;

	//Se inicializan las variables para el control del zoom
	int umbral = 0, multiplicador = 0, multiplicador1 = 0, factorx = 0,
factory = 0;

	//Se inicializan las imagenes donde se iran almacenando los diversos
procesos
    Mat canny, imageROI;
	vector<vector<Point> > contornos;
	vector<Vec4i> hierarchy;

	//Se crea una ventana y se redimensiona en funcion de los valores leidos
en el fichero
	namedWindow("Contornos2",0);
	resizeWindow("Contornos2",largo , alto);

	//Se inicia el bucle para desarrollar el procesamiento de la imagen
    while (true)
    {
		//Se mueve la ventana segun los valores del fichero
	    moveWindow("Contornos2", xrect+x, yrect+y);
		Mat frame;

		// Se guarda en una imagen la captura
	    frame=cvQueryFrame(capture);

		// Si esta no existe se envia mensaje de error
		if(frame.empty()){
			printf("No se puede acceder");
			return -1;
			}

		//Se pasa de BGR a escala de grises
	    cvtColor(frame, canny, CV_BGR2GRAY);

		//Se aplica un suavizado Gaussiano
	    GaussianBlur(canny, canny, Size(7,7), 1.5, 1.5);

		//Se realiza el operador Canny
		Canny(canny, canny, umbrales[umbral], umbrales[umbral]*2, 3);

		//Se buscan los contornos de la imagen ya generada tras aplicar el
operado Canny
		findContours( canny, contornos, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
		Mat drawing = Mat::zeros( canny.size(), CV_8UC3 );

		//Recorremos la imagen y se vuelven a dibujar los contornos en el
color seleccionado
		for( int i = 0; i< contornos.size(); i++ )
		{
			drawContours( drawing, contornos, i, color[seleccion], 1, 8,
hierarchy, 0, Point() );
		}
```

```
            if (multiplicador <= 0 || multiplicador >  drawing.rows ||
multiplicador > drawing.cols)
            {
                  imageROI = drawing;
            }

            //Se hace zoom si no se esta tratando en valores estandar
            else if (multiplicador >0 || multiplicador-factory <  drawing.rows
|| multiplicador-factorx < drawing.cols )
            {
                  imageROI= drawing(Rect((multiplicador-
factorx),(multiplicador-factory),(drawing.cols-multiplicador-
factorx),(drawing.rows-multiplicador-factory)));
            }

            //Dependiendo de si el boton esta pulsado se modifican las
variables
            if (GPIO.input(19) == true )
                        seleccion = 1;
            if (GPIO.input(18) == true )
                         seleccion = 0;
            if (GPIO.input(26) == true )
                        seleccion = 2;
            if (GPIO.input(13) == true )
                        seleccion = 3;
            if (GPIO.input(6) == true )
            {           if (umbral < 2)
                              umbral++;
            }
            if (GPIO.input(5) == true )
            {
                        if (umbral > 0)
                              umbral--;
            }
            if (GPIO.input(17) == true )
                        x= x-10;
            if (GPIO.input(22) == true )
                        x=x+10;
            if (GPIO.input(27) == true )
                        y=y+10;
            if (GPIO.input(4) == true )
                        y=y-10;
            if (GPIO.input(24) == true )
                        multiplicador = multiplicador + 10;

            if (GPIO.input(23) == true )
            {
                        multiplicador = multiplicador - 10;
                        factorx = 0;
                        factory =0;
                        //Condicion para evitar que la imagen se salga de la
ventana
                        if (multiplicador <= 0)
                              multiplicador = 0;
            }
            if (GPIO.input(21) == true )
            {
                        //Condicion para evitar que la imagen se salga de la
ventana
                        if ((multiplicador-factorx)>0 )
                              factorx= factorx+10;
            }
```

```
                if (GPIO.input(12) == true )
                {
                                factorx = factorx-10;
                                //Condicion para evitar que la imagen se salga de la
ventana
                                if (drawing.cols-factorx-factorx > drawing.cols)
                                        factorx=factorx+10;
                }
                if (GPIO.input(16) == true )
                {
                                factory = factory-10;
                                //Condicion para evitar que la imagen se salga de la
ventana
                                if (drawing.cols-factory-factory > drawing.cols)
                                        factory=factory+10;
                }
                if (GPIO.input(20) == true )
                {
                                //Condicion para evitar que la imagen se salga de la
ventana
                                if ( (multiplicador-factory)>0 )
                                        factory=factory+10;
                }
                //Se muestra la imagen
                 imshow("Contornos2", imageROI);
        }
    return 0;

}
```

A.6. Manual de Usuario

MANUAL DE USUARIO

Antes de comenzar este manual el usuario debe pasar previamente por un oftalmólogo especializado, el cual le ayudará a realizar la configuración del dispositivo tras una serie de pruebas.

1. Montaje del sistema

1.1. Elementos

Primero hay que revisar que se tienen todos los elementos.

1. Raspberry Pi(Fig. 66)

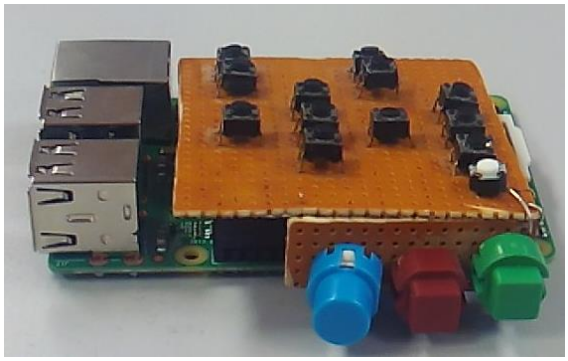


Figura 66. Raspberry Pi con botones

2. Batería (Fig.67)

Figura 67. Bateria portatil

3. Gafas con conversor HDMI a VGA (Fig. 68)

Figura 68. Gafas con camara y adaptador

4. Tarjeta microSD previamente configurada con el especialista (Fig. 69).

Figura 69. Tarjeta microSD

1.2. Montaje

Conecte la tarjeta microSD en la ranura correspondiente de la Raspberry Pi (Fig. 70).

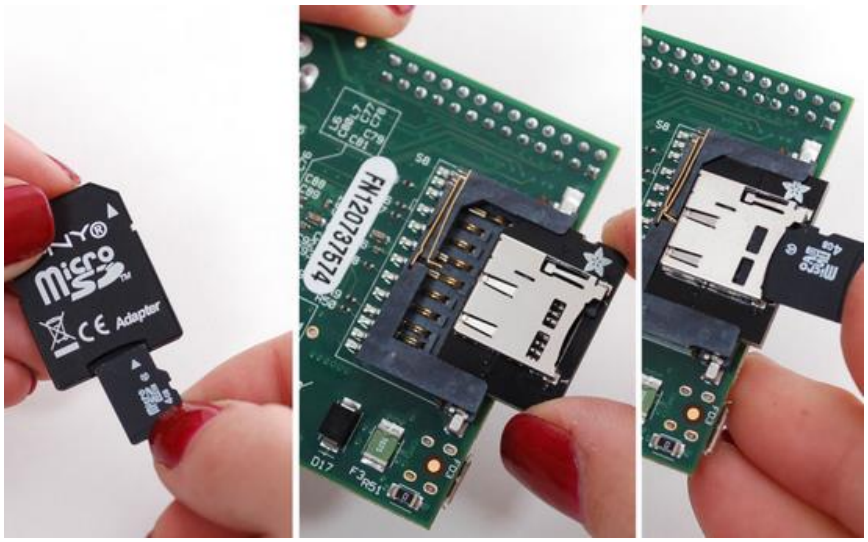


Figura 70. Montaje SD

Cerciórese de que la batería está cargada y posteriormente enchúfela en la clavija micro-USB (Fig. 71).

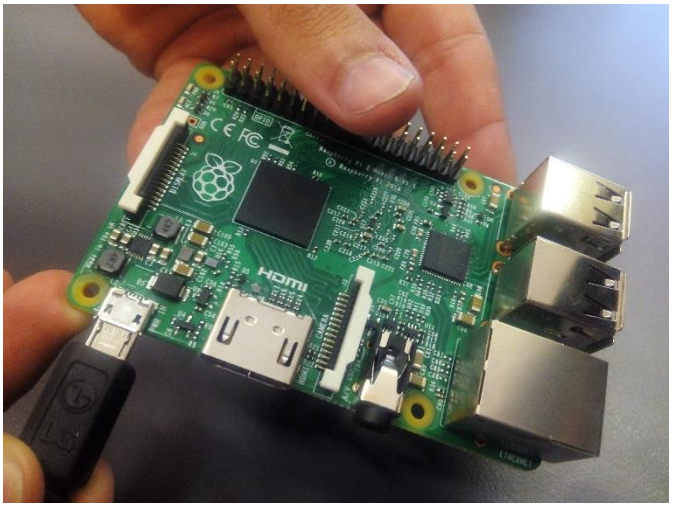


Figura 71. Conexion alimentacion

Al conectarlo se encenderán los dos leds de alimentación de la Raspberry. En caso de no ser así, compruebe que los cables estén bien conectados y vuelva a revisar si la batería se encuentra cargada.

Por último, conecte los tres cables que tienen incorporada las gafas.

- Conecte al puerto HDMI el conector correspondiente(Fig. 72) (Previamente conectado al conversor VGA-HDMI)

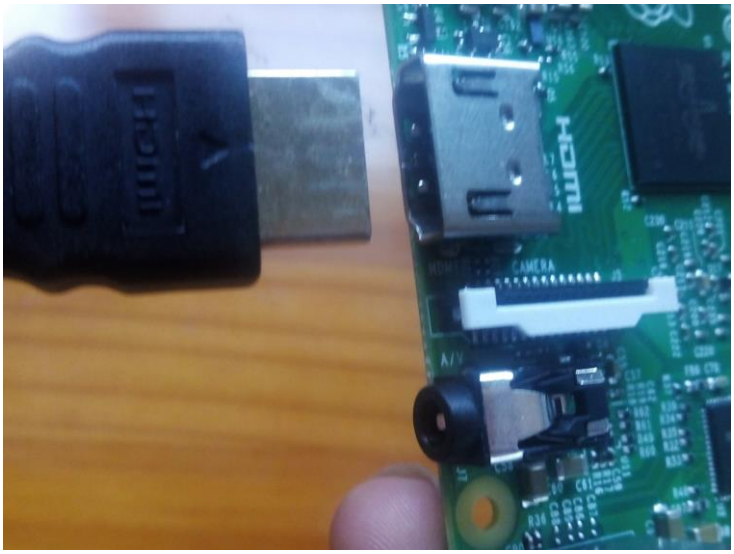


Figura 72. Conexion HDM

- Conecte los dos USB (FIG. 73), el de la cámara y el de alimentación.

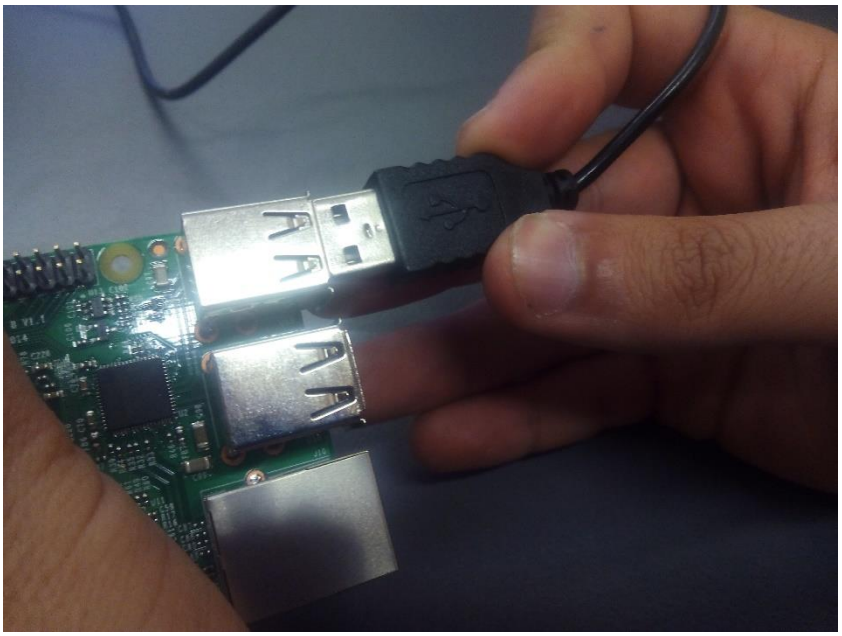


Figura 73. Conexionado USB

Una vez realizadas todas estas conexiones, ya debería poder ver a través de las gafas una pantalla y el programa cargado. En el siguiente video se puede ver el montaje completo del sistema.

https://www.youtube.com/watch?v=bxur_UdWApA&feature=youtu.be

2. Configuración de los botones

Como podrá observar, el sistema cuenta con un panel con un total de 16 botones (Fig. 74) Estos le servirán para configurar el sistema en función de sus necesidades.

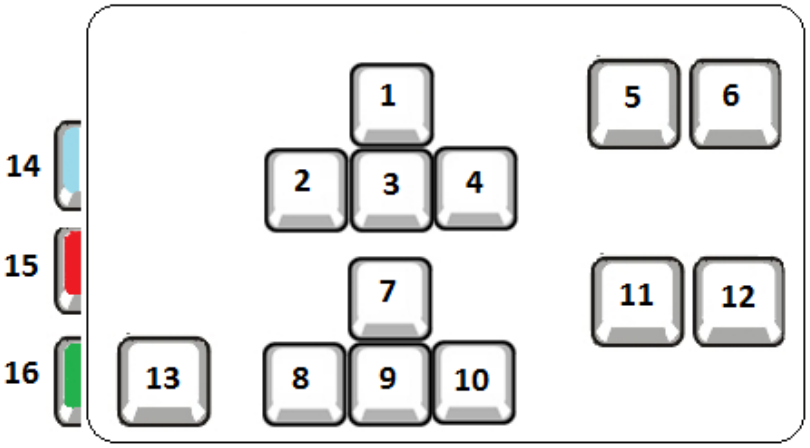


Figura 74. Esquema botones

Boton	Función	Boton	Función
1	Función	9	Desplazar la imagen hacia abajo
2	Desplazar la ventana hacia arriba	10	Desplazar la imagen hacia la derecha
3	Desplazar la ventana hacia la izquierda	11	Hacer Zoom
4	Desplazar la ventana hacia abajo	12	Deshacer Zoom
5	Desplazar la ventana hacia la derecha	13	Contornos en color blanco
6	Aumentar el nivel de los contornos	14	Contornos en color azul
7	Disminuir el nivel de los contronos	15	Contornos en color rojo
8	Desplazar la imagen hacia arriba	16	Contornos en color verde

En la parte superior se encuentran 6 botones, los 4 centrales se encargan de controlar el desplazamiento de la pantalla. Esto permite ajustar la ventana dentro de su campo de visión. Tenga en cuenta que debido a las limitaciones de la pantalla, el desplazamiento de la ventana puede implicar perdida de información. Los otros dos botones situados a la derecha se encargarán de controlar el grado de contornos que se muestran, es decir, si se encuentra en interior y desea mayor detalle, deberá subir el nivel de contornos al máximo.

En la parte inferior se encuentran 6 botones. Los dos botones situados a la derecha se encargan de hacer y quitar zoom, esto le permitirá ajustar la imagen a la pantalla. Los otros 4 botones permitirán desplazar la imagen, esto le servirá cuando haga zoom pero quiere que la pantalla se centre en otra zona de la que se ha seleccionado.

Finalmente, hay 4 botones, uno en la esquina inferior izquierda que pondrá los contornos de color blanco. Los otros tres, situados en el lateral del sistema, permitirán cambiar el color de los contornos en azul, rojo o verde.

3. Posibles fallos

A continuación, se enumeran posibles errores que le pueden ocurrir al utilizar el sistema, y el procedimiento que ha de seguir para solucionarlos.

3.1. Al conectarlo las gafas no se encienden

Si al conectar todo el sistema, las gafas no se encienden, debe realizar 3 comprobaciones:

- Compruebe que todo se encuentra correctamente conectado.
- Compruebe que llega corriente a la Raspberry Pi y por tanto los leds de alimentación están encendidos.
- Si siguen sin encenderse reinicie la Raspberry Pi.

3.2. Las gafas se encienden pero no se ve la imagen

Si al conectar todo el sistema, las gafas se encienden pero solo ve dos pantallas azules, debe realizar 3 comprobaciones:

- Compruebe que el adaptador HDMI-VGA está correctamente conectado, tanto a la Raspberry Pi como a las gafas.
- Compruebe que la tarjeta SD es la que le proporcionó su oftalmólogo y conéctela de nuevo.
- Compruebe que mientras esta todo conectado la cámara está encendida. Para comprobar esto, debe observar como el contorno de la lente esta iluminado.

3.3. La cámara no se enciende

Las gafas funcionan pero la cámara no se enciende. Las comprobaciones que puede realizar son las siguientes:

- El USB está correctamente conectado, en caso de ser así, pruebe en enchufarlo en otro de los puertos.
- Compruebe que todo se encuentra correctamente conectado.
- Compruebe que llega corriente a la Raspberry Pi y por tanto los leds de alimentación están encendidos.