UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

AUTOMATIZACIÓN DE PROCESOS COMPOSITIVOS PARA LA CREACIÓN DE PIEZAS ARTÍSTICAS MULTIMEDIA

PROYECTO FIN DE CARRERA
INGENIERÍA DE TELECOMUNICACIÓN

Autor: Elvira Burdiel Galende
Tutor: Julio Villena Román

Leganés, Julio 2011

Título: Automatización de procesos compositivos para la creación de piezas artísticas multimedia

Autor: Elvira Burdiel Galende

Tutor: Julio Villena Román

EL TRIBUNAL

Presidente: Antonio de la Oliva

Secretario: Jesús Arias Fisteus

Vocal: Jessica Rivero Espinosa

Realizado el acto de defensa del Proyecto Fin de Carrera el día 15 de Julio de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

AGRADECIMIENTOS

A Daniel Canogar, por haberme brindado la oportunidad de participar en tan interesante proyecto y la confianza depositada en mí.

A Julio Villena, por su apoyo y ayuda, por la apertura con la que recibió la idea del proyecto y por la genial actitud y disposición a la hora de supervisarlo.

Y sobre todo a Diego Mellado, porque sin él nada de esto habría sido posible.

RESUMEN

El presente proyecto fin de carrera aborda el análisis y desarrollo de un sistema software de visión por ordenador aplicado a la instalación de una pieza artística.

El sistema que se ha diseñado detecta la forma y posición de un conjunto de objetos anclados a una pared y genera un vídeo, a proyectar sobre dicha pared mediante un proyector de salida, de forma que se muestren una serie de vídeos coincidiendo de forma precisa con la superficie de los objetos anclados. Estos vídeos deben seguir un patrón de reproducción configurable, consistente en una composición de giros a distintas velocidades y sentidos, y con diversos instantes de inicio y finalización.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Motivación

El presente proyecto fin de carrera aborda el análisis y desarrollo de un sistema software de visión por ordenador aplicado a la instalación de una pieza artística, en colaboración con el estudio del artista Daniel Canogar [L32].

La pieza artística que se desea instalar, de nombre “Spin”, consta de un conjunto de objetos, concretamente DVDs estándar, anclados a una pared en blanco, colocados en distintas posiciones y con diversas inclinaciones. Sobre estos objetos, y coincidiendo con la forma de cada uno de ellos, se desea proyectar un conjunto de vídeos. Cada uno de estos vídeos proyectados debe coincidir de forma precisa con la superficie del DVD correspondiente, y debe tener un patrón propio de reproducción consistente en un conjunto de giros a distintas velocidades y sentidos de giro y con distintos instantes de inicio y finalización.

La instalación de esta pieza de forma manual resulta imprecisa y altamente laboriosa. Es por ello que surge la necesidad de crear una aplicación que facilite la tarea, automatizando parte de la labor de detección y generación de la proyección, y facilitando la conformación de la composición artística.

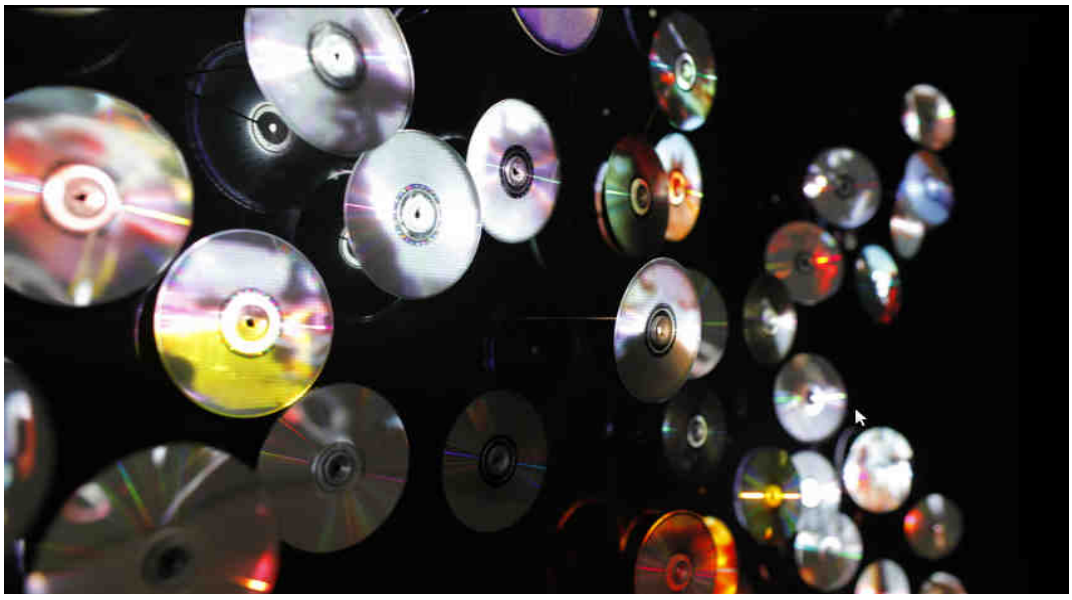

Figura 1.1 – Detalle de la pieza Spin

1.2 Objetivos

El proyecto plantea la aplicación de las nuevas tecnologías, más concretamente las técnicas de visión por ordenador, a algo que hasta hace no mucho estaba alejado del mundo de la ciencia y la tecnología como es el arte. Esto muestra cómo los desarrollos tecnológicos son aplicables a los más diversos fines, y cómo las nuevas técnicas encuentran en el campo del arte un espacio para desarrollarse, innovar y aprovecharse de forma creativa. Así mismo, la tecnología dota al arte de una mayor capacidad y versatilidad, ya que con ella pueden ser creadas piezas antes impensables, abriendo nuevos horizontes al desarrollo artístico y estético.

La pieza artística estaría formada por los siguientes componentes físicos:

- Un conjunto de DVDs estándar, de 12 cm de diámetro, anclados a la pared mediante un alambre rígido que permite la fijación del DVD a unos centímetros de la pared, cada uno de ellos con una inclinación distinta.

- Una webcam o una cámara de fotos, la cual permite captar imágenes de los DVDs para detectar su posición concreta y la superficie que muestran al proyector debida a esa posición.

- Un ordenador con el sistema de software objeto de este proyecto instalado. Este procesará las imágenes captadas por la cámara para obtener la posición y superficie mostrada de los DVDs y generar una proyección de salida con los vídeos a proyectar integrados formando los patrones deseados.

- Un proyector de salida conectado al ordenador, que apunte hacia la pared donde está la pieza y proyecte sobre los DVDs la imagen de los vídeos integrados generada por el software desarrollado, de forma que estos coincidan con la superficie de los DVDs.

A continuación se muestra un croquis cenital del montaje, donde se pueden ver los DVDs anclados a la pared y el proyector.

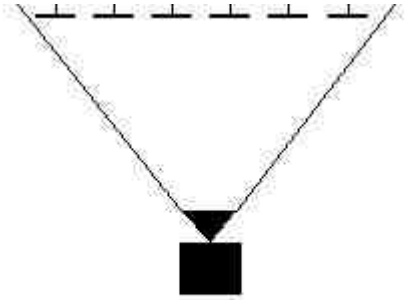

Figura 1.2 – Croquis del montaje

La siguiente figura muestra el aspecto de la pared donde se encuentran anclados los DVDs, mientras se proyectan los vídeos.

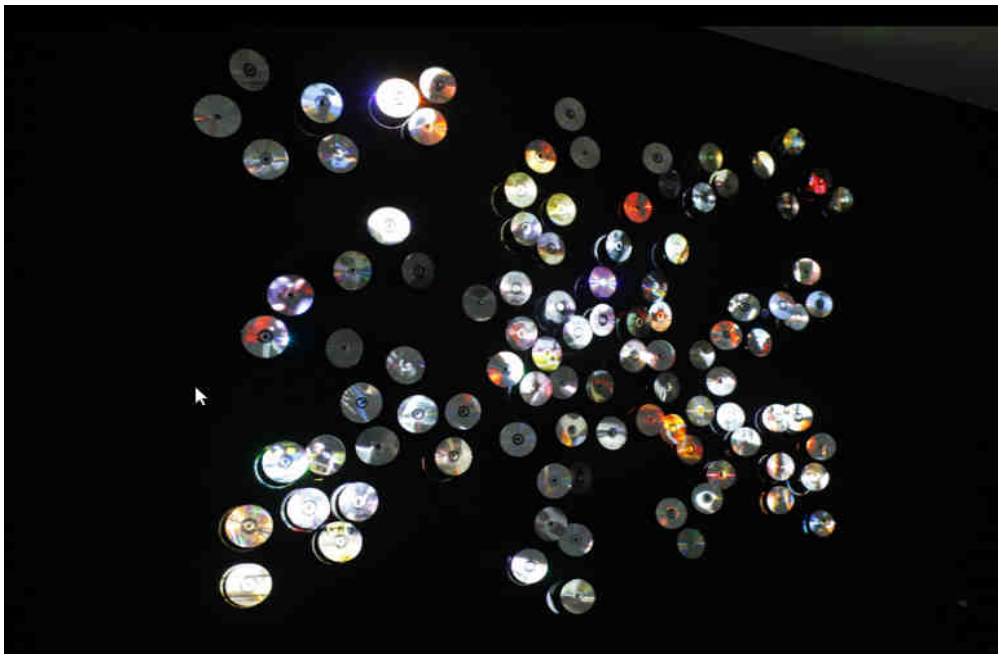

Figura 1.3 – Imagen de la pieza Spin

Gracias a la superficie reflectante de los DVDs y al ángulo en el que se coloca el proyector, la luz de los vídeos proyectados se refleja y muestra un patrón de luces en la pared opuesta, creando el siguiente efecto visual:

Figura 1.4 – Imagen del reflejo de la pieza Spin

Durante el presente proyecto se analizará el problema propuesto y se desarrollará el software de visión por ordenador que permita, a partir de las imágenes captadas por la cámara, la detección precisa de las siluetas de los DVDs y la generación de la imagen a proyectar en la que estén integrados todos los vídeos, que deben coincidir con los DVDs de la pared y seguir un patrón de reproducción configurable.

El software se programará en C++, lenguaje robusto, potente y versátil, para el que existen actualmente varias librerías de aplicación útil a entornos visuales y artísticos. Más concretamente se va a utilizar la librería OpenCV, debido al peso de la visión por ordenador que tiene el proyecto concreto.

OpenCV (Open Source Computer Vision) [Bra08] es una librería de código abierto que contiene más de 500 algoritmos optimizados para análisis de imagen y vídeo. Desde su introducción en 1999, ha sido ampliamente adoptada como la principal herramienta de desarrollo por la comunidad de investigadores y desarrolladores de visión por ordenador.

1.3 Estructura de la memoria

Primeramente se analizará la relación histórica del arte con la ciencia y la tecnología, y la evolución de dicha relación, para centrarse después de forma más concreta a la programación aplicada al arte.

Se estudiarán distintas herramientas comúnmente utilizadas en ámbitos de arte tecnológico, y con mayor detalle OpenCV, opción seleccionada para este proyecto.

A continuación se pasará al diseño del sistema planteado, para después explicar con mayor detalle la implementación de código concreta.

Por último se explicará el modo de empleo de la aplicación, mediante el manual de usuario, finalizando con las conclusiones obtenidas del desarrollo del proyecto y los trabajos futuros que este plantea.

2 ARTE Y TECNOLOGÍA

2.1 Distinción entre arte y ciencia

Los diversos campos de la producción intelectual han cuestionado los argumentos que desde el siglo XVIII se han esgrimido para explicar las diferencias que separan al arte de la ciencia [Sua10]. Los fundamentos de esa distinción son bien conocidos y se han convertido, con el correr del tiempo, en un lugar común socialmente convenido. Como términos en supuesta tensión, la ciencia se asocia con lo frío, lo racional y lo objetivo, mientras que el arte se ubica del lado de lo emotivo, lo irracional y la subjetividad. Estos presupuestos a los que se recurre para establecer las diferencias entre ambos objetos son relativizados desde el momento en que se reconoce que la creatividad, la investigación y la experimentación son aspectos que participan y definen tanto al trabajo científico como al artístico.

Si tenemos en cuenta las definiciones que actualmente se asocian a los términos arte y ciencia, podemos decir que la relación entre ambas esferas es un fenómeno tardío. La conformación de estos campos fue un proceso largo en el que se crearon instituciones especializadas encargadas de definir, regular y legitimar las respectivas prácticas.

Durante el período comprendido entre los siglos XV y XVIII se separaron, progresivamente, las esferas ciencia, arte y literatura, y se definieron, en muchos aspectos, sus límites y rasgos característicos hasta la contemporaneidad. Mientras Leonardo Da Vinci se inscribe en la historia del arte como el artista paradigmático del Renacimiento por reunir los intereses y habilidades disponibles en el horizonte de su tiempo sin discriminar disciplinas o campos de producción, será justamente a partir de entonces que tendrá lugar el proceso de diferenciación y especialización que encuentra su punto más alto en el siglo XIX, con la estabilización de la noción moderna de artista.

Es también en el siglo XV cuando las artes visuales comienzan a formar parte del

conjunto de “artes liberales”, integradas tradicionalmente por el trivium [L1] (gramática, retórica, dialéctica) y el quadrivium [L2] (aritmética, geometría, astronomía, música). Durante la Edad Media, aquellas habían permanecido asociadas a las “artes mecánicas” vinculadas al conjunto de actividades artesanales basadas en la manualidad y en la transmisión de generación en generación a partir de la enseñanza práctica del oficio. Este cambio implicó también la transformación del artesano en artista y, al mismo tiempo, la elevación de su estatus social. El artista ya no era sólo alguien que poseía una capacidad práctica “para hacer”, sino también alguien que estudiaba, que conocía la historia y los textos, en el mismo nivel que poetas y músicos. Esta transformación supuso a la vez un nuevo lugar de producción: la obra de arte adquirió la capacidad de ser documento, ventana abierta al mundo.

Durante el siglo XVII, con la llamada “revolución científica”, se produce la creación de los cuerpos colegiados como la Royal Society en Inglaterra [L3] y la Académie des Sciences en Francia [L4], la Académie Royale de Peinture et de Sculpture [L5], todas durante el siglo XVII, y un siglo después la Royal Academy of Arts [L6]. Estas instituciones van a contribuir a la estabilización de un discurso diferenciado y legitimador de la producción por un lado del arte, y por otro de la ciencia.

Será finalmente durante el siglo XIX cuando se definan las características de lo que conocemos como ciencia moderna y la figura de artista como genio y creador individual, especialmente en relación con el movimiento romántico.

Los vínculos entre ciencia y arte durante ese siglo han sido estudiados observando los puntos de contacto, implícitos o explícitos, en las propuestas de los artistas. El realismo y el impresionismo en pintura, y el naturalismo en literatura, han sido analizados teniendo en cuenta sus consignas de objetividad y fe en los métodos de observación. Así como el naturalismo buscó aplicar los principios de las ciencias experimentales a la descripción de la realidad, el impresionismo elaboró una concepción analítica de la naturaleza. Esta utilización de principios científicos en las producciones artísticas del siglo XIX observará distintos grados de compromiso

y será acompañada y criticada por la historia y la crítica del arte.

Avanzando sobre las primeras décadas del siglo XX, Stephen Wilson [L7] compone una periodización interna que se extiende desde 1870 a 1920 en la que, según señala, arte y ciencia refuerzan la especialización en sus campos desarrollando al mismo tiempo aportes de interés tanto para una esfera como para la otra. Estos avances suponen, en muchos casos, logros paralelos, aunque artistas y científicos no hayan tenido contacto entre sí. Esas coincidencias se explicarían por la convivencia en el marco de un mismo *Zeitgeist*.

La modernidad elaboró propuestas, tanto artísticas como científicas, que evidencian la confianza en la posibilidad humana de conocer y ordenar el mundo desde el punto de vista del sujeto activo. Esta confianza se expresó en numerosos proyectos utópicos en los que la técnica y la tecnología ocupaban un lugar central, como herramientas que aseguraban un avance en todos los campos. Algo se ha modificado en los modos de darse de esta relación, que compromete no sólo a una nueva forma de pensar lo artístico a partir de las novedades introducidas por la neovanguardia, sino también a las representaciones que sobre lo científico habilitó el pensamiento postmoderno.

2.2 El arte y las nuevas tecnologías

A través de la historia, el arte ha servido de espejo, reflejando la imagen del momento y la sociedad en la que la pieza fue creada [Tri06]. Desde algo tan simple como las pinturas humanas primitivas en las paredes de una cueva a algo tan complejo como las actualmente en alza cámaras oscuras digitales, el medio que un artista escoge resulta ser una instantánea del mundo en el que vive tanto como la imagen en sí misma.

Desde el comienzo del siglo XX, los artistas han incorporado la imaginería de las máquinas a su arte [Sha09]. Pero, estuviesen asustados por la amenaza de la

deshumanización y mecanización de la sociedad o satisfechos con las cadenas de montaje o los objetos fabricados por máquinas, sólo tomaron prestadas sus imágenes y su estilo para usarlos en medios de arte tradicional. Fue en los años 60 cuando los artistas empezaron a expandir las fronteras de sus medios con la incorporación de máquinas y procesos tecnológicos al arte. Desde las retransmisiones de radio y las cintas de vídeo a los objetos de funcionamiento mecánico y las imágenes de computación digital, las 4 últimas décadas han visto incrementar el uso de varias tecnologías distintas para hacer arte. En lugar de inspiración visual o temática para pintores o escultores, la tecnología hoy se usa como parte del arte, en servicio al arte y como una forma de arte. Pero estas artes están aún en pañales y, por cada nuevo uso de la tecnología en el arte surge un nuevo conjunto de preguntas prácticas, teóricas y aestéticas en torno a él.

Las nuevas posibilidades surgen cuando los artistas utilizan métodos no tradicionales para expresar una idea [Klu72]. Los artistas han tomado y recontextualizado la tecnología de cada día y lo más destacado de la cultura popular para crear una experiencia activa que deleita los sentidos y donde objetos estáticos cobran vida.

A menudo se utilizan indistintamente como sinónimos términos como “arte digital”, “arte electrónico”, “arte multimedia”, “arte interactivo”, “arte de los nuevos medios” o “arte tecnológico”. Todos ellos se refieren a proyectos que se valen de las tecnologías y los medios de comunicación emergentes.

Gracias a la omnipresencia de ordenadores, teléfonos móviles, consolas e internet, un público más amplio ha cambiado sus viejas reservas contra la tecnología por una casi insaciable curiosidad por todo aquello que sea tecnológico. Ante este trasfondo, nuevas herramientas sin precedentes y posibilidades se han abierto al mundo del arte y el diseño. Los jóvenes artistas están incrementando el uso que se hace de los lenguajes de programación, los sensores y los microprocesadores, así como de las máquinas de rápido prototipado y segmentación 3D. El uso innovador de hardware y software potentes ha hecho asequible y, sobre todo, mucho más

sencillo su uso.

Hoy, el cielo es el límite cuando se trata de ideas para medios experimentales, interfaces no convencionales y experiencias espaciales interactivas. Se juega con las nuevas fronteras de la percepción, interacción y montaje creado por la tecnología actual.

2.3 Arte y programación

Desde 1940, el código se ha desarrollado para ayudar en el trabajo de los campos de ciencia e ingeniería [Rea10]. Seymour Papert [L8], un pionero en investigación sobre ordenadores y creatividad, explica la situación en ese tiempo de este modo:

“El mundo estaba en guerra. Había que realizar cálculos complejos bajo presiones de tiempo que los matemáticos no solían sentir: cálculos numéricos relacionados con el uso y diseño de armas; manipulaciones lógicas para romper códigos cada vez más complejos antes de que la información dejase de ser novedosa... Es poco probable que se parasen a pensar en hacer ordenadores amigables para los usuarios de estilos más suaves que los suyos”.

Las decisiones tomadas sobre ordenadores y lenguajes de programación desde esa época, junto con otros factores, han dificultado la síntesis entre software y arte. Es un hecho desafortunado que muchos lenguajes usados en arte no fuesen originalmente diseñados para esas áreas. Los deseos de software de diseñadores, arquitectos y artistas son a menudo distintos a los de esos científicos, matemáticos e ingenieros. Las habilidades técnicas requeridas para crear formas visuales con la mayoría de los lenguajes predominantes, como C++ y Java, a menudo lleva algunos años adquirirlas.

Una forma alternativa de considerar el código se muestra a través del trabajo de artistas que, en los años 50 y 60, comenzaron a experimentar con software y temas relacionados con software, como la desmaterialización y los sistemas

aestéticos. Estas exploraciones fueron primeramente presentadas al público general en la exhibición Cybernetic Serendipity [L9], que se realizó en el Instituto de Arte Contemporáneo de Londres en 1968, y también en las muestras “Software-Tecnología de la información: Su nuevo significado para el arte” [L13] mantenida en el museo judío de Nueva York en 1970, e “Información”, mantenido en el MOMA [L10] en 1970. El comisario de la exhibición “Software”, Jack Burnham [L11], describió el trabajo como “arte que es transaccional en el sentido de que maneja estructuras de comunicación subyacentes e intercambio de energía”. Entre los trabajos mostrados, el ambicioso “Perfil de visitante” de Hans Haacke [L12] buscó revelar el estatus de élite social de los patrones del museo como una forma de crítica al mundo del arte. Usando una interfaz de ordenador, este tabulaba información personal solicitada al visitante. Les Levine exhibió “Systems Burn-off X Residual Software” [L14] una colección de fotografías que discutían el contexto del software. Levine declaró que las imágenes son hardware y la información sobre las imágenes es software. Escribió la provocativa afirmación: “Todas las actividades que no tienen conexión con objetos o materiales masivos son el resultado de software”. Parecido a la pieza de Levine, muchos de los trabajos incluidos en la exposición “Información” del MoMA estaban caracterizados como “arte conceptual”.

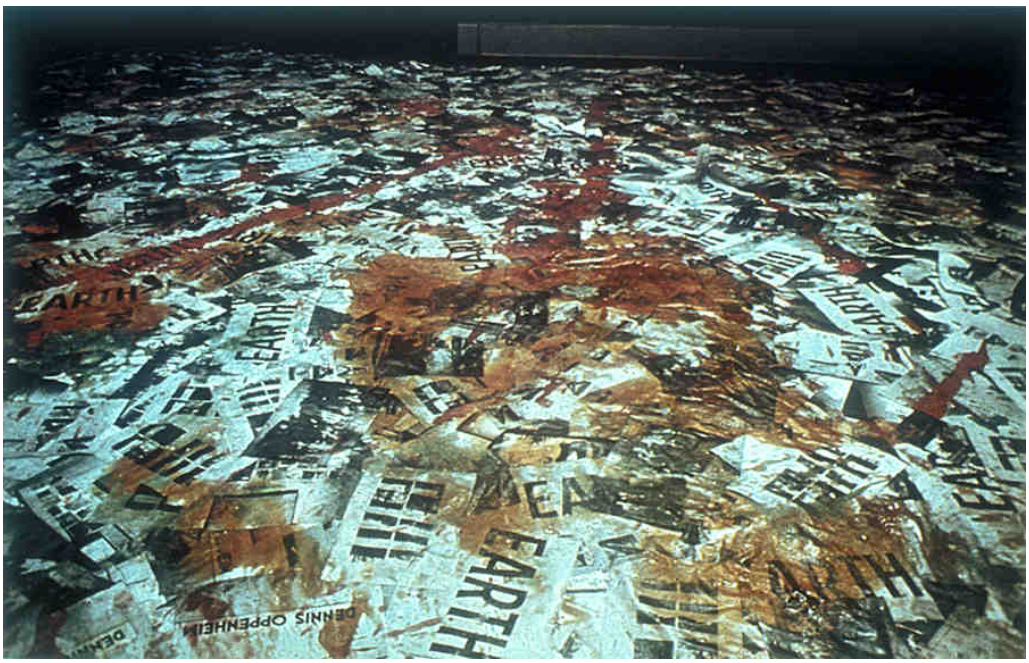

Figura 2.1 – Systems Burn-off X Residual Software, por Les Levine

Al mismo tiempo, artistas y músicos, incluyendo Mel Bochner [L15], John Cage

[L16], Allan Kaprow [L17], Sol LeWitt [L18], Yoko Ono [L19] y La Monte Young [L20], crearon un tipo distinto de arte conceptual y basado en procesos, escribiendo instrucciones y creando diagramas como forma de arte. Por ejemplo, en lugar de realizar dibujos físicos, LeWitt codificó sus ideas como instrucciones que fueron utilizadas para producir dibujos. Él escribió una serie de reglas para definir la tarea de un dibujante, pero las reglas estaban abiertas a interpretación; por lo tanto muchos resultados distintos eran posibles. El trabajo artístico de Ono, como “Cloud Piece”, eran instrucciones para la vida; cada breve texto pedía al lector llevar a cabo acciones como reír, dibujar, sentarse o volar. Como los programadores, todos estos creadores escribieron instrucciones de acciones. Usando el inglés como lenguaje de programación, introdujeron la ambigüedad, la interpretación e incluso la contradicción.

Simultáneamente a estas exploraciones centradas en lo conceptual, los ingenieros estuvieron creando sistemas de programación para la creación de imágenes visuales. En 1963, en los laboratorios Bell, Kenneth C. Knowlton escribió BEFLIX [L21], un programa especializado en animación de construcciones, el cual usó para crear vídeos con los primeros ordenadores en colaboración con artistas como Stan VanDerBeek Lillian F. Schwartz. La película generada por ordenador “Permutations” [L22], fue creada en 1966 por John Whitney Sr. usando GRAF[Hur67], una librería de programación desarrollada por Dr. Jack Citron en IBM. Tanto BEFLIX como GRAF fueron construidos sobre lenguaje Fortran. De estas y otras exploraciones, el desarrollo de los lenguajes de programación escritos expresamente para el arte ha ido ganando velocidad hasta llegar a la actual actividad frenética.

En 1980, la proliferación de los ordenadores personales permitió que la programación llegase a un público más amplio, lo cual llevo al desarrollo de HyperTalk [L23], un lenguaje de programación para la aplicación única de Apple “HyperCard” [L24] (un sistema hipermedia inicial). Los lenguajes relacionados Lingo fueron desarrollados para la primera publicación del Adobe Director [L25] en 1988 (actualmente Macromedia Director y anteriormente MacroMind Director).

Lingo fue el primer lenguaje de programación usado por muchos diseñadores y artistas de la era, llevando al desarrollo de la World Wide Web a comienzos de los 90. Los primeros días de la Web fomentaron la exploración intensiva de la programación gráfica, principalmente canalizada a través del lenguaje ActionScript. El incremento del nivel de conocimientos de programación en las comunidades del arte y la arquitectura ha llevado a la actual proliferación de las opciones de programación.

La influencia del código no está limitada a la pantalla y la imagen proyectada, también se siente en el espacio físico. El código se utiliza para controlar elementos de productos, arquitectura e instalaciones. Se usa para crear archivos de salida, debido a la impresión y creación física a través de máquinas controladas por ordenador, que cortan y ensamblan materiales como madera, metal y plástico. El código se extiende rápidamente más allá de los límites de la pantalla y está comenzando a controlar más aspectos del mundo físico.

La última década ha sido testigo de la proliferación de artistas cuyo medio primario es el software. Procesos algorítmicos que utilizan la programación computacional como medio, permiten a los artistas generar formas visuales cada vez más complejas que de otro modo no se podrían haber imaginado ni mucho menos delineado.

2.4 Principales opciones actuales en programación aplicada al arte

A la hora de utilizar la programación para fines artísticos [Nob09][Bra08], las principales características que se esperan de la tecnología o el lenguaje de programación a utilizar son unas grandes posibilidades gráficas y sonoras, un alto nivel de interactividad física y capacidad de visión por ordenador, así como gran versatilidad y flexibilidad, ya que los fines para los que se va a utilizar pueden ser

muy variados teniendo en cuenta la ancha gama de expresiones artísticas y las frecuentes combinaciones entre estas. Estas características dotarán al artista de las herramientas que le permitan crear una rica experiencia sensorial al observador.

Basándose en estos criterios, las tecnologías de código abierto más destacables que existen actualmente y que son utilizadas por aquellos artistas que incorporan las nuevas tecnologías de programación a sus piezas, son cuatro: Processing, Arduino, OpenFrameworks y OpenCV.

2.4.1 Processing

Es un lenguaje de programación basado en Java y un entorno de desarrollo para aplicaciones en escritorio, web o teléfonos móviles [L26].

Processing fue una de los primeros proyectos de código abierto que fue específicamente diseñado para simplificar la práctica de crear aplicaciones gráficas interactivas para que no-programadores pudieran fácilmente crear obras de arte. Artistas y diseñadores desarrollaron Processing como una alternativa a herramientas de código abierto similares. El código es completamente abierto y libre para bajar, usar y modificar. Casey Reas y Ben Fry comenzaron el proyecto en el MIT bajo la tutela de John Maeda, pero un grupo de desarrolladores lo mantuvieron realizando aplicaciones frecuentes de la aplicación central.

El proyecto Processing incluye un entorno integrado de desarrollo (IDE) que puede ser usado para desarrollar aplicaciones, un lenguaje de programación específicamente diseñado para simplificar la programación de diseño visual, y herramientas de publicación de las aplicaciones en web o en escritorio.

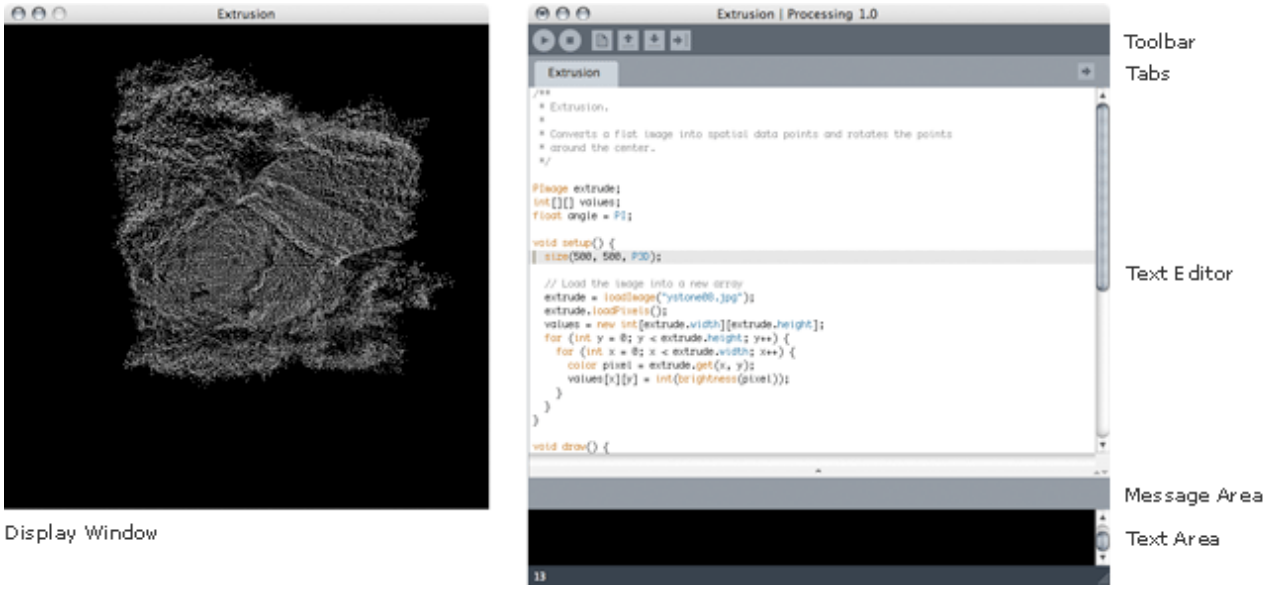

Figura 2.2 – IDE Processing [L26]

Processing es una aplicación Java, es decir, corre sobre una máquina virtual de Java (JVM). Una aplicación Processing que el artista crea es también una aplicación de Java, ya que requiere de una máquina virtual Java sobre la que correr. Se pueden realizar dos aproximaciones a la ejecución de aplicaciones Processing. La primera consiste en ejecutar la aplicación Processing en la Web, lo cual significa poner la aplicación en una página HTML donde un explorador web la encontrará e iniciará la máquina virtual Java para que ejecute la aplicación. La segunda consiste en ejecutar la aplicación Processing en el escritorio, usando la máquina virtual Java que está instalada en la máquina del usuario.

Debido a que Processing está construido en Java y se ejecuta usando Java, puede hacer casi todo lo que Java puede hacer, y aunque Java no puede hacer todo lo que se puede ver en arte y diseño computacional, se acerca bastante. Se puede leer y escribir información en Internet; trabajar con imágenes, vídeo y sonido; dibujar en dos y tres dimensiones; crear inteligencia artificial, realizar simulaciones físicas; y mucho más.

2.4.2 Arduino

Es un sistema que integra una placa base con microprocesador, un IDE y un lenguaje de programación para crear hardware y controles propios [L27].

La palabra Arduino se refiere a las tres herramientas por separado, las cuales, empaquetadas, crean un conjunto de herramientas al cuál se denomina Arduino. En primer lugar, hay un controlador Arduino, existen en diversos formatos y tamaños, de grandes a pequeños pasando por esquemas libremente disponibles que cualquiera con los suficientes conocimientos puede ensamblar por un bajo coste. En segundo lugar, hay un lenguaje y un compilador, el cual crea el código para el microcontrolador y, de forma similar al lenguaje Processing, simplifica las tareas que desafían a diseñadores y desarrolladores cuando trabajan con interacción física y hardware. Finalmente, hay un entorno de programación Arduino que, de nuevo como en el IDE de Processing, es un simple entorno de desarrollo integrado de código abierto en Java.

Figura 2.3 – Controlador Arduino [L27]

El entorno Arduino tiene el objetivo de simplificar la creación de aplicaciones u objetos interactivos mediante la simplificación del lenguaje de programación usado para crear las instrucciones y proveer de un potente a la par que simple controlador que pueda ser usado fácilmente para muchas tareas comunes de programación al

mismo tiempo que sigue siendo lo suficientemente robusto para soportar proyectos más complejos. El controlador Arduino es uno de los proyectos de código abierto más populares y destacables porque es muy potente. Programar microcontroladores puede ser, para los no entrenados, desalentador y frustrante. Aún así, ser capaz de crear computadores que funcionen del tamaño de una caja de cerillas que pueden interactuar fácilmente con hardware abre un nuevo mundo lleno de posibilidades a los artistas y diseñadores interactivos.

Arduino suele ser utilizado para crear controles físicos con los que la gente pueda interactuar, incluidos botones, discos, palancas y tiradores; para entornos interactivos que usen sensores de peso, sensores de ultrasonidos, sensores de infrarrojos y sensores de temperatura; para crear nuevas formas de controlar aplicaciones, controlar la luz de edificios, cámaras de vídeo o cámaras de movimiento; para juguetes mecánicos o pequeña robótica; para programas autónomos que se comunican con otros y les mandan señales, creando una red de miniatura o no tan miniatura. La potencia del controlador, lo que puede hacer y puede permitir a alguien hacer, es enorme.

2.4.3 Openframeworks

Es un framework o marco de programación para artistas y diseñadores que utiliza el potente lenguaje de programación C++ [L28].

Como se puede suponer por su nombre, openFrameworks (oF) es un framework, es decir, una colección de código creado para ayudar a realizar algo en particular. Algunos frameworks están diseñados para trabajar con bases de datos, otros para trabajar con tráfico de internet. Los frameworks no proveen de una herramienta ya compilada en la forma en que Processing y Arduino lo hacen, pero proveen de código que puede ser utilizado para crear programas propios.

De forma especifica, openFrameworks es un framework para artistas y diseñadores

que trabajen con diseño interactivo y arte comunicativo. Está escrito en C++ y para utilizarlo es necesario escribir los programas en C++. Este lenguaje de programación es antiguo y muy potente. Se pueden crear características usando openFrameworks que no sería posible crear en Processing porque el lenguaje subyacente es mucho más flexible y de más bajo nivel. Aunque esto no siempre es necesario, cuando lo es se aprecia la diferencia.

openFrameworks está desarrollado por Zach Lieberman, Theo Watson, Arturo Castro, y Chris O'Shea, junto con la ayuda de sus colaboradores en la Parsons School of Design, MediaLab madrid y Hangar Center for the Arts, así como una dispersa red programadores, artistas e ingenieros. El framework se originó en el programa de diseño y arte mediático de la Parsons School of Design, donde Lieberman estudió. Él se encontró en la creciente necesidad de mayor potencia para sus proyectos y comenzó a aprender C++. Según trabajaba, se dio cuenta de que aunque miles de librerías estaban disponibles para los desarrolladores de C++, ninguna le proveía del mismo tipo de facilidades de uso y baja barrera de iniciación para artistas del que sí proveía Processing para desarrollo en Java. Así nació openFrameworks.

La librería openFrameworks está diseñada como un pegamento de propósito general, y envuelve conjuntamente a varias librerías comúnmente usadas, bajo una ordenada interfaz: openGL para gráficos, rtAudio para entrada y salida de audio, freeType para fuentes, freeImage para entrada y salida de imágenes, quicktime para reproducción y grabación secuencial de video.

Además openFrameworks dispone de un conjunto de addons, que son pequeñas librerías de terceros que pueden ser agregadas al proyecto para extender su funcionalidad. Estos addons sirven para mantener la principal librería de openFrameworks pequeña y manejable, mientras que al mismo tiempo permite la suma de otras funcionalidades y soporte para otras librerías. Los addons actuales son los siguientes:

- OfxDirList, que ayuda a listar el contenido de un directorio.

- OfxXmlSettings, que encapsula tinyXml para ayudar a cargar y guardar xml.
- OfxOsc, que encapsula oscpack para permitir la comunicación con aplicaciones openframeworks usando protocolo OSC.
- OfxNetwork, que contiene un código de red multiplataforma para comunicación UDP y TCP así como multicast.
- OfxThread, que contiene código para manejo de hilos multiplataforma.
- OfxVectorMath, contiene objetos matemáticos vectoriales, como vectores y matrices.
- OfxVectorGraphics, un envoltorio para CreEPS que provee facilidad de uso para salida postscript de openFrameworks.
- Ofx3dModelLoader, un envoltorio que hace posible la carga y muestra de modelos 3ds.
- OfxOpenCv, que encapsula partes de la librería openCV de Intel para funcionalidades de visión por ordenador.

El código está escrito para ser multiplataforma (PC, Mac, Linux, iPhone) y multi-compilador. La API está diseñada para ser mínima y fácil de abarcar. Hay muy pocas clases, y dentro de esas clases, hay muy pocas funciones. El código ha sido implementado para que entre las clases haya mínimos cruces referenciales, haciendo bastante sencilla la extracción, reutilización y expansión, en caso de que fuese necesaria.

El último de los addons mencionados en la lista, encapsula parte de otra librería de C++, openCV, la cual se va a estudiar con detenimiento en el próximo apartado.

2.4.4 OpenCV

OpenCV es una librería de visión por ordenador de código abierto [L29]. La librería está escrita en C y C++ y corre bajo Linux, Windows y Mac OS X. También hay interfaces en desarrollo para Python, Ruby, Matlab y otros lenguajes.

OpenCV fue diseñado para tener una alta eficiencia computacional y con especial atención hacia aplicaciones en tiempo real. Está escrita en C optimizado y puede aprovechar la ventaja de los procesadores multinúcleo. Automáticamente utiliza la librería IPP (Integrated Performance Primitives) apropiada durante la ejecución si dicha librería está instalada.

Uno de los principales objetivos de OpenCV es proveer de una infraestructura de visión por ordenador de uso sencillo que ayude a la gente a construir aplicaciones de visión bastante sofisticadas de forma rápida. La librería OpenCV contiene más de 500 funciones que abarcan muchas áreas de visión, incluida la inspección de productos en fábrica, la imaginería médica, seguridad, interfaces de usuario, calibración de cámaras, visión estéreo y robótica. Debido a que la visión por ordenador y el aprendizaje automático a menudo van juntos, OpenCV también contiene una completa librería de aprendizaje automático de propósito general (Machine Learning Library – MLL). Esta sublibrería está centrada en el reconocimiento y agrupación de patrones estadísticos. La MLL es muy útil en tareas de visión que son la base de la misión de OpenCV, pero es lo suficientemente generalista para ser usada para cualquier tipo de problema de aprendizaje automático.

Son muchos los modos en los que la visión por ordenador puede ser usada hoy en día. Además de los casos más comunes y que vienen fácilmente a la mente de cualquier persona, como la vigilancia, las imágenes y vídeo en la Web o las interfaces de juego, hay otras aplicaciones en las que también resulta de gran utilidad. Por ejemplo, la mayoría de imágenes aéreas y a nivel de calle, como por ejemplo las de Google's Street View [L30], hacen un fuerte uso de la calibración de cámara y las técnicas de unión de imágenes. También se utilizan en vehículos no tripulados, análisis médicos o en los procesos de fabricación: virtualmente todo lo que se produce en masa ha sido automáticamente inspeccionado en algún punto usando visión por ordenador.

La licencia de código abierto de OpenCV ha sido estructurada de forma que se puede construir un producto comercial usando todo o parte de OpenCV. No hay ninguna obligación de que el producto resultado sea de código abierto también o que las mejoras sean devueltas al dominio público. Hay una amplia comunidad de usuarios, la cual incluye importantes empresas como IBM, Microsoft, Intel, SONY, Siemens, y Google, y centros de investigación como Stanford, MIT, CMU, Cambridge e INRIA.

Desde su lanzamiento alfa en Enero de 1999, OpenCV ha sido utilizado en muchas aplicaciones, productos y trabajos de investigación. Estas aplicaciones incluyen la unión de imágenes de satélite y de mapas web, el alineamiento de escaneado de imágenes, la reducción de ruido en imágenes médicas, el análisis de objetos, los sistemas de detección de intrusos y seguridad, los sistemas de monitorización automática, los sistemas de inspección de fabricación, la calibración de cámaras, aplicaciones militares y el manejo de vehículos no tripulados por tierra, mar y aire. Ha sido utilizado incluso en reconocimiento de música y sonido, donde las técnicas de reconocimiento de visión son aplicadas a imágenes de espectrograma del sonido. OpenCV fue una parte clave del sistema de visión del robot “Stanley”, creado en Stanford, el cual ganó el premio DARPA Grand Challenge de carrera de robots en el desierto.

2.4.4.1 Orígenes de OpenCV

OpenCV surgió de una iniciativa de investigación de Intel para aplicaciones avanzadas de CPU intensiva. Para este fin, Intel lanzó muchos proyectos incluyendo seguimiento de rayos en tiempo real y paredes de mostrado 3D. Uno de los autores trabajando para Intel en aquel tiempo estaba visitando universidades y se dio cuenta de que algunos grupos universitarios líderes, como el MIT Media Lab, tenían infraestructuras de visión por ordenador bien desarrolladas y abiertas internamente, código que había pasado de estudiante en estudiante y que daba a cada nuevo estudiante un punto del que empezar muy valioso en el desarrollo de su propia aplicación de visión por ordenador. En lugar de reinventar las funciones básicas desde cero, un nuevo estudiante podía comenzar construyendo encima de

lo que ya existía antes.

Por lo tanto, OpenCV fue concebido como una manera de hacer la infraestructura de visión por ordenador universalmente disponible. Con la ayuda del equipo de librerías de rendimiento de Intel, OpenCV comenzó con un núcleo de código implementado y las especificaciones de algoritmos, siendo enviado a los miembros del equipo ruso de librerías de Intel. Este es el "dónde" de OpenCV: comenzó el laboratorio de investigación de Intel con la colaboración del grupo de Software Performance Libraries junto con la implementación y optimización experta en Rusia.

Uno de los principales miembros del equipo ruso, Vadim Pisarevsky, logró codificar y optimizar la mayor parte de OpenCV y todavía está en el centro de gran parte del esfuerzo de OpenCV. Junto a él, Víctor Eruhimov ayudó a desarrollar la infraestructura inicial, y Valery Kuriakin dirigió el laboratorio ruso y dio un gran apoyo al esfuerzo.

Había varios objetivos para OpenCV desde el principio:

- Investigación de visión avanzada, proporcionando código no sólo libre, sino también optimizado para infraestructuras básicas de la visión. No más reinventar la rueda.

- Difundir el conocimiento de la visión proporcionando una infraestructura común en la que los desarrolladores podrían basarse, de forma que el código sería más sencillo de leer y transferir.

- Aplicaciones comerciales basadas en visión avanzada, haciendo el código portátil y de funcionamiento optimizado de forma gratuita, con una licencia que no requiere que las aplicaciones comerciales se abran o se liberan.

Estos objetivos constituyen el "por qué" de OpenCV. Permitiendo a las aplicaciones de visión por computador se incrementaría la necesidad de procesadores rápidos.

Actualizaciones de drivers a procesadores más rápidos generarían más ingresos para Intel que la venta de software extra. Tal vez por eso este código abierto y libre surgió de un proveedor de hardware en lugar de una compañía de software. En cierto sentido, hay más espacio para ser innovador en software dentro de una empresa de hardware.

En cualquier esfuerzo de código abierto, es importante alcanzar una masa crítica en la que el proyecto se convierte en auto-sostenible. En 2008, momento del que se disponen datos, había habido alrededor de dos millones de descargas de OpenCV, y este número estaba creciendo a un promedio de 26.000 descargas al mes. El grupo de usuarios estaba cerca de los 20.000 miembros [Bra08]. OpenCV recibe muchas contribuciones de los usuarios, y el desarrollo central se ha trasladado mayoritariamente fuera de Intel. En el camino de su desarrollo, OpenCV se vio afectada por el auge y crisis de las puntocom, y también por los numerosos cambios de gestión y dirección. Durante estas fluctuaciones, hubo momentos en que no había nadie de Intel trabajando en OpenCV. Sin embargo, con la llegada de los procesadores multinúcleo y las muchas nuevas aplicaciones de la visión por ordenador, el valor de OpenCV comenzó a subir. Hoy en día, OpenCV es un área activa de desarrollo en varias instituciones, por lo que se espera ver muchas actualizaciones en multicámara, calibración, percepción de profundidad, métodos para la visión mezclada con telémetros láser, y un mejor reconocimiento de patrones, así como un gran apoyo para las necesidades de visión robótica.

2.4.4.2 Estructura y contenido

OpenCV está ampliamente estructurado en cinco componentes principales, cuatro de los cuales se muestran en la siguiente figura.

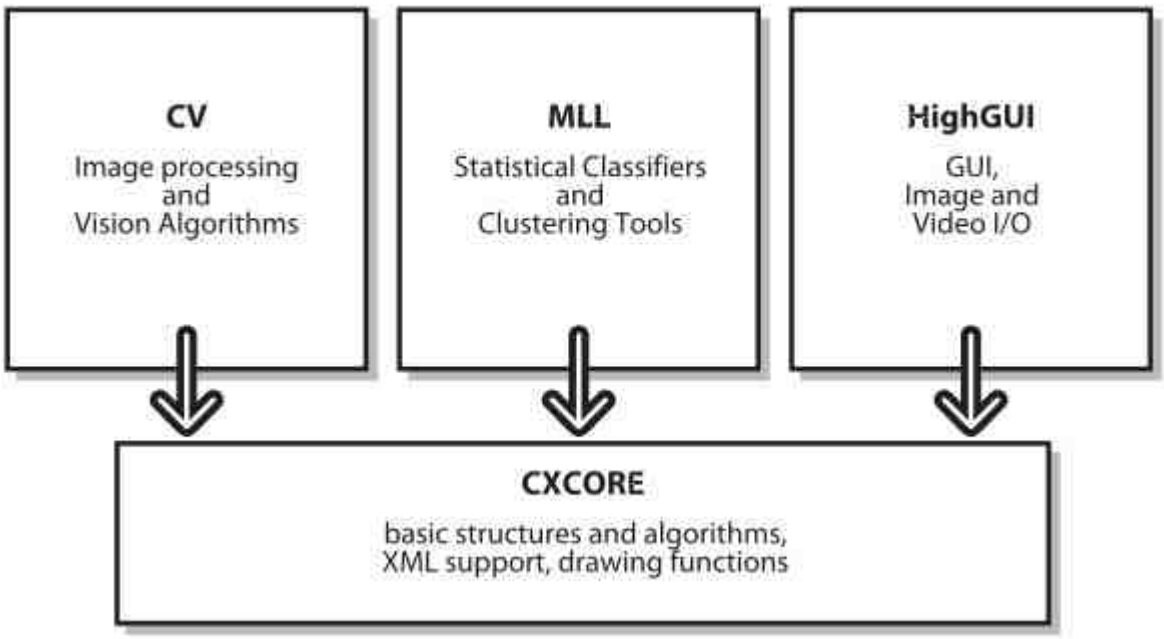

Figura 2.4 – Estructura básica de OpenCV [Bra09]

El componente de CV contiene el procesamiento básico de imágenes y de algoritmos de visión por ordenador de alto nivel.

ML es la biblioteca del aprendizaje automático, que incluye muchos clasificadores estadísticos y herramientas de agrupamiento.

HighGUI contiene rutinas de E / S y funciones para almacenar y cargar vídeos e imágenes.

CXCore contiene el contenido y estructura de la información básica.

La figura no incluye CvAux, que contiene las dos áreas desaparecidas (HMM incrustado de reconocimiento de caras) y algoritmos experimentales (segmentación fondo / primer plano).

OpenCV fue diseñado para ser portátil. Fue escrito originalmente para compilar a través de Borland C++, MSVC++, y los compiladores de Intel. Esto significaba que el código C y C++ tenía que ser bastante estándar a fin de que soporte multiplataforma más fácilmente. De las plataformas que soporta, la arquitectura de

32-bit de Intel (IA32) en Windows es la más madura, seguida por Linux en la misma arquitectura. Mac OS X portátil se convirtió en una prioridad sólo después de que Apple empezó a usar procesadores Intel. Estos son seguidos por soporte de 64 bits en la memoria extendida (EM64T) y la arquitectura de 64 bits de Intel (IA64). La portabilidad es menos madura en el hardware de Sun y otros sistemas operativos.

OpenCV ha sido portado a casi todos los sistemas comerciales, desde PowerPC Macs a perros robot. OpenCV funciona bien en la línea de procesadores de AMD, e incluso las optimizaciones disponibles en el IPP se aprovechan de las extensiones multimedia (MMX) en los procesadores de AMD que incorporan esta tecnología.

3 DISEÑO DEL SISTEMA

Tal y como se señalaba en la introducción, la pieza artística para la que se desea diseñar un sistema de instalación consta de un conjunto de DVDs estándar anclados a una pared en blanco, colocados en distintas posiciones y con diversas inclinaciones. Sobre estos DVDs, y coincidiendo con la forma de cada uno de ellos, se pretende proyectar un conjunto de vídeos. Cada uno de estos vídeos proyectados debe coincidir de forma precisa con la superficie del cd correspondiente, y debe tener un patrón propio de reproducción consistente en un conjunto de giros a distintas velocidades y con distintos instantes de inicio y finalización.

Por tanto este sistema de instalación debe ser capaz de detectar la posición de cada uno de los DVDs con respecto al área sobre la que incide la luz que sale del proyector, y generar el vídeo final de salida que se enviará al proyector. Este vídeo debe estar compuesto por multitud de vídeos que se muestran justo en la zona del vídeo de salida que coincide con la superficie de cada DVD durante la proyección.

El sistema a diseñar e implementar podría dividirse en 4 fases diferenciadas:

1. Calibración de los dispositivos de entrada al sistema
2. Delimitación del área de la pared sobre la que incidirá la imagen salida del proyector
3. Detección de los contornos de los DVDs fijados a la pared
4. Ensamblaje de los vídeos a proyectar sobre la superficie de los DVDs en una único vídeo salida que enviar al proyector.

A continuación se estudiará con detalle cada una de estas fases.

3.1 Calibración de los dispositivos de entrada

Dado que el sistema que se está diseñando se centra en objetivos de visión por ordenador, es necesario que este reciba la información visual necesaria de la escena donde se desea realizar la detección. Esta información visual puede transmitirse de dos formas, mediante vídeo o mediante fotografías.

Por tanto se dispondrá de dos opciones de dispositivos de entrada: una webcam, conectada directamente al ordenador que procesa la información, o una cámara de fotos, con la que se habrán tomado las imágenes previamente y las cuales se habrán transferido al ordenador antes de comenzar a ejecutar el software de detección.

Para la primera opción, el software obtiene una imagen en tiempo real de lo que capta la webcam conectada al ordenador. Para la detección basta con una imagen estática, pero el vídeo obtenido con la webcam permite al usuario variar la iluminación o el ángulo desde el que se están tomando las imágenes de forma dinámica hasta obtener una imagen apropiada que permita la mayor tasa de detección posible.

La segunda opción, aunque más limitada a la hora de actualizar las imágenes de entrada, es útil en los casos en que no sea posible colocar la webcam del ordenador cerca del proyector de salida para obtener las imágenes. Con esta opción, dos imágenes deben ser tomadas previamente y transferidas al sistema para su procesado, una para la detección de la superficie de proyección y otra para la detección de los contornos de los DVDs. Se hablará de ello más detenidamente en próximos apartados.

En ambos casos, los dispositivos de entrada tienen lentes que no son perfectas, y por tanto introducen cierta distorsión en las imágenes tomadas. Para que esta no afecte a la hora de detectar la correcta posición de los DVDs sobre la pared, el primer paso del sistema será anular dicha distorsión calibrando los dispositivos.

Esta calibración permitirá relacionar de forma segura las medidas obtenidas mediante el dispositivo con las medidas reales de los objetos filmados o fotografiados.

Se puede caracterizar el comportamiento de la cámara utilizada a través de dos modelos: un modelo de la geometría de la cámara y un modelo de la distorsión de la lente. El proceso de calibración calcula el valor de los parámetros de los que constan dichos modelos, de forma que el dispositivo de entrada quede totalmente definido. A continuación se analizarán los modelos que se van a utilizar en el sistema:

Modelo de geometría de la cámara

El modelo de cámara más simple es la cámara de agujero de alfiler, en la cual un único rayo de luz entra para cada punto del espacio por el agujero de la cámara, y este único rayo se proyecta sobre la superficie de la imagen. En esta cámara la imagen siempre está enfocada y el tamaño de la imagen proyectada está directamente relacionado con la distancia entre el agujero y la pantalla donde se proyecta la imagen, la cual coincide con la distancia focal. El siguiente esquema muestra la configuración del modelo de la cámara de agujero de alfiler.

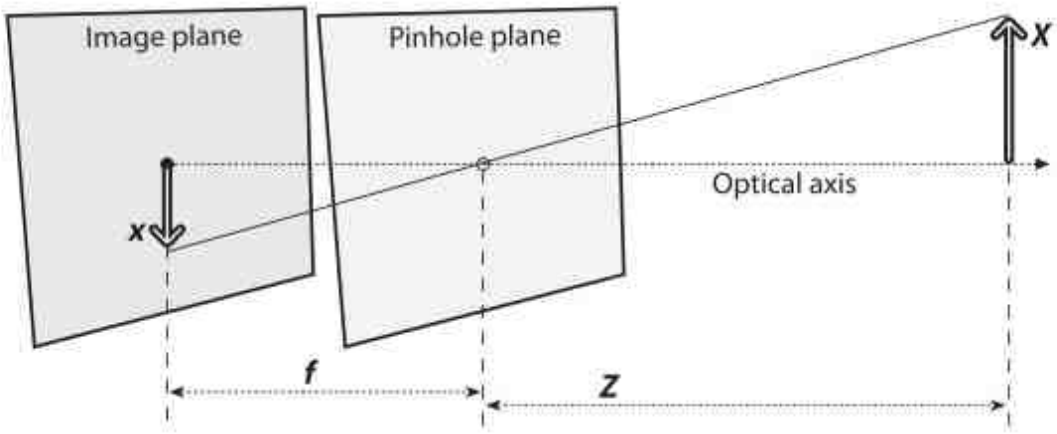

Figura 3.1 Modelo de cámara de agujero de alfiler [Bra08]

Se puede determinar la posición de un punto (x, y) en la imagen proyectada a partir

del punto real (X,Y) mediante las siguientes ecuaciones:

$$x = f_x \cdot \left(X / Z \right)$$
$$y = f_y \cdot \left(Y / Z \right)$$

En estas ecuaciones se distingue entre el factor f_x y el factor f_y, ya que en las cámaras reales cada píxel de la imagen no suele ser cuadrado sino rectangular, por lo que el factor f va a depender del tamaño del elemento individual en esa dirección, siendo $f_x=s_x \cdot F$ y $f_y=s_y \cdot F$, donde F es la distancia focal de la cámara, s_x y s_y los tamaños de dicho elemento individual.

Estas ecuaciones suponen que el punto de intersección entre el plano de la imagen y el eje óptico coincide exactamente con el centro de la imagen, pero esto no suele ser cierto en las cámaras reales, por lo que se debe introducir un factor de desplazamiento que incluya este efecto:

$$x = f_x \cdot \left(X / Z \right) + c_x$$
$$y = f_y \cdot \left(Y / Z \right) + c_y$$

Estos cuatro parámetros (f_x, f_y, c_x, c_y) permiten caracterizar la cámara desde el punto de vista geométrico y constituyen los parámetros intrínsecos de la cámara.

Modelo de distorsión de la lente

En el modelo de cámara definido anteriormente se ha supuesto un único rayo de luz por punto de la imagen, pero esta intensidad no suele ser suficiente por lo que se utilizan lentes para hacer converger múltiples rayos en un mismo punto. Sin embargo estas lentes no son perfectas en el mundo real, por lo que introducen cierta distorsión en la imagen.

Hay dos tipos principales de distorsión debida a las lentes: la radial y la tangencial. La primera se debe a la forma de la lente, mientras que la segunda se debe al

proceso de ensamblaje de la cámara.

Las lentes de cámaras reales con frecuencia distorsionan los píxeles situados cerca de los bordes de la imagen, produciendo una protuberancia que es la fuente del efecto ojo de pez o efecto barril. Esto ocurre debido a que, con algunas lentes, los rayos más alejados del centro se curvan más que los que están más cerca. Este tipo de distorsión, la distorsión radial, es 0 en el centro óptico de la imagen y aumenta según el punto se acerca a la periferia. La siguiente imagen ilustra el efecto.

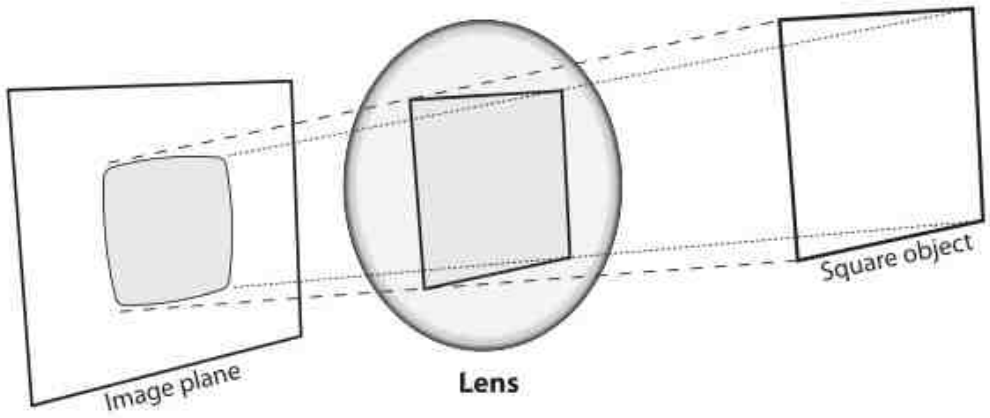

Figura 3.2 – Distorsión radial [Bra08]

Esta distorsión en la práctica es pequeña y se puede caracterizar por los primeros términos de una serie de Taylor expandida alrededor de r=0. Para cámaras baratas, generalmente se usan los dos primeros términos, k1 y k2, mientras que para cámaras con alta distorsión, como lentes de ojo de pez, se utiliza un tercer término k3. En general, la situación radial de un punto de la imagen puede ser reescalado según las siguientes ecuaciones:

$$x_{corregida} = x \cdot \left(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6\right)$$
$$y_{corregida} = y \cdot \left(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6\right)$$

Donde (x, y) es la situación original del punto distorsionado, y ($x_{corregida}$, $y_{corregida}$) es la nueva situación que resulta de la corrección.

El segundo tipo de distorsión más común es la distorsión tangencial. Esta se debe a defectos durante la fabricación que hacen que la lente no esté exactamente paralela al plano de la imagen. Los puntos de la imagen quedan desplazados de forma elíptica en función de la localización y el radio. La siguiente imagen muestra el efecto.

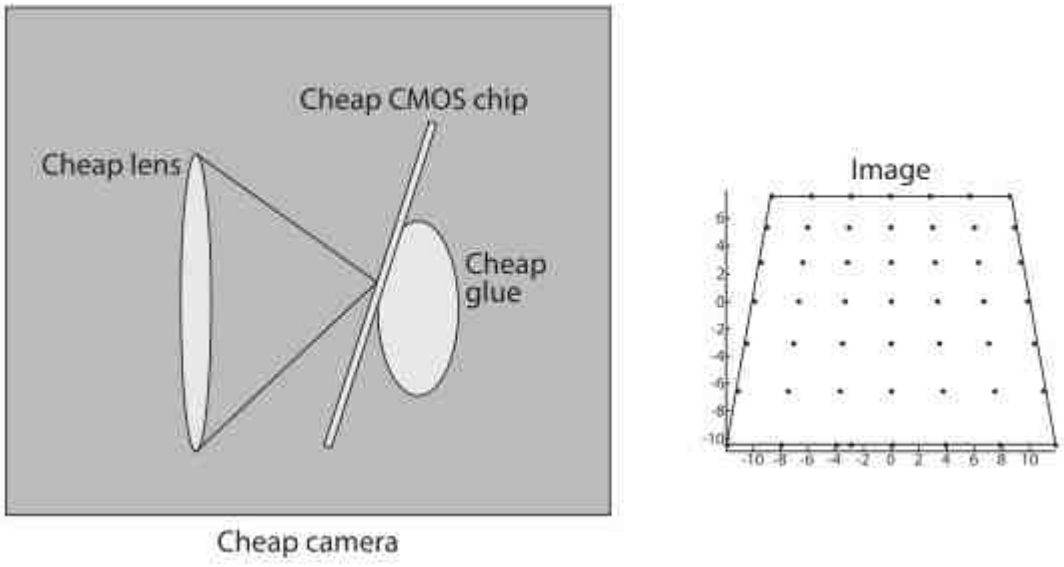

Figura 3.3 – Distorsión tangencial [Bra08]

La distorsión tangencial puede ser caracterizada por dos parámetros, p_1 y p_2, obteniendo las siguientes ecuaciones:

$$x_{corregida} = x \cdot \left(2 \cdot p_1 \cdot y + p_2 \cdot \left(r^2 + 2 \cdot x^2\right)\right)$$
$$y_{corregida} = y \cdot \left(p_1 \cdot \left(r^2 + 2 \cdot y^2\right) + 2 \cdot p_2 \cdot x\right)$$

Por lo tanto, en total se requieren 5 coeficientes de distorsión para definir la lente de un dispositivo de entrada. En OpenCV se suelen empaquetar en un vector (una matriz de 5x1) que contiene a k_1, k_2, p_1, p_2 y k_3 , en ese orden.

Resumiendo ambos modelos, para caracterizar por completo cada uno de los dispositivos de entrada del sistema diseñado se necesitan 4 parámetros intrínsecos (f_x, f_y, c_x, c_y) y 5 parámetros de distorsión (k_1, k_2, p_1, p_2, k_3). Estos parámetros se

obtienen mediante la calibración del dispositivo.

El método de calibración utilizado por las funciones que proporciona OpenCV consiste en captar una estructura conocida que contiene muchos puntos individuales identificables. Viendo esta estructura desde varios ángulos, es posible determinar la localización y orientación relativa de la cámara en el momento de la toma de cada imagen y obtener los parámetros de calibración.

Cualquier objeto apropiadamente caracterizado podría ser usado como objeto de calibración, pero la elección más práctica es un patrón regular, como por ejemplo un tablero de ajedrez. Teóricamente se podría caracterizar una lenta mediante una única toma de imagen de un objeto tridimensional de estructura apropiada, pero OpenCV opta por utilizar múltiples tomas de un objeto plano, como es el caso del tablero de ajedrez, ya que su manejo es mucho más sencillo.

Por tanto se utilizará un patrón de cuadros alternos blancos y negros, del que se tomarán distintas imágenes en distintas perspectivas y con diversos ángulos de rotación. Cuanto mayor sea el número de esquinas que tenga el tablero de ajedrez que se utilice, y mayor sea el número de imágenes tomadas en distintos ángulos y con distintas rotaciones, mejor será la calidad y la robustez de la calibración. En el caso del sistema bajo diseño, se han utilizado las 7 imágenes que se muestran a continuación.

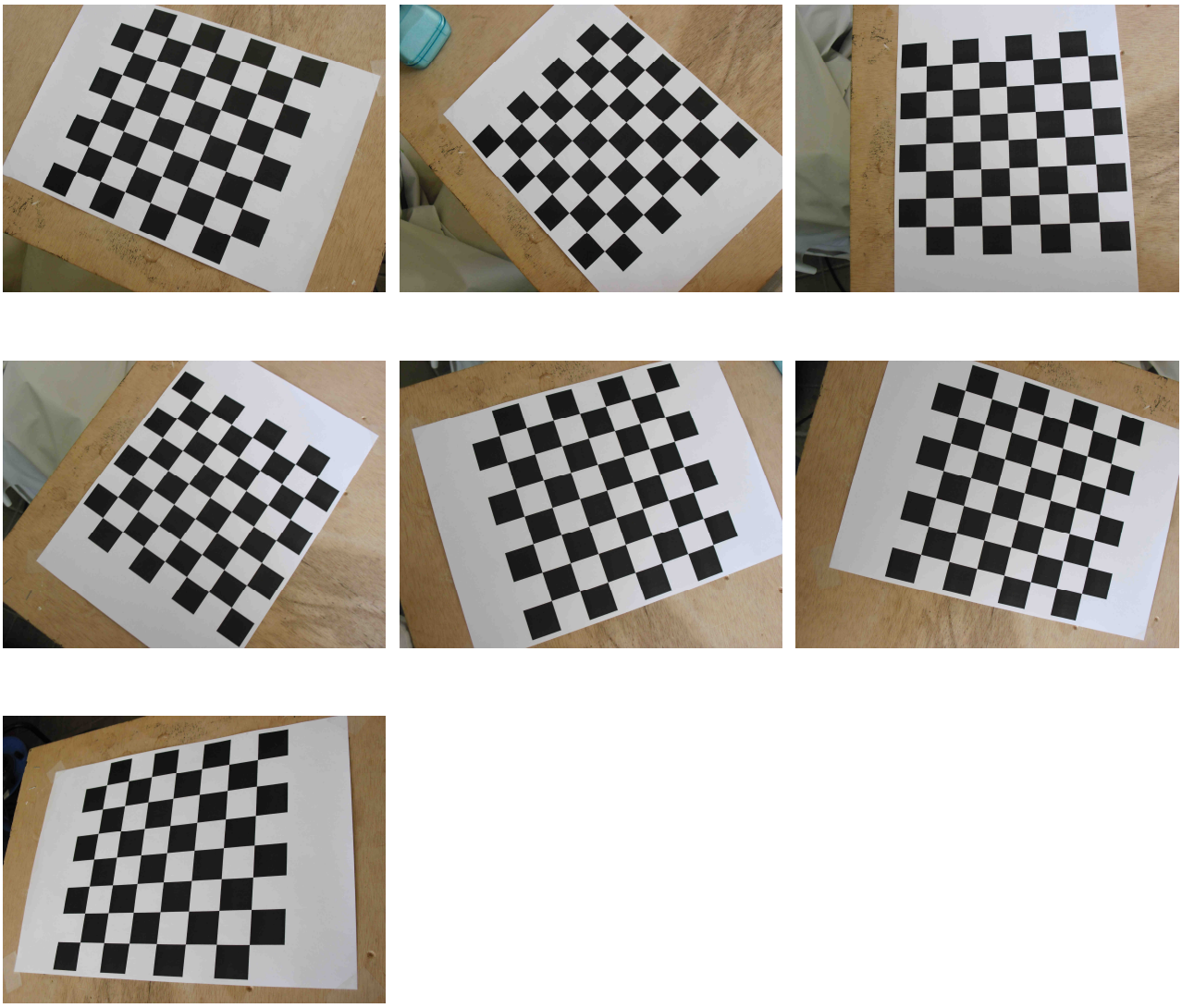

Figura 3.4 – Ejemplo de imágenes de calibración tomadas

OpenCV proporciona una serie de funciones que permiten localizar las esquinas del patrón del tablero de ajedrez y, a partir de esos puntos detectados, determinar los parámetros intrínsecos y de distorsión del dispositivo.

La medida de la distorsión que introduce la lente del dispositivo sólo es necesario realizarla una vez por dispositivo. A partir de esta medida se obtendrán dos archivos xml, *intrinsics.xml* y *distortion.xml*, donde se almacena la información de la cámara y sus defectos, y en cada ejecución del software de detección bastará con invertir esa distorsión basándose en los datos almacenados en dichos archivos.

3.2 Delimitación del área de proyección

El objetivo del sistema es obtener una imagen que, proyectada desde el proyector de salida, coincida con las siluetas de los DVDs. Para conseguir esto es necesario conocer en primer lugar cuál es la zona de la pared, en la cual están colocados los DVDs, sobre la que la luz del proyector incide. De esta forma se podrá conocer la posición relativa de los DVDs en la imagen proyectada y por tanto será posible hacer coincidir la zona de proyección de los vídeos con la superficie de estos.

Para facilitar esta delimitación, inicialmente se proyectará una imagen completamente en blanco con el proyector. El área donde se recae dicha imagen debe ser detectada por el sistema. A continuación se muestra un ejemplo de la imagen captada con la luz blanca proyectada sobre la pared.

Figura 3.5 – Ejemplo de imagen inicial para detección de perspectiva

El dispositivo utilizado para captar la imagen se ha calibrado previamente, tal y como se estudió en el apartado anterior. Por tanto se dispone de los parámetros

que caracterizan a la cámara, gracias a los cuales se eliminará en primer lugar la distorsión de la imagen captada.

Dado que lo que se desea detectar en esta fase es el contorno de proyección de luz, la información del color no es necesaria. De hecho, dado que la luz proyectada es blanca y la habitación donde se realiza la proyección estará a oscuras, en el límite entre el área de proyección y el resto de la pared se encontrará el mayor contraste de intensidad lumínica. Por ello se aplicará un filtro de binarización blanco/negro, con el que se obtendrá una imagen bicolor en la que será más sencillo detectar límites de contornos. El aplicar dicho filtro consiste en evaluar el nivel de intensidad de cada píxel de la imagen y, comparándolo con un umbral prefijado, convertir dicho píxel a negro si está por debajo del umbral o a blanco si está por encima. El umbral de este filtro se podrá modificar dinámicamente en tiempo de ejecución, ya que la imagen de entrada puede tener distintos niveles de claridad. A continuación se muestra un ejemplo de como quedaría la imagen tras ser filtrada.

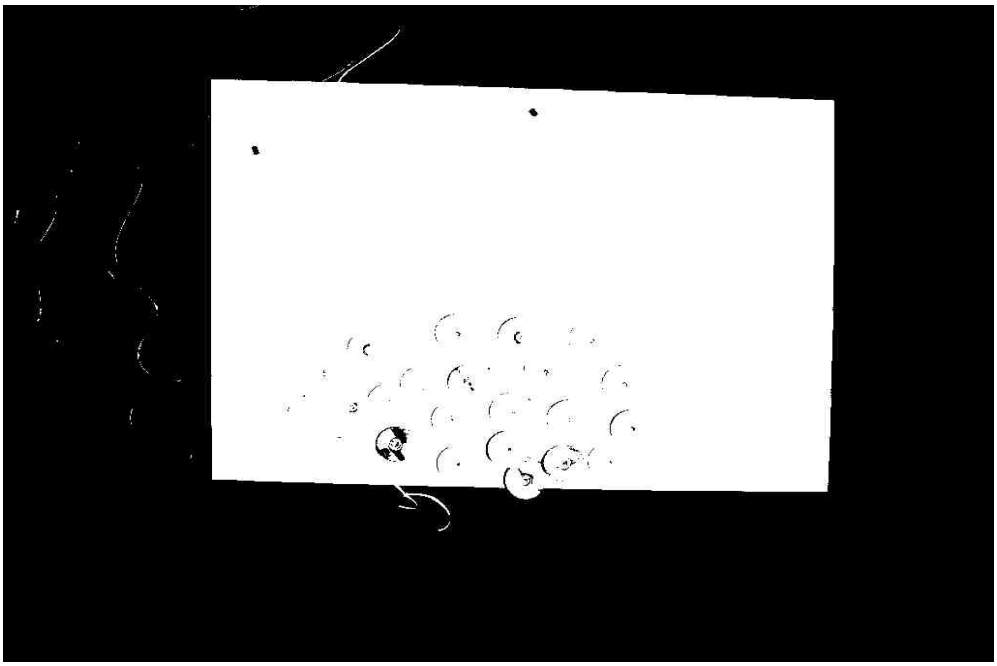

Figura 3.6 – Ejemplo de imagen tras el filtro binario, para detección de perspectiva

El siguiente paso será detectar los contornos sobre esta imagen, mediante las

funciones que facilita OpenCV para ello.

Un contorno es una lista de puntos que representa una curva que separa distintos segmentos o zonas diferenciadas en una imagen. El contorno reúne el conjunto de puntos que forman los píxeles límite de una zona de la imagen. Hay muchos modos de representar un contorno, en OpenCV se utilizan secuencias. En cada entrada de la secuencia se almacena información del siguiente punto de la curva, de forma que estos puntos quedan enlazados y es posible recorrerlos y tratarlos como una unidad.

La función de obtención de contornos de OpenCV distingue 4 modos distintos de determinación de los contornos de la imagen. Estos modos distinguen qué tipo de contornos se desean encontrar y de qué manera se debe presentar el resultado. Son los siguientes:

- CV_RETR_EXTERNAL
  Obtiene solamente la información de los contornos más externos, en el caso en el que haya unos contornos dentro de otros.

- CV_RETR_LIST
  Obtiene todos los contornos y los almacena en una lista, conectados entre sí uno tras otro.

- CV_RETR_CCOMP
  Obtiene todos los contornos y los organiza en una jerarquía de dos niveles, donde los contornos del nivel superior son externos y los del segundo nivel son contornos de agujeros (contornos dentro de otros contornos).

- CV_RETR_TREE
  Obtiene todos los contornos y reconstruye una jerarquía completa de contornos anidados, según los contornos estén unos dentro de otros.

En el caso del sistema que se está diseñando, se utilizará el modo CV_RETR_LIST, ya que se desean obtener todos los posibles contornos, por si acaso apareciese en la imagen un contorno mayor que contuviese al borde de la proyección, y no resulta relevante la jerarquía de dichos contornos. Los contornos detectados se filtrarán posteriormente para seleccionar únicamente el que interesa en esta fase.

Adicionalmente, la función de detección de contornos de OpenCV permite seleccionar entre 5 métodos de obtención de los contornos. Estos métodos definen la forma en la que los contornos son aproximados, y son los siguientes:

- CV_CHAIN_CODE
  Produce contornos en el código de cadena de Freeman [Free67], en comparación con el resto de los métodos que produce secuencias de vértices. Con una cadena de Freeman, un polígono se representa como una secuencia de pasos en una de las 8 direcciones (2 horizontales, 2 verticales y 4 diagonales). Cada paso se designa por un entero entre 0 y 7, comenzando por la dirección vertical hacia arriba en el 0, y continuando en el sentido de las agujas del reloj hasta la dirección diagonal hacia arriba-izquierda que correspondería con el 7. Un conjunto de estos pasos o desplazamientos en el orden adecuado recorrería el contorno a detectar.

- CV_CHAIN_APPROX_NONE
  Traduce todos los puntos del código de cadena a puntos normales en coordenadas cartesianas.

- CV_CHAIN_APPROX_SIMPLE
  Comprime los segmentos horizontales, verticales y diagonales de la cadena de puntos, dejando únicamente sus puntos finales.

- CV_CHAIN_APPROX_TC89_L1 o CV_CHAIN_APPROX_TC89_KCOS
  Aplica el algoritmo de aproximación de cadena de Teh-Chin, algoritmo

paralelo que detecta puntos dominantes en una curva cerrada. Este procedimiento primero determina la región de soporte de cada punto basándose en sus propiedades locales, luego mide la significancia relativa de cada punto y por último detecta los puntos dominantes mediante un proceso de supresión de no-máximos. [Teh89]

- CV_LINK_RUNS
  Utiliza un algoritmo completamente distinto al de los métodos anteriores, el cual une segmentos horizontales de 1s. El único modo de obtención que permite este método es CV_RETR_LIST.

En el caso del sistema que se pretende diseñar, se utilizará el método CV_CHAIN_APPROX_SIMPLE, ya que es sencillo y efectivo, y devuelve la información de la curva de forma comprimida y en coordenadas cartesianas con respecto a la imagen.

Otros contornos a parte de la zona de proyección pueden aparecer en la imagen captada. Para seleccionar el contorno que se busca, se utilizará una medida del área de dicho contorno. Probablemente estos otros contornos espurios serán de un tamaño inferior al del contorno buscado. Se utilizarán dos variables, área mínima y área máxima, las cuáles se podrán modificar dinámicamente y permitirán eliminar cualquier contorno con área inferior al área mínima o, en caso necesario, con área superior al área máxima. Esto facilitará la selección de un único contorno en entornos donde la imagen captada pueda contener elementos que podríamos considerar “ruidosos”.

A continuación se muestra un ejemplo de cómo sería el contorno detectado, superpuesto sobre la imagen inicial.

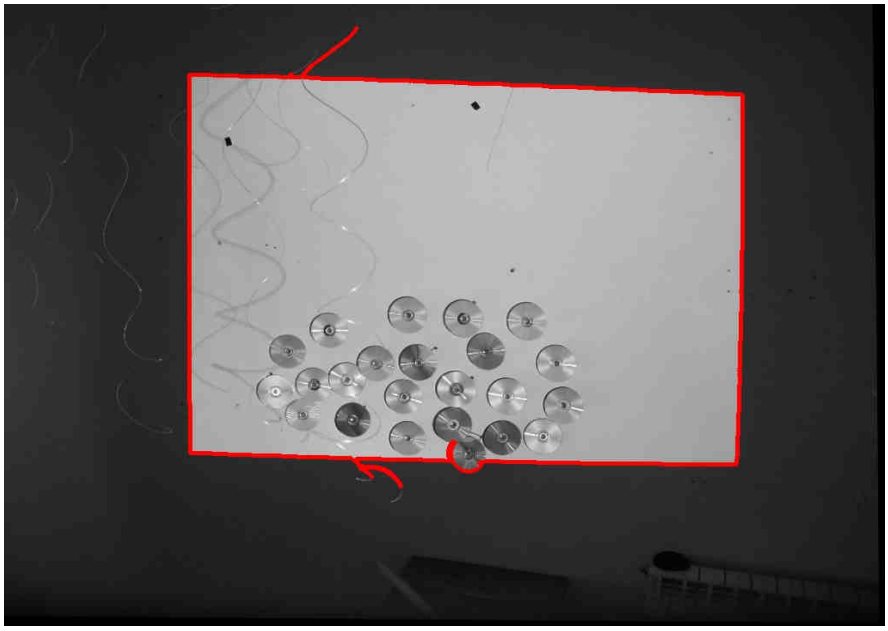

Figura 3.7 – Ejemplo de imagen con contorno detectado, para detección de perspectiva

Como se puede observar en la figura, el contorno detectado puede no ser perfecto, ya que suele estar afectado por elementos que se encuentren cercanos al borde del área de proyección. Sin embargo la zona que se quiere delimitar sí que es un polígono perfecto, así que es necesario eliminar estos elementos perturbadores del contorno.

Además, la información que se desea almacenar, y que se utilizará para limitar el área de búsqueda de bordes de DVDs, son únicamente los puntos de las esquinas. Con estos se podrá después transformar la perspectiva de la imagen en la que se desean detectar los bordes en la siguiente fase, de forma que dicha imagen coincida de forma precisa con la superficie sobre la que incide la luz.

Un método sencillo para calcular las esquinas del contorno consiste en hallar en primer lugar el punto central de dicho contorno, y a partir de este calcular el punto más alejado de él en cada uno de los cuadrantes. Estos cuadrantes quedan delimitados por la horizontal y la vertical, tomando como origen de coordenadas el

punto central hallado.

Continuando con el ejemplo anterior, este sería el resultado del polígono formado por las 4 esquinas detectadas.

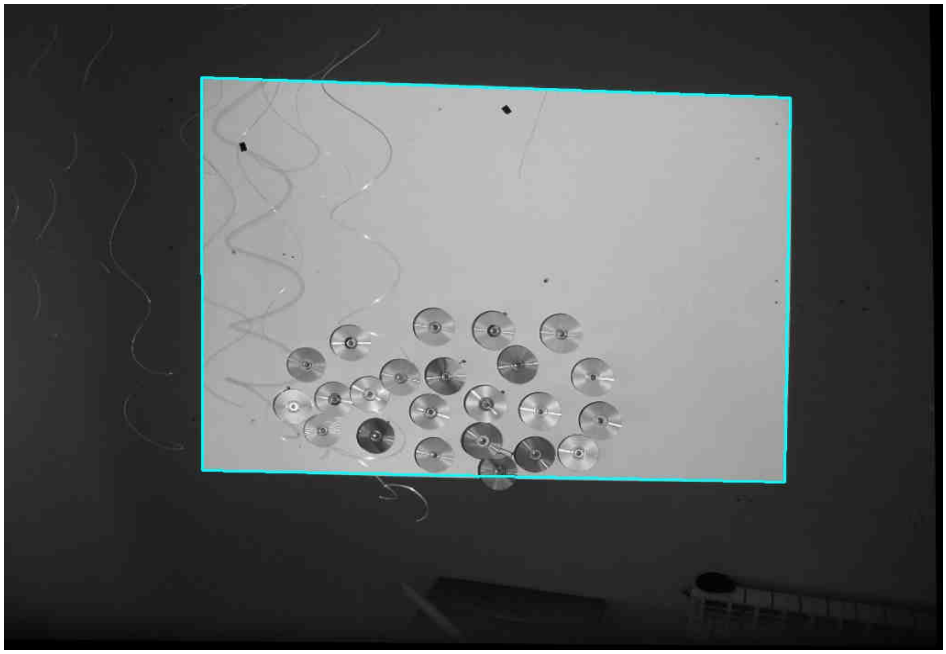

Figura 3.8 – Ejemplo de imagen con polígono de cuatro esquinas detectado, para detección de perspectiva

Una vez que se ha conseguido delimitar el área de proyección, es necesario encontrar un modo de transformar cada imagen captada por el dispositivo de entrada de forma que sólo se procese esta área. Las esquinas detectadas se corresponden con las esquinas globales de la imagen que realmente debe procesar el sistema en la siguiente fase, ya que es el área que ve el proyector de salida. Por tanto bastará con modificar cada imagen tomada por la cámara de entrada mediante una transformación geométrica utilizando la posición de estas esquinas detectadas, de manera que dichas esquinas se conviertan en las esquinas generales de la imagen resultado, la cual será utilizada para detectar la posición de los DVDs.

Existen dos tipos de transformaciones planas en imágenes: transformaciones afines y transformaciones de perspectiva u homografías [Bra09].

El primer tipo, las transformaciones afines, utilizan una matriz de 2x3. Visualmente pueden explicarse del siguiente modo: partiendo de un rectángulo, mediante una transformación de este tipo se podría obtener cualquier paralelogramo. Pueden comprimir o estirar la imagen, pero los lados deben ser siempre paralelos dos a dos. Las transformaciones afines tienen menor número de parámetros que las de perspectiva, y por lo tanto son más sencillas de resolver. Sin embargo, aunque para pequeñas modificaciones el resultado de esta transformación es suficiente, los verdaderos cambios de perspectiva sólo pueden modelarse mediante el siguiente tipo de transformación.

El segundo tipo, las transformaciones de perspectiva, utilizan una matriz de 3x3. Visualmente pueden explicarse del siguiente modo: partiendo de un rectángulo, mediante una transformación de este tipo se podría obtener cualquier trapezoide. Las transformaciones afines son una subclase de las transformaciones de perspectiva. Estas últimas ofrecen mayor flexibilidad, y son las que se utilizarán en este sistema.

En estas transformaciones geométricas, se está convirtiendo un conjunto de puntos de una imagen a otro conjunto de puntos en otra imagen, lo cual puede parecer un mapeado de dos dimensiones a dos dimensiones. Sin embargo esto no es correcto, ya que la transformación de perspectiva es realmente un mapeado de un plano en dos dimensiones a un espacio tridimensional, y posteriormente otro mapeado desde este espacio tridimensional a un subespacio bidimensional distinto. Es por esta razón por lo que se necesita una matriz de 3x3 para este tipo de transformaciones.

A continuación se muestra un ejemplo de como quedaría la imagen rectificada.

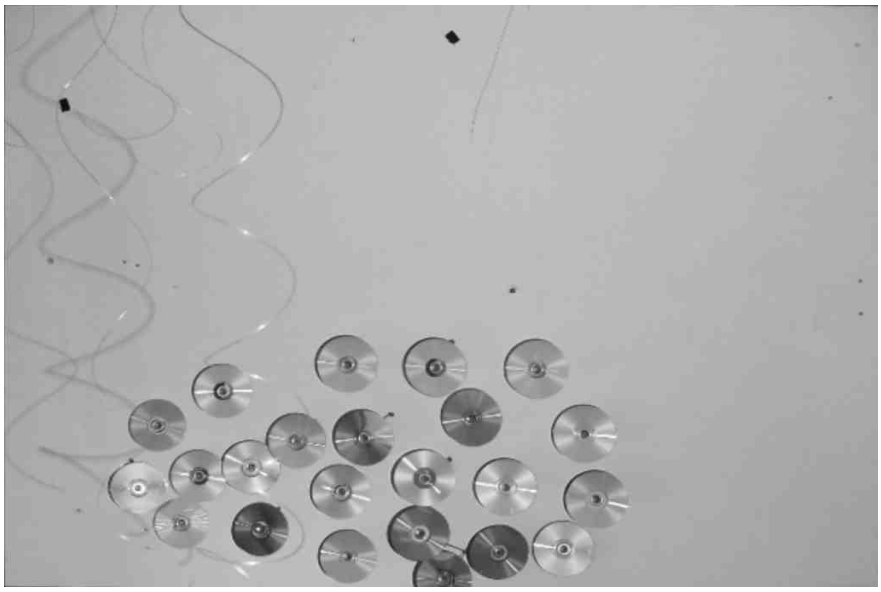

Figura 3.9 – Ejemplo de imagen con perspectiva ya transformada

La información de la posición y tamaño del área de proyección será necesaria en el siguiente apartado a la hora de detectar la posición de los DVDs, por lo que se almacenará la situación de las esquinas del polígono para poder transformar las futuras imágenes captadas durante la detección de DVDs y poder convertirlas a la perspectiva deseada.

3.3 Detección de los contornos de los DVDs

Inicialmente para esta fase, se tomará una imagen desde el dispositivo de entrada, ya sea cámara de vídeo o cámara de fotos. Para esta fase no es necesario que el proyector esté encendido ni proyectando una imagen en blanco, ya que la zona de proyección ya está delimitada y la luz que emite el proyector originaría sombras en la parte trasera de los DVDs que complicarían la detección precisa de los bordes de estos. Esta imagen debe ser tomada sin desplazar el dispositivo de entrada con respecto a la toma de la imagen utilizada en la fase anterior, ya que si no las coordenadas de los puntos que delimitan la zona de proyección no serían las mismas en la nueva imagen.

Cuando no es posible mantener fijo el dispositivo con el que se obtienen las imágenes entre la toma de una imagen y otra, se puede utilizar la misma para ambas fases. Los brillos y sombras que produce la intensa luz direccional del proyector hacen que la detección de los contornos de los DVDs sea menos efectiva que con luz ambiente, pero no imposible. Sin embargo el que las imágenes que se utilicen en ambas fases estén sacadas desde exactamente el mismo ángulo y posición sí que es un factor crucial en el buen funcionamiento del sistema. Este sería un posible ejemplo de imagen a utilizar.

Figura 3.10 – Ejemplo de imagen inicial para detección de DVDs

En primer lugar sería necesario eliminar de esta imagen la distorsión introducida por la cámara y cambiar la perspectiva tomando únicamente la zona de proyección como imagen a procesar. El resultado sería el mostrado en la siguiente figura.

Figura 3.11 – Ejemplo de imagen con perspectiva transformada, para detección de DVDs

Uno de los principales problemas a la hora de detectar los contornos de los DVDs está originado por la propiedad reflectante de la superficie de estos. Al incidirles la luz, ya sea luz ambiente o luz directa, aparecen una serie de reflejos y brillos sobre la cara exterior de los DVDs que complican su aislamiento y detección como un ente único.

En algunos de estos brillos la luz blanca se ha descompuesto en componentes de diversos colores. Aprovechando esta característica, se realizará un primer procesado de la imagen, intentando eliminar o reducir aquellos brillos que poseen alguna tonalidad de color. Para ello se descompondrá la imagen en sus componentes R, G y B. Comparando píxel a píxel estas tres imágenes de canal único, en aquellas zonas donde la intensidad de una de ellas sobresalga sobre las otras dos, esta se eliminará. Recombinando las tres componentes de nuevo, se obtendrá una imagen equivalente a la inicial, sólo que con menor número de brillos.

A continuación se debe aplicar un filtro que permita detectar más fácilmente los límites de los DVDs, eliminando la información superflua de la foto. En este

sistema se ha optado por dos filtros distintos en paralelo, ya que ambos tienen sus ventajas y sus inconvenientes, y pueden rendir mejor en unos casos o en otros. Estos filtros son el filtro binario y el filtro Canny.

Filtrado binario y operadores morfológicos

El filtro binario es un filtro blanco/negro que, a partir de un umbral, convierte los píxeles a negro cuando su intensidad se encuentra por debajo de ese umbral o a blanco cuando se encuentra por encima. Este tipo de filtro ayuda a la detección de los contornos de los DVDs ya que, con el umbral adecuado, resalta la diferencia de intensidad ya existente entre la superficie del DVD y el fondo de la pared. El umbral de este filtro se podrá modificar dinámicamente para ajustarlo al nivel de claridad variable de la imagen de entrada.

Tras obtener la imagen binaria, se utilizará un filtrado morfológico [Bra09] para reducir el problema de segmentación de los contornos debido a los brillos y reflejos de estos, para aislar cada DVD y para eliminar el ruido residual que pudiese quedar en la imagen. Las transformaciones básicas del filtrado morfológico son las de erosión y dilatación. En el caso del sistema a estudio, utilizaremos dos combinaciones de ambas, la apertura y el cierre.

El instrumento fundamental de las transformaciones morfológicas es el elemento estructurante, también llamado kernel. Un elemento estructurante se define como una configuración de píxeles en la cual hay un origen definido, también llamado ancla. Aplicar un filtrado morfológico consiste en realizar un sondeado a cada píxel de la imagen utilizando el elemento estructurante. Cuando el origen del elemento está alineado con un píxel dado, su intersección con la imagen define un conjunto de píxeles en los que se aplica una operación morfológica particular. En principio el kernel puede tener cualquier forma, pero comúnmente se utiliza una forma simple como un cuadrado, un círculo o un diamante, con el origen en el centro. En este sistema se utilizará una cruz, ya que es el elemento más sencillo que mejor se ajusta a la línea curva del borde de los DVDs.

La operación básica de erosión reemplaza el píxel actual por el valor inferior de todos los píxeles que abarca el kernel. La dilatación es la operación complementaria, ya que reemplaza el píxel actual por el valor máximo de todos los píxeles que abarca el kernel. Ya que la imagen binaria sólo contiene píxeles en blanco o en negro, cada píxel será reemplazado por otro de uno de los dos colores. La erosión tiene el efecto de convertir en fondo (negro) los píxeles que se encuentran en el límite de las figuras de la imagen. En el caso de la dilatación, el efecto es el opuesto y los píxeles que se encuentran en el límite se convertirán en figura (blanco). Cuáles y cuántos de estos píxeles limítrofes cambian de color dependerá de la forma del elemento estructurante. Así, en una imagen erosionada las figuras se reducirán, lo cual permitirá eliminar pequeños elementos ruidosos, mientras que en una dilatada las figuras aumentarán de tamaño, permitiendo que desaparezcan pequeños agujeros dentro de las figuras.

La operación de apertura consiste en la suma de una operación de erosión seguida de una operación de dilatación. La operación de cierre consiste en la suma de una operación de dilatación seguida de una operación de erosión.

Tras aplicar un filtro de cierre, los pequeños agujeros de las figuras blancas se rellenan y los objetos adyacentes quedan conectados. Básicamente, cualquier agujero o espacio libre demasiado pequeño para contener el elemento estructurante completo será eliminado durante el filtrado. Recíprocamente, el filtrado de apertura elimina las figuras pequeñas de la escena, todas aquellas que son demasiado pequeñas para contener al kernel.

Estos filtros son muy útiles en la detección de objetos, ya que el cierre conecta los elementos erróneamente fragmentados mientras que la apertura borra las pequeñas manchas debidas al ruido de la imagen. Por ello resulta útil utilizar ambos en secuencia. Aplicando ambos filtros sucesivamente se obtiene una imagen donde se muestran los objetos principales de la escena. También se puede aplicar el filtrado de apertura antes que el de cierre si se desea dar prioridad al

filtrado del ruido, pero esto puede tener como consecuencia la eliminación de algunos objetos fragmentados.

A continuación se muestra un ejemplo del efecto que tiene la utilización de los operadores de cierre y apertura conjuntamente sobre una imagen binaria. A la izquierda se muestra la imagen original, mientras que a la derecha se muestra la imagen tras la transformación morfológica.

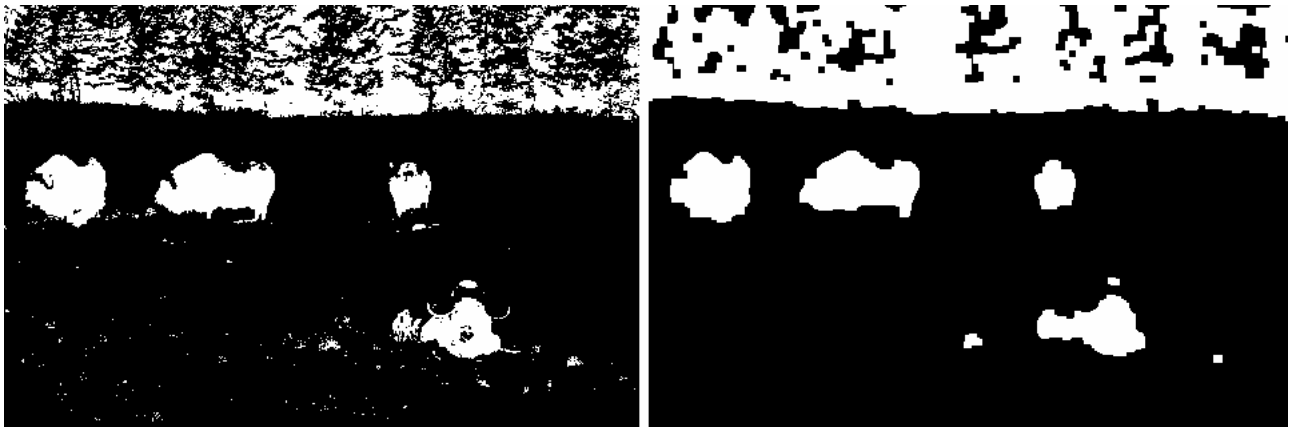

Figura 3.12 – Transformación morfológica

Como ya se ha comentado, en el sistema diseñado se utilizará un filtrado de apertura seguido de otro de cierre, utilizando un kernel simple en forma de cruz. Un ejemplo del resultado sería el mostrado en la siguiente figura.

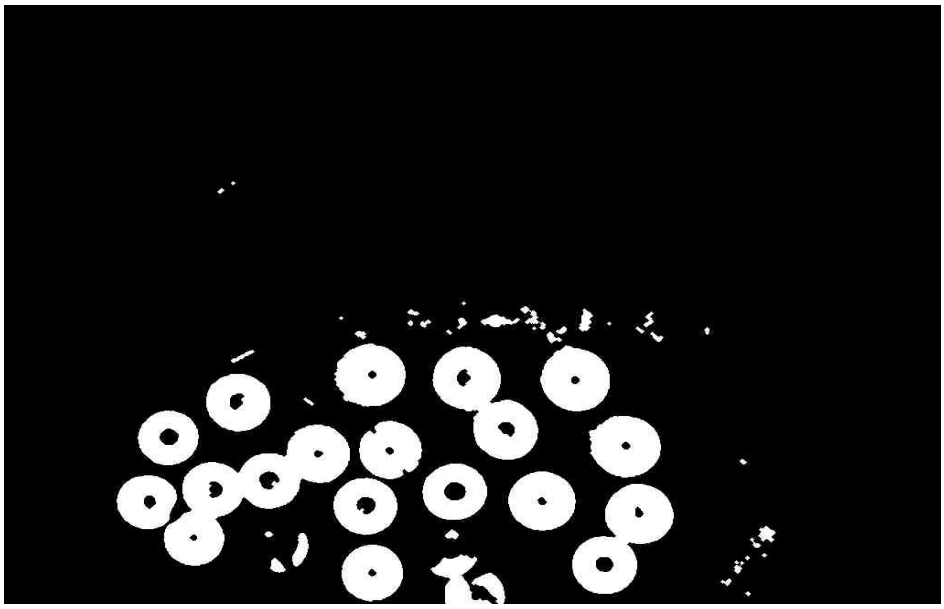

Figura 3.13 – Ejemplo de imagen tras filtrado binario y apertura/cierre, para detección de DVDs

Filtrado Canny

El otro tipo de filtrado realizado en paralelo junto con el binario es el filtrado Canny [Bra09]. Este es un método para encontrar límites de objetos basado en derivadas.

La primera derivada de una función será mayor en los puntos en los que ésta cambie rápidamente, y por tanto crecerá rápidamente según se va aproximando a la discontinuidad de un borde, mientras que se reducirá drásticamente tras haber pasado dicha discontinuidad. Por tanto la derivada tendrá un máximo local allí donde haya una discontinuidad, el cuál se puede localizar buscando los 0's en la segunda derivada. Los límites en la imagen original serán 0's en la segunda derivada. Desafortunadamente, los límites menos significativos también serán 0's.

El método de detección de bordes de Canny calcula las primeras derivadas en x e y, y estas se combinan en cuatro derivadas direccionales. Los puntos donde esas derivadas direccionales son máximos locales son los candidatos a convertirse en bordes. El algoritmo de Canny intenta ensamblar los píxeles individuales candidatos a ser límite aplicando un umbral de histéresis. Se utilizan dos umbrales, uno superior y otro inferior. Si un píxel tiene un gradiente mayor que el del umbral superior es aceptado como píxel límite. Si un píxel está por debajo del umbral inferior, no se considera píxel límite. Si el gradiente del píxel está entre ambos umbrales, este será aceptado solo si está conectado a otro píxel que esté por encima del umbral superior. Se recomienda un ratio entre umbrales de entre 2:1 y 3:1.

A continuación se muestra la imagen anterior aplicando un filtro Canny.

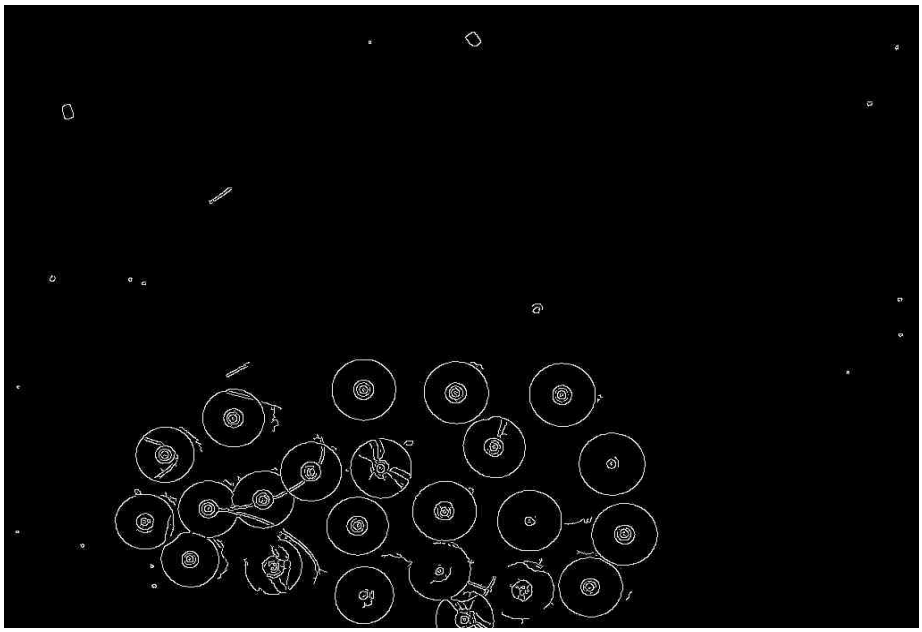

Figura 3.14 – Ejemplo de imagen tras filtrado Canny, para detección de Vds.

A partir de las dos imágenes filtradas anteriores, obtenidas con los métodos de filtrado binario y filtrado Canny, es necesario obtener los contornos de los DVDs. Al igual que en la fase anterior, se utilizarán las funciones facilitadas por OpenCV, utilizando la opción CV_RETR_LIST como modo de recuperación y la opción CV_CHAIN_APPROX_SIMPLE como método de obtención de contornos.

En muchos casos estos contornos muestran entrantes debidos a los brillos y reflejos de la superficie del cd, al ruido o a la iluminación. Dado que el borde de un cd siempre es una curva convexa, se pueden eliminar algunos de estos errores convirtiendo los contornos detectados a una versión de dichos contornos que suprima estos entrantes, de forma que queden totalmente convexos.

La siguiente imagen muestra un ejemplo de los contornos convexos detectados, superpuesta a la imagen original.

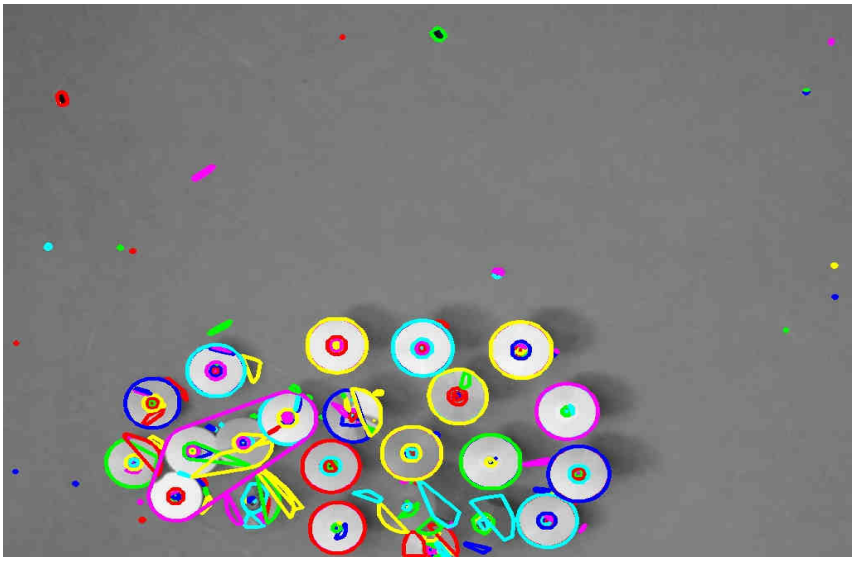

Figura 3.15 – Ejemplo de imagen con contornos detectados, para detección de DVDs

En algunos casos y a pesar de todo el procesamiento y el filtrado previo, los contornos finalmente detectados quedan fragmentados debido a reflejos y brillos insalvables sobre la superficie del DVD. Por ello el sistema facilita una herramienta de ensamblado manual de estos contornos, que permite unir partes de un mismo DVD que han sido detectadas como figuras distintas. Bastará con señalarlas con el ratón sobre la imagen que muestra todos los contornos detectados. Al unir dos contornos, se volverá a comprobar que el nuevo contorno es convexo, y se convertirá en convexo en caso de que no lo fuera.

Hasta ahora el diseño realizado ha sido independiente de cuál es la forma de los elementos que se desean detectar. En la última parte del procesamiento de esta fase se va a tener en cuenta que el elemento a detectar son DVDs. Si se desease aplicar el sistema a la detección de otro tipo de elementos, bastaría modificar este último paso del proceso para adecuarlo a la forma de dichos elementos.

Los elementos que se desean detectar son DVDs, y por tanto círculos perfectos. Sin embargo estos DVDs no se encuentran situados de forma totalmente paralela al dispositivo de entrada ni al proyector de salida, por lo que estos serán detectados como elipses de diverso tamaño y forma, dependiendo de su

inclinación y su distancia a la pared.

El conocer este dato facilita la detección precisa de las formas de los DVDs, ya que los contornos detectados deben seguir el patrón de una elipse. Es posible aproximar los contornos detectados a elipses mediante OpenCV, lo cuál resulta útil como una última herramienta que corrija los errores en la detección de los límites. La rutina de OpenCV devuelve la elipse que se aproxima mejor al contorno inicial, por lo que no todos los puntos del contorno estarán incluidos en la elipse. Esta se obtiene mediante una función de ajuste de mínimos cuadrados, intentando por tanto minimizar el error cuadrático de la aproximación.

En la siguiente imagen se muestra un ejemplo de elipses detectadas, superpuestas sobre la imagen inicial.

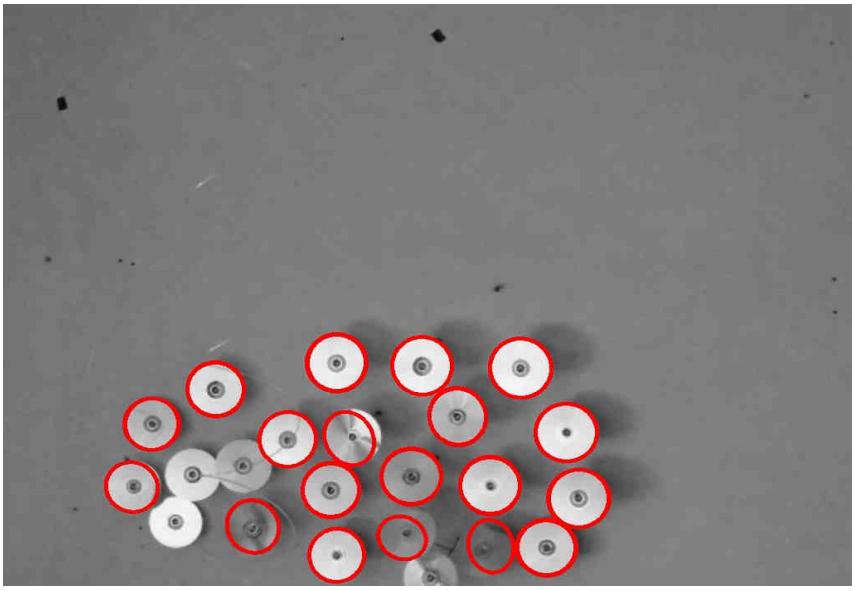

Figura 3.16 – Ejemplo de imagen con elipses detectados, para detección de DVDs

Para eliminar posibles contornos ruidosos que, tras todo el procesamiento, hayan podido mantenerse y haber sido transformados en elipses, el sistema permite realizar un último filtrado teniendo en cuenta el radio de las elipses. Este filtrado es sencillo de realizar, ya que la mayoría de las elipses, al estar a una distancia similar del dispositivo de entrada y ser DVDs estándar, van a tener un radio parecido. De esta forma, existirá un parámetro modificable dinámicamente que marque el radio

mínimo que ha de tener una elipse para ser detectada, y así mismo otro con el radio máximo.

Finalmente, aquellas elipses que se desea que muestren un vídeo sobre su superficie tras la siguiente fase, serán señaladas manualmente. De esta forma se permite definir exactamente qué elipses han sido detectadas correctamente con precisión, y tras señalar estas se podrá reprocesar la imagen con otros valores de los parámetros configurables para tratar de ajustar mejor las que no.

Las elipses detectadas y elegidas se mostrarán en una máscara a la par que se almacenarán en un archivo de texto. La máscara permite visualizar fácilmente qué elipses han sido detectadas, mientras que el archivo de texto será útil a la hora de reutilizar una misma detección de elipses para varios procesados del ensamblaje de vídeos, el cual se lleva a cabo en la última fase.

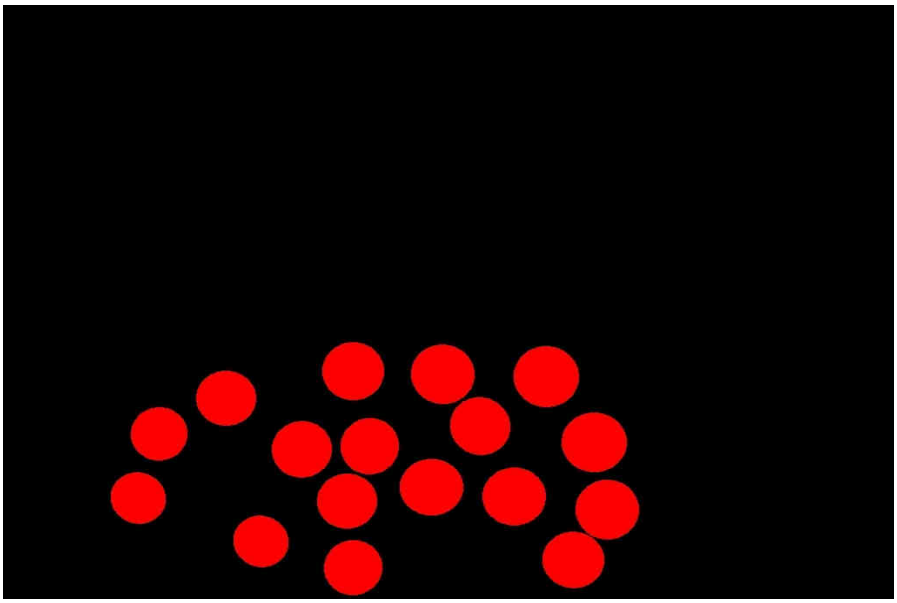

Figura 3.17 – Ejemplo de máscara con superficies detectadas

El archivo de texto que almacenará las elipses se compondrá de un conjunto de líneas, cada una de las cuales definirá una elipse. El formato de cada línea será el siguiente:

x_centro y_centro ancho alto ángulo

donde *x_centro* e *y_centro* son las coordenadas cartesianas del centro de la elipse, tomando como origen la esquina superior izquierda de la imagen, *ancho* y *alto* son las longitudes de los semiejes, y *ángulo* corresponde con el ángulo de inclinación del primero de los ejes de la elipse con respecto a la horizontal.

3.4 Ensamblaje de los vídeos

La salida final del sistema diseñado ha de ser un vídeo que, proyectado desde el proyector de salida conectado al ordenador, muestre sobre cada elipse detectada un vídeo diferente. Estos vídeos tendrán unos instantes de comienzo y fin de reproducción configurables, y contendrán giros de la imagen, cuyo número, velocidad, sentido e instante de inicio y fin también serán configurables. (Ver croquis del montaje en figura 1.2)

Debido a que sucesivas instalaciones de la pieza artística en diversos entornos suelen utilizar la misma configuración de reproducción de vídeos, las características de ésta se encontrarán almacenadas en un archivo de texto. Por lo tanto el primer paso de esta fase de la aplicación será leer la información necesaria de dicho archivo. Cada línea de este se corresponde con el vídeo de una elipse, y su formato es el siguiente:

id nombre frame_inicio frame_fin n_giros inicio_giro1 fin_giro1 n_vueltas1 inicio_giro2 fin_giro2 n_vueltas2 ...

donde

- *id* es un número que identifica unívocamente al vídeo
- nombre es el nombre del archivo de vídeo de donde se obtendrán las imágenes, incluyendo la ruta de acceso a él
- *frame_inicio* y *frame_fin* marcan los instantes del vídeo final en los que el vídeo actual comienza y termina de reproducirse

- *n_giros* determina el número de giros que efectuará el vídeo a lo largo de la reproducción

  Los siguientes tres valores se repetirán tantas veces como indique *n_giros*, ya que habrá una por giro
- *inicio_giroX* y *fin_giroX* indican el frame del vídeo actual en el que la imagen comienza y termina de girar
- *n_vueltasX* indica el número de vueltas que el vídeo actual debe dar durante el giro, el signo indica el sentido del giro

Cada uno de estos vídeos se asociará a una de las elipses detectadas. Para poder reproducir el vídeo dentro de la elipse de forma que se muestre la mayor cantidad de imagen posible pero que al mismo tiempo no aparezca ninguna zona sin imagen, hay que redimensionar el vídeo adecuadamente. Dado que la imagen del vídeo va a girar sobre el centro de la elipse, el punto más alejado de dicho centro se va a corresponder con el eje mayor de la elipse. Por tanto el vídeo debe escalarse para que la dimensión menor del vídeo, ya sea altura o anchura, coincida con el tamaño del eje mayor de la elipse.

Para determinar qué puntos de la imagen del vídeo caen dentro de la elipse, y por tanto se van a mostrar, es necesario conocer los puntos que delimitan dicha elipse.

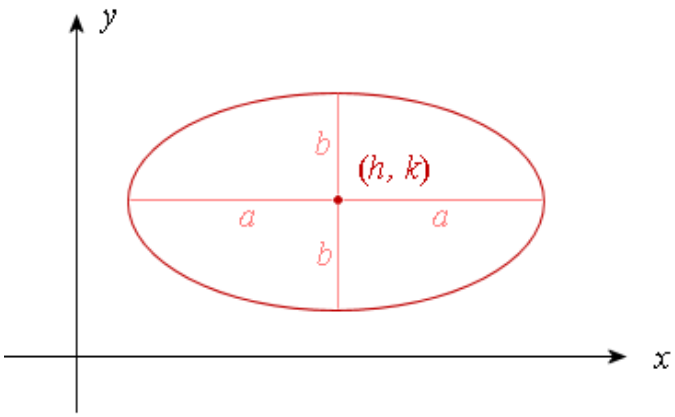

Figura 3.18 – Elipse

Una elipse es el lugar geométrico de todos los puntos de un plano tales que la suma de las distancias a otros dos puntos fijos llamados focos es una constante

positiva [L31]. La ecuación en coordenadas cartesianas de una elipse sería la siguiente:

$$\frac{(x-x_1)^2}{a^2}+\frac{(y-y_1)^2}{b^2}=1$$

donde (x_1, y_1) es el centro de la elipse y *a* y *b* es el tamaño de los ejes.

Aunque esta es la ecuación más común que representa una elipse, en este caso va a ser más útil utilizar las ecuaciones paramétricas siguientes:

$$X(t)=X_c+a\,\cos t\,\cos\varphi-b\,\sin t\,\sin\varphi$$
$$Y(t)=Y_c+a\,\cos t\,\sin\varphi+b\,\sin t\,\cos\varphi$$

donde (X_c, Y_c) es el centro de la elipse, *a* y *b* es el tamaño de los ejes, φ es el ángulo de inclinación del eje principal de la elipse con respecto al eje horizontal, y *t* es un parámetro que varía entre 0 y 2π. El valor de *t* no se corresponden geométricamente con ningún elemento, pero al ir variando dicho parámetro se obtendrán todos los valores (*X(t), Y(t)*) que conforman el perímetro de la elipse.

Para averiguar si un punto de la imagen cae dentro del área de la elipse es necesario conocer cuál es el punto del perímetro de esa elipse correspondiente al mismo ángulo que el punto analizado, y comprobar si su distancia al centro es mayor o menor que el de este. Distancias menores que la del punto del perímetro implican que el punto está dentro y por tanto se debe reproducir, distancias mayores implican que está fuera y por tanto no se debe reproducir.

Para obtener el ángulo en el que está situado un punto con respecto al centro de la elipse, basta con utilizar un poco de trigonometría y aplicar la siguiente ecuación:

$$\varphi=\arctan\left(\frac{(y-y_c)}{(x-x_c)}\right)$$

El signo y el valor de este ángulo deberá ajustarse en función del cuadrante en el que se encuentre el punto, lo cual se averigua fácilmente conociendo el signo de las coordenadas *x* e *y*. También ha de tenerse en cuenta el caso en que *x* sea 0, ya que esta función tiene una singularidad para ese conjunto de puntos. Suponiendo que la función arcotangente devuelve un valor de ángulo en radianes de entre -π/2 y π/2, el ángulo donde está situado el punto se puede obtener mediante el siguiente esquema:

$x > 0, y >= 0$ (primer cuadrante) $\rightarrow \varphi = \text{atan}(y/x)$

$x < 0, y >= 0$ (segundo cuadrante) $\rightarrow \varphi = \text{atan}(y/x) + \pi$

$x < 0, y < 0$ (tercer cuadrante) $\rightarrow \varphi = \text{atan}(y/x) + \pi$

$x > 0, y < 0$ (cuarto cuadrante) $\rightarrow \varphi = \text{atan}(y/x) + 2\pi$

$x = 0, y >= 0 \rightarrow \varphi = \pi/2$

$x = 0, y < 0 \rightarrow \varphi = 3\pi/2$

Los puntos que conforman el perímetro de la elipse podrían calcularse en cada comprobación, pero esto resultaría en un procesamiento muy complejo, ya que se desconoce el valor del parámetro *t* para el que el ángulo coincidiría con el buscado, y por tanto habría que ir barriendo sus posibles valores hasta dar con él. Resulta mucho más sencillo calcular los puntos de la elipse inicialmente y almacenarlos en una tabla. La información que se necesita almacenar para cada punto de la elipse está compuesta por el ángulo en el que está situado ese punto y la distancia de dicho punto al centro. Por tanto se irán recorriendo los posibles valores de *t* entre 0 y 2π, y almacenando ambos datos en la tabla. Para facilitar aún más el acceso a la información, y dado que una precisión de un grado en la medida es suficiente para el presente sistema, se almacenarán las distancias al centro de la elipse en una tabla de 1x360, donde el índice de la tabla indica el número de grados del ángulo donde está situado el punto, y el valor almacenado indica la distancia a este.

A la hora de tomar la imagen del vídeo correspondiente a una elipse en un instante dado, se ha de tener en cuenta la configuración de instantes de inicio y fin de la reproducción y de los giros. Para cada giro se conocen el número de vueltas que

debe dar la imagen y los instantes de inicio y fin del giro. Por tanto se puede obtener la velocidad radial de giro, es decir, el número de grados que debe girar la imagen del vídeo entre un frame y el siguiente, dividiendo los 360 grados, multiplicados por el número de vueltas, entre el tiempo de duración del giro, que no es otro que la diferencia entre los instantes de inicio y fin de giro. Asimismo, el signo del número de vueltas indicará el sentido del giro, de forma que un número de vueltas negativo dará lugar a una velocidad radial negativa, es decir un giro en el sentido contrario.

Para realizar dicha rotación se utilizará una transformada geométrica como las estudiadas en la fase II al hablar del cambio de perspectiva del área de proyección detectada. En este caso la matriz de transformación será una matriz de rotación que girará todos los píxeles de la imagen los grados indicados.

Para obtener el vídeo final en el que estén insertados todos los vídeos con forma de elipse, se generará una imagen para cada frame del tamaño en el que se desea realizar la proyección. Se recorrerán las elipses una por una, y para cada una de ella se insertarán los píxeles correspondientes a la zona del vídeo incluida en la elipse, en la posición adecuada y con la forma correspondiente. Uniendo todos estos frames, se obtendrá el video final que se proyectará a través del proyector de salida.

La siguiente imagen muestra un ejemplo de uno de los instantes del vídeo de salida.

Figura 3.19 – Vídeo resultado a proyectar sobre los DVDs

El sistema permite adicionalmente realizar esta última fase del procesamiento, la generación de vídeos, a partir de un archivo de texto donde estén almacenadas las elipses detectadas. El proceso es equivalente al explicado en este apartado, con la salvedad de que inicialmente la información de esas elipses debe ser leída desde el archivo en lugar de obtenerse directamente de la fase anterior. Esto es útil para poder generar diversas configuraciones de vídeos de salida sobre un mismo conjunto de DVDs detectados que no van a ser desplazados de una proyección a otra.

4 IMPLEMENTACIÓN

El sistema se ha implementado utilizando el lenguaje C++, orientado a objetos, y las librerías OpenCV, como ya se ha comentado anteriormente. Este se compone de las siguientes clases:

- `Calibration`
- `PerspectiveDetection`
- `PerspectiveChange`
- `EllipsesDetection`
- `OneMethodEllipsesDetection`
- `Hull`
- `Ellipse`
- `VideoGenerator`
- `Video`
- `VideoFromFileGenerator`

Estas clases junto con el método main y los métodos callback incluidos en el archivo del método main, forman el total del código del proyecto.

Se van a analizar cada una de estas clases por separado según en la fase en la que se utilizan (ver fases en el apartado de diseño), aunque algunas de ellas se emplean en más de una fase. Antes de nada se describirá el método main, dejando sin embargo los métodos callback para el final del apartado, para que su explicación sea más comprensible, al haber estudiado ya las fases donde se utilizan.

4.1 Método main

Método principal del proyecto, desde el que se va llamando al resto de elementos de la aplicación para que lleven a cabo las tareas deseadas.
Este método muestra inicialmente un menú con tres opciones a elegir:

1. Detección con webcam → toma como dispositivo de entrada la webcam conectada al ordenador.
2. Detección con fotografías → toma como entrada dos imágenes tomadas previamente mediante una cámara de fotos.
3. Generación de vídeo desde archivo → a partir de un archivo elipses.txt ya generado en ejecuciones anteriores, donde se ha almacenado la descripción de las elipses detectadas, genera un nuevo vídeo de salida donde los vídeos configurados en el archivo de configuración coincidan con dichas elipses.

Cuando la opción 1 es pulsada, el método inicializa la captura de la cámara mediante la función `cvCreateCameraCapture`. A continuación crea un objeto de tipo `PerspectiveChange` que almacenará el cambio de perspectiva detectado por la fase II de la ejecución, y un objeto de tipo `PerspectiveDetection`, que ejecutará la detección del cambio de perspectiva con la llamada al método `detect` de este último que se realiza justo después de su inicialización.
Una vez obtenida la perspectiva y almacenada en el objeto `PerspectiveChange`, el método `main` crea un objeto de tipo `EllipsesDetection`, el cuál se encargará de ejecutar la detección de los bordes de los DVDs a través de la llamada a su método `detect` que se realiza justo después de su inicialización. Este último, antes de acabar, será el encargado de llamar a la clase `VideoGenerator` para generar el vídeo de salida de la fase IV.

Cuando la opción 2 es pulsada, la lista de acciones es la misma salvo por la inicialización de la captura de la cámara.

Cuando la opción 3 es pulsada, se crea un objeto de tipo

`VideoFromFileGenerator` y se llama a su método `writeVideo` para que se encargue de generar el vídeo.

4.2 Calibración de los dispositivos de entrada

Para esta fase será necesaria únicamente la clase `Calibration`.

4.2.1 Calibration

Esta clase permite realizar el calibrado de los dispositivos de entrada, ya sea una webcam o una cámara de fotos. Para efectuar la medida de las propiedades del dispositivo, parte de un conjunto de imágenes, tomadas con este, de un tablero de ajedrez en distintas perspectivas y con varios ángulos de rotación. Esta clase contiene los siguientes métodos:

```
Calibration()
```

Constructor que inicializa las constantes del número de cuadros del tablero de ajedrez, el número de imágenes utilizadas para la calibración y el ancho en cuadros del tablero de ajedrez.

```
void getParams(char* dist_filename, char* intr_filename)
```

Método que obtiene los parámetros intrínsecos de la cámara y los coeficientes de distorsión, y los almacena en dos archivos de texto para que puedan ser utilizados posteriormente. Los parámetros que recibe como entrada son los nombres que deben tener estos archivos de texto.
En primer lugar se llama al método `getImage` con cada imagen del tablero de ajedrez que se utilice durante la calibración. Así se obtienen los puntos de las esquinas de dicha imagen y se almacenan de forma ordenada en las matrices necesarias para llevar a cabo la calibración. Estas matrices son 3:

- `image_points` → almacena las coordenadas (x, y) de los puntos de las esquinas detectados. Tiene tamaño (num_esquinas*num_imágenes) x 2.
- `object_points` → almacena las coordenadas físicas de cada punto, es decir, la imagen en la que se encuentra, el número de punto con respecto al resto de puntos de esa misma imagen, y teóricamente también podría almacenar un valor para la coordenada z, que en este caso será siempre 0. Tiene tamaño (num_esquinas*num_imágenes) x 3.
- `point_counts` → almacena el número de puntos detectados en cada imagen. Tiene tamaño num_imagenes x 1.

Tras obtener estos valores, se llama a la función `cvCalibrateCamera2`, la cual devuelve dos archivos de texto con los coeficientes intrínsicos y los de distorsión, que se almacenarán de forma local.

```
void getImage(char* filename, int image_num)
```

Método que detecta en una imagen las esquinas del patrón de tablero de ajedrez que aparece en esta, y almacena la información de dichas esquinas en la posición adecuada de las matrices que se utilizarán para el calibrado del dispositivo.
Recibe como parámetro de entrada el nombre de la imagen a procesar, incluyendo su ruta.
Para llevar a cabo esto, inicialmente carga la imagen desde el archivo con el método `cvLoadImage`. Se utiliza el flag `CV_LOAD_IMAGE_GRAYSCALE` para cargar la imagen en escala de grises, ya que la información del color no es relevante a la hora de procesar un patrón de cuadros blanquinegros.
A continuación se utilizará la función `cvFindChessboardCorners` para localizar las esquinas del tablero, y la función `cvFindCornerSubPix` para procesar los puntos detectados de forma que se eliminen los posibles errores de precisión debidos al carácter discreto de los píxeles.
Una vez obtenidos los puntos de las esquinas, estos se almacenan en las matrices necesarias para realizar la calibración, en la posición correspondiente al número de imagen.

```
void undistort(IplImage* img, char* dist_filename, char* intr_filename);
```

Método que, a partir de una imagen de entrada y del nombre de dos archivos con coeficientes intrínsecos y de distorsión de la cámara, obtiene una versión sin distorsionar de la imagen inicial.
Los archivos de coeficientes deben estar almacenados localmente, y por tanto deben haber sido calculados con anterioridad a la llamada de este método. Estos coeficientes se cargan en dos matrices mediante el método `cvLoad`, con las cuales se calcula los mapas de distorsión en las coordenadas x e y, mediante el método `cvInitUndistortMap`.
Aplicando estos mapas de distorsión a la imagen de entrada con el método `cvRemap`, se consigue una versión sin distorsión de la imagen, la cual se almacenará en la misma variable para su acceso posterior desde el punto de la aplicación desde el que este método ha sido llamado.

4.3 Delimitación del área de proyección

Durante esta fase se emplearán las clases `PerspectiveDetection`, `PerspectiveChange`, y se utilizará la clase `Calibration` de la fase anterior únicamente para corregir la distorsión de la imagen de entrada.

4.3.1 PerspectiveDetection

Esta clase encapsula todo el procesamiento necesario para delimitar el área sobre la que el proyector de salida incide, y también preparar un objeto de la clase `PerspectiveChange` que permita cambiar la perspectiva de cualquier imagen desde otros puntos de la aplicación, de forma que la perspectiva de esta coincida con la zona de proyección. Contiene los métodos descritos a continuación.

`PerspectiveDetection(CvCapture* cam, PerspectiveChange* perspCh, CvTrackbarCallback cbPerspThres, CvTrackbarCallback cbPerspWindow)`

Constructor de la clase. Se encarga de inicializar las constantes, la interfaz gráfica de ventanas, llamando al método `createWindows`, y de obtener una primera imagen con el dispositivo de entrada, llamando al método `getNewImage`.
Los parámetros de entrada son un puntero al objeto `CvCapture` correspondiente a la cámara de vídeo, en caso de que esta exista, un puntero a un objeto `PerspectiveChange` donde se almacenará el cambio de perspectiva calculado con el resto de métodos que se ejecuten en la clase, y los métodos callback correspondientes a los cambios en los umbrales y a los cambios en la ventana seleccionada para mostrar.

`virtual ~PerspectiveDetection()`

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

`void createWindows()`

Método que crea las ventanas de Menú y de Umbrales en las que se mostrarán las opciones y variables de la ejecución. Para ello utilizará los métodos `cvNamedWindow`, `cvResizeWindow` y `cvMoveWindow`. Las ventanas creadas son las siguientes:

- Thresholds -> muestra los umbrales utilizados en el procesamiento de esta fase, para que puedan ser modificados dinámicamente. Cualquier actualización en estos provoca un reprocesado mediante el método callback correspondiente.
- Menu → muestra el menú de esta fase de la aplicación, es decir, qué teclas ejecutan qué acciones. Es una imagen jpg que se lee desde un archivo y se muestra directamente. Esta ventana contiene una barra deslizante que

permite señalar qué imagen del estado del procesamiento se muestra en grande. La ventana mostrada se modifica mediante el método callback correspondiente. Las opciones son las siguientes:

0. Initial + Video→ muestra la imagen que se va a procesar en esta fase. Esta imagen será leída desde el archivo “foto_perspectiva.jpg” si no se está utilizando la webcam, o tomada directamente desde la webcam en el instante seleccionado si sí. Si la captura de la webcam ha sido inicializada, se mostrará también una ventana Video donde se podrá ver la imagen en tiempo real capturada por la webcam.
1. Filter → muestra la imagen bicolor tras el filtrado blanco/negro. La barra de deslizamiento “Binary Thres”, de la ventana de umbrales, permite modificar el valor del umbral del filtro de forma dinámica.
2. Contours → muestra el contorno detectado a partir de la imagen filtrada, superpuesto con la imagen inicial. Las barras de deslizamiento “Min Area” y “Max Area” de la ventana de umbrales permiten modificar de forma dinámica los valores de área mínima y área máxima del contorno seleccionado.
3. Polygon → muestra el polígono del contorno de la ventana anterior formado por las esquinas detectadas, también superpuesto con la imagen inicial.
4. Perspective → muestra el resultado de la transformación de perspectiva seleccionada, utilizando para ello la imagen inicial.

`void detect()`

Método que contiene el ciclo de ejecución de esta fase, esperando la pulsación de una tecla con una llamada a `cvWaitKey` y realizando la acción apropiada según la tecla pulsada. Además, actualiza la imagen que muestra la ventana de vídeo en caso de que se esté utilizando la webcam.

Las acciones asociadas a las distintas teclas que pueden ser pulsadas son las siguientes:

- 1 → obtiene una nueva imagen mediante el método `getNewImage` y la

procesa mediante el método `processImage`.

- 2 → almacena el área de proyección que se ha detectado con el procesamiento actual en el objeto `PerspectiveChange` y muestra la transformación de perspectiva final llamando al método generatePerspective.
- Intro → termina la fase actual y pasa a la siguiente fase con la información del cambio de perspectiva en el objeto `PerspectiveChange`.
- Esc → sale de la aplicación.

`void getNewImage()`

Método que obtiene una nueva imagen a procesar. En caso de que la captura de vídeo no sea `NULL`, la imagen debe ser tomada de la webcam mediante la función `cvQueryFrame`. En caso de que lo sea, la imagen debe ser tomada del archivo "foto_perspectiva.jpg" mediante la función `cvLoadImage`. En ambos casos se obtendrá la imagen en escala de grises, utilizando la transformación `cvCvtColor` con el flag `CV_BGR2GRAY` en el primer caso, y utilizando directamente el flag `CV_LOAD_IMAGE_GRAYSCALE` en el segundo. Después se realizará el calibrado de la imagen a partir de los coeficientes intrínsecos y de distorsión del dispositivo de entrada. Para ello se creará un objeto de tipo `Calibration` y se llamará al método `undistort` de este.

Por último, el método `getNewImage` reseteará la información del objeto `PerspectiveChange` para que permita almacenar la nueva perspectiva obtenida del procesamiento posterior de la nueva imagen.

Finalmente, la imagen obtenida se almacena en la variable `imgIni` y se muestra en la ventana `Initial`.

`void processImage()`

Método que procesa la imagen obtenida para extraer de ella la información del área de proyección.
Para ello inicialmente le aplica un filtro binario mediante la función `cvThreshold`, utilizando el flag `CV_THRESH_BINARY` y el valor de umbral en el que esté situada la barra de desplazamiento. El resultado se mostrará en la ventana Filter.
A continuación se utiliza la función `cvFindContours` para detectar los contornos sobre esta imagen binaria, filtrando dichos contornos a partir de su área según el valor de área mínima y área máxima que estén fijados en las barras de desplazamiento. El resultado se muestra en la ventana `Contours`, superpuesto a la imagen inicial, utilizando la función `cvDrawContours`.
Después se toma el contorno detectado y se obtienen las esquinas de dicho contorno. El polígono formado por dichas esquinas se muestra en la ventana `Polygon`.
Por último se va a almacenar la información de las esquinas del polígono obtenido en el objeto `PerspectiveChange`, llamando al método `addSrcQuad` de este para cada uno de los cuatro puntos.

`void generatePerspective()`

Método que, a partir de las esquinas detectadas en el procesamiento, calcula la matriz de transformación que utilizará el objeto `PerspectiveChange` y la aplica a la imagen que se ha utilizado en esta fase.
Para ello llama a los métodos `generateWarpMatrix` y `changePerspective` de la clase `PerspectiveChange`, y muestra la imagen resultado que genera este último en la ventana Perspective.

4.3.2 PerspectiveChange

Clase que encapsula la información y el procesamiento necesario para transformar la perspectiva de una imagen. Para ello consta de dos matrices de 4 puntos,

`srcQuad` y `dstQuad`, que almacenan las esquinas de la imagen original y la imagen que se desea obtener, de forma que el primer punto de `srcQuad` se correspondería con el primero de `dstQuad`, el segundo con el segundo, y así sucesivamente. La primera de estas matrices se rellena desde la clase `PerspectiveDetection` y la segunda es inicializada coincidiendo con las esquinas globales de la imagen. A partir de ellas se obtiene la matriz de transformación que servirá para cambiar posteriormente la perspectiva de cualquier imagen que se especifique.

Los métodos de esta clase se describen a continuación.

```
PerspectiveChange()
```

Constructor de la clase. Resetea la matriz `srcQuad` mediante una llamada al método `resetSrcQuad`, y reserva espacio en memoria para la matriz de transformación.

```
virtual ~PerspectiveChange()
```

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

```
void resetSrcQuad()
```

Método que resetea la matriz srcQuad, la cual almacena los puntos de las esquinas de la imagen origen a la hora de realizar una transformación de perspectiva. Estos valores se inicializan a -1 para poder saber que aún no han sido fijados.

```
void addSrcQuad(int x, int y)
```

Método que almacena el punto (x, y) en la primera posición de la matriz `srcQuad`

que aún no se haya utilizado.

`void updateImage(IplImage* image)`

Método que almacena la imagen que se desea transformar, y además inicializa los valores de la matriz `dstQuad` de forma que coincidan con las esquinas globales de dicha imagen.

`void generateWarpMatrix()`

Método que ordena la matriz de puntos `srcQuad` mediante una llamada a `orderSrcQuad`, para a continuación generar la matriz de transformación llamando a la función `cvGetPerspectiveTransform`.

`void orderSrcQuad(CvPoint2D32f* orderedQuad)`

Método que ordena los valores introducidos en la matriz `srcQuad`, colocando inicialmente la esquina superior izquierda, a continuación la superior derecha, luego la inferior izquierda y por último la inferior derecha. Para saber qué punto se corresponde con cada una de estas esquinas, se comparan sus valores de `x` e `y` con el valor medio de estas coordenadas en los 4 puntos.

`void changePerspective()`

Método que cambia la perspectiva de la imagen almacenada con `updateImage`, utilizando la función `cvWarpPerspective` y la matriz de transformación obtenida en la llamada a `generateWarpMatrix`. La imagen resultante se almacena en la variable dst, para que pueda ser accedida por la línea de código que haya llamado al método.

4.4 Detección de los contornos de los cd's

Esta fase se basará en las clases `EllipsesDetection`, `OneMethodEllipsesDetection`, `Hull` y `Ellipse`, y también se utilizarán las clases `Calibration` a la hora de corregir la distorsión de la imagen de entrada, y la clase `PerspectiveChange` a la hora de modificar la perspectiva para que coincida con el área de proyección, las cuales ya han sido descritas en los apartados anteriores.

4.4.1 Ellipse

Esta clase encapsula la información referente a una elipse detectada en la imagen. Las variables que almacena son el punto central, el tamaño de los semiejes y el ángulo de inclinación del eje principal con respecto a la horizontal.
También posee una variable puntero a otro objeto de tipo `Ellipse`, de forma que varias elipses puedan ser almacenadas en forma de secuencia.
Los métodos de los que dispone la clase son los siguientes:

```
Ellipse()
```

Constructor vacío.

```
Ellipse(CvPoint cent, CvSize ax, double ang)
```

Constructor de la clase, que inicializa los valores de los parámetros de la elipse con aquellos recibidos como parámetros de entrada de este método.

```
Ellipse(CvBox2D box)
```

Constructor de la clase, que inicializa los valores de los parámetros de la elipse con aquellos que se especifican en el parámetro de entrada del método, que es de tipo

`CvBox2D`. Este objeto describe el rectángulo que contiene a la elipse que se desea crear. Los valores de centro y ángulo del objeto `CvBox2D` coinciden con los de la elipse, sin embargo los ejes no se corresponden ya que en el primero se utilizan dimensiones completas mientras que en la segunda se almacenan semiejes. Bastará con dividir entre dos las dimensiones laterales del rectángulo recibido como entrada antes de almacenarlas en las variables correspondientes a los semiejes de la elipse.

```
virtual ~Ellipse()
```

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

```
void paintEllipse(IplImage* image, CvScalar colour, int thickness, int lineType)
```

Método que pinta la elipse actual en la imagen que recibe como primer parámetro de entrada, llamando a la función `cvEllipse`. En la llamada a esta función, se utilizan los valores de color, grosor y tipo de línea que también se reciben como parámetros de entrada.

4.4.2 Hull

Clase que encapsula la información de un contorno, almacenando este como una secuencia de puntos. También posee una variable de tipo puntero a otro objeto `Hull`, para permitir encadenar objetos de este tipo de forma secuencial. Los métodos que facilita esta clase son los descritos a continuación.

```
Hull()
```

Constructor vacío. Reserva el espacio de memoria necesario para almacenar posteriormente la secuencia de puntos del contorno.

`virtual ~Hull()`

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

`void paintHull(IplImage* image, CvScalar colour, int thickness, int lineType)`

Pinta el contorno sobre la imagen recibida como primer parámetro, utilizando el color, grosor y tipo de línea también recibidos como parámetros de entrada.
Para pintar el contorno, este método recorre la secuencia de puntos extrayéndolos de dos en dos de forma consecutiva, y dibujando una línea entre ambos con la función `cvLine`.

`double pointDistance(CvPoint2D32f pt)`

Este método devuelve el valor de la distancia del punto que recibe como parámetro de entrada al punto más cercano del contorno. Para ello utiliza la función `cvPointPolygonTest`. El valor absoluto del número devuelto indica el número de píxeles de distancia, mientras que el signo indica si este está dentro o fuera del contorno (positivo para dentro, negativo para fuera).

4.4.3 EllipsesDetection

Esta clase encapsula todo el procesamiento necesario para detectar los bordes de los DVDs anclados en la pared, almacenándolos en forma de secuencia de objetos `Ellipse` donde cada elemento se corresponde con el borde un DVD. Dado que el procesamiento de la imagen se bifurca en dos líneas paralelas, cada una de ellas con un filtrado inicial distinto, se utilizarán dos objetos de la clase `OneMethodEllipsesDetection`. Cada uno de ellos encapsulará el procesamiento de cada una de las líneas de procesado, mientras que la clase

`EllipsesDetection` se encargará de integrar ambas con la interfaz gráfica y de efectuar la obtención y pretratamiento de las imágenes captadas por el dispositivo de entrada. La clase contiene los siguientes métodos:

```
EllipsesDetection(CvCapture* cam, PerspectiveChange* perspC, CvMouseCallback callbackEllipse, CvMouseCallback callbackHull, CvTrackbarCallback cbEllipThres, CvTrackbarCallback cbEllipWindow, CvTrackbarCallback cbEllipMethod)
```

Constructor de la clase. Se encarga de inicializar las constantes y variables varias de la clase, y de obtener una primera imagen con el dispositivo de entrada, llamando al método `getNewImage`.
Los parámetros de entrada son un puntero al objeto `CvCapture` correspondiente a la cámara de vídeo, en caso de que esta exista, un puntero a un objeto `PerspectiveChange` donde está almacenada la información del cambio de perspectiva a realizar a las imágenes de entrada, dos métodos callback utilizados para el tratamiento de eventos relativos a las ventanas de contornos y de elipses, y otros dos métodos callback utilizados para detectar la modificación de los umbrales de procesamiento y de la ventana principal a mostrar.

```
virtual ~EllipsesDetection()
```

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

```
void createWindows()
```

Método que crea las ventanas de Menú y de Umbrales en las que se mostrarán las opciones y variables de la ejecución. Para ello utilizará los métodos `cvNamedWindow`, `cvResizeWindow` y `cvMoveWindow`. Las ventanas creadas son las siguientes:

- Thresholds -> muestra los umbrales utilizados en el procesamiento de esta fase, para que puedan ser modificados dinámicamente. Cualquier actualización en estos provoca un reprocesado mediante el método callback correspondiente.
- Menu → muestra el menú de esta fase de la aplicación, es decir, qué teclas ejecutan qué acciones. Es una imagen jpg que se lee desde un archivo y se muestra directamente. Esta ventana contiene una barra deslizante que permite señalar qué imagen del estado del procesamiento se muestra en grande. También contiene una segunda barra deslizante que permite seleccionar el método de procesado que se muestra en la ventana, ya que existen dos ramas de procesado distintas. La ventana mostrada se modifica mediante el método callback correspondiente. Las opciones son las siguientes:
    0. Initial + Video→ muestra la imagen que se va a procesar en esta fase. Esta imagen será leída desde el archivo “foto_cds.jpg” si no se está utilizando la webcam, o tomada directamente desde la webcam en el instante seleccionado si sí. Si la captura de la webcam ha sido inicializada, se mostrará también una ventana Video donde se podrá ver la imagen en tiempo real capturada por la webcam.
    1. Filter → muestra la imagen tras el filtrado ya sea Binario o Canny, según la opción señalada. La barra de deslizamiento “Binary Thres” en el primer caso, o las barras “Low Canny Thres” y “High Canny Thres” en el segundo, todas ellas en la ventana de umbrales, permiten modificar el valor de los umbrales del filtro de forma dinámica.
    2. Contours → muestra los contorno detectados a partir de la imagen filtrada, superpuestos con la imagen inicial. Esta ventana tiene asociado un método de callback que detecta cuando un punto de ella ha sido pulsado con el ratón, para poder seleccionar contornos partidos en dos, con el fin de unirlos en un único contorno.
    3. Ellipses → muestra las elipses aproximadas a partir de los contornos de la ventana anterior, también superpuestas a la imagen inicial. Las barras de deslizamiento “Min Rad” y “Max Rad” de la ventanta de

umbrales permiten modificar de forma dinámica los valores de radio mínima y radio máxima de las elipses mostradas. Esta ventana tiene asociado un método de callback que detecta cuando un punto de ella ha sido pulsado con el ratón, para poder seleccionar las elipses que se han detectado correctamente. Cuando una elipse ha sido seleccionada, esta aparece en un color distinto durante el resto de la ejecución.

4. Mask → muestra la máscara resultado, en la cuál se pueden observar las elipses detectadas que ya han sido seleccionadas para la posterior generación de vídeos.

```
void detect()
```

Método que contiene el ciclo de ejecución de esta fase, esperando la pulsación de una tecla con una llamada a `cvWaitKey` y realizando la acción apropiada según la tecla pulsada. Además, actualiza la imagen que muestra la ventana de vídeo en caso de que se esté utilizando la webcam.

Las acciones asociadas a las distintas teclas que pueden ser pulsadas son las siguientes:

- 1 → obtiene una nueva imagen mediante el método `getNewImage` y la procesa mediante el método `processImage`.
- 2 → borra la última elipse seleccionada de la lista de elipses elegidas, llamando al método `delLastChosenEllip`.
- 3 → borra la lista completa de elipses seleccionadas, llamando al método `delAllChosenEllip.`
- 4 → fusiona dos contornos previamente seleccionados con el ratón en la ventana `Contours`, llamando al método `joinSelectedHulls` del correspondiente objeto de la clase `OneMethodEllipsesDetection.`
- Intro → termina la fase actual y pasa a la siguiente fase con la información del conjunto de elipses seleccionadas en la máscara. Para ejecutar la siguiente fase, crea un objeto de tipo `VideoGenerator` con estas elipses y llama al método `write` de este.

- Esc → sale de la aplicación.

`void getNewImage()`

Método que obtiene una nueva imagen a procesar. En caso de que la captura de cámara sea distinta de `NULL`, la imagen debe ser tomada de la webcam mediante la función `cvQueryFrame`. En caso de que sea `NULL`, la imagen debe ser tomada del archivo “foto_cds.jpg” mediante la función `cvLoadImage`. Después se realizará el calibrado de la imagen a partir de los coeficientes intrínsecos y de distorsión del dispositivo de entrada. Para ello se creará un objeto de tipo `Calibration` y se llamará al método `undistort` de este.
A continuación se realiza el cambio de perspectiva, para que la imagen que se procese coincida con la superficie sobre la que va a proyectar el proyector de salida. Para esto se utiliza el objeto `PerspectiveChange`, llamando a los métodos `updateImage` y `changePerspective.`
Después se realiza a la imagen un pretratamiento para eliminar aquellos brillos que sean de un color particular mediante la comparación de las componentes R, G y B de la imagen, llamando al método `deleteColorSparkles`.
Finalmente, la imagen obtenida se almacena en la variable `imgIni` y se muestra en la ventana Initial.

`void processImage(bool binary, bool canny)`

Método que procesa la imagen obtenida imgIni para extraer de ella la información de la posición de los bordes de los cd's, utilizando dos líneas de procesamiento paralelas. Los booleanos que recibe como parámetros de entrada indican si se debe ejecutar la línea de filtrado binario y si se debe ejecutar la línea de filtrado Canny respectivamente. Para ello se llamará al método processImage del objeto de la clase OneMethodEllipsesDetection que corresponda.

```
void deleteColorSparkles(IplImage* imgColorIni)
```

Método que elimina los brillos de la superficie de los DVDs que poseen alguna tonalidad de color, aprovechando la comparación de las distintas componentes R, G y B de la imagen.
Para ello inicialmente se divide la imagen en dichas componentes mediante la función `cvSplit`. A continuación se llama al método `processColor` para cada componente, utilizando siempre las otras dos como componentes de comparación. Por último se volverán a unir las nuevas componentes obtenidas, mediante la función `cvMerge`.

```
void processColor( IplImage* imgIni, IplImage* imgOther1, IplImage* imgOther2, IplImage* imgRes )
```

Método que procesa la componente de color de una imagen, que recibe como primer parámetro, comparándola con las otras dos, que recibe como segundo y tercer parámetro de entrada, para reducir los brillos de una única tonalidad.
Para ello recorre la componente píxel por píxel, y compara la intensidad de cada uno con el valor medio de la intensidad del mismo píxel en las otras dos componentes. En caso de que este fuera superior, lo que significaría que hay brillo, sustituiría su valor por el de la media obtenida.

```
void ellipseToMascara()
```

Método que almacena la elipse que ha sido marcada con el ratón sobre las ventanas de elipses detectadas. El almacenamiento se realiza tanto en el archivo de texto ellipses.txt como en la secuencia de elipses seleccionadas que guarda la clase, llamando al método `storeSelectedEllipse`. También se muestra en la ventana Mask y en la ventana de elipses detectadas dicha elipse aparecerá en otro color.

`void storeSelectedEllipse(Ellipse* elip)`

Almacena una nueva elipse en la secuencia de elipses seleccionadas. Estas elipses son las que posteriormente generarán el vídeo final.

`void delLastChosenEllip()`

Elimina la última elipse que ha sido seleccionada, eliminándola de la secuencia de elipses, de la máscara y del archivo de texto.

`void delAllChosenEllip()`

Elimina todas las elipses seleccionadas, reseteando tanto la secuencia como la imagen de máscara y el archivo de texto.

`void paintChosenEllipses()`

Pinta las elipses detectadas que ya han sido seleccionadas, en la imagen de elipses y con un color distinto para que puedan distinguirse.

`void deleteSelectedEllipses()`

Elimina la secuencia de elipses seleccionadas, para el caso en que se desee resetear esta.

4.4.4 OneMethodEllipsesDetection

Clase que encapsula el procesamiento necesario para la detección de los bordes de los DVDs a partir de una imagen pretratada. Esta clase constituye líneas paralelas de procesamiento que se utilizan en la clase `EllipsesDetection`.

Los métodos disponibles en esta clase se describen a continuación.

`OneMethodEllipsesDetection(bool canny, IplImage*  img)`

Constructor de la clase. Recibe como parámetros de entrada un booleano que indica si el método de filtrado es Canny o binario, y una imagen con la que inicializar el procesado.
Se encarga de inicializar las constantes y las variables, llamando también al método `newImage` para almacenar la imagen.

`virtual ~OneMethodEllipsesDetection()`

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

`void newImage(IplImage* in)`

Método que almacena una nueva imagen para su posterior procesado, inicializando también el resto de imágenes que se generarán durante este de forma que tenga el mismo tamaño que la imagen inicial.

`void processImage()`

Método que procesa la imagen almacenada como imagen inicial. Para ello, en primer lugar llamará al método `processImageBinary` o `processImageCanny`, en función de si el objeto constituye la rama de filtrado binario o de filtrado Canny. Finalmente se llamará al método `calculateContours`, que obtendrá los contornos a partir de la imagen resultante del filtrado anterior.

`void processImageBinary()`

Método que realiza el filtrado inicial de la imagen en el caso de la rama binaria.
Inicialmente aplica un filtro binario blanco/negro mediante la función `cvThreshold`, utilizando el flag `CV_THRESH_BINARY` y el valor del umbral

seleccionado en la barra de desplazamiento correspondiente.

A continuación se realizan las operaciones de apertura y cierre, por este orden y con el número de iteraciones fijado por el valor de las barras de desplazamiento. El primer paso para llevar a cabo estas operaciones es generar el kernel a utilizar, que en este caso será una cruz. Para ello se utilizará la función `cvCreateStructuringElementEx` con el flag `CV_SHAPE_CROSS` y un tamaño de 3x3. Con este kernel se llamará a la función `cvMorphologyEx`, primero con el flag `CV_MOP_OPEN` y el número de operaciones de apertura fijado, y posteriormente con el flag `CV_MOP_CLOSE` y el número de operaciones de cierre fijado.

```
void processImageCanny()
```

Método que realiza el filtrado inicial de la imagen en el caso de la rama Canny. Para ello se utiliza la función `cvCanny`, la cual recibe como parámetros los umbrales superior e inferior que indican las barras de desplazamiento de la ventana correspondiente.

```
void calculateContours()
```

Método que calcula los contornos que aparecen en la imagen filtrada, y posteriormente llama a los métodos adecuados para filtrar dichos contornos y convertirlos a elipses.

En primer lugar, este método llama a la función `cvFindContours`, que devuelve una secuencia de contornos detectados en la imagen filtrada. A continuación se llamará a los métodos `reshapeContours` y `filterContours` para filtrar y convertir estos a elipses, y los métodos `paintHulls` y `paintEllipses`, para mostrar los contornos y las elipses detectadas en las ventanas correspondientes, superponiendo estos a la imagen inicial.

`void reshapeContours()`

Método que opera sobre los contornos inicialmente detectados para convertirlos en formas convexas. Para ello recorre la secuencia de contornos y emplea la función `cvConvexHull2` con cada uno de ellos, utilizando el flag `CV_CLOCKWISE`. Con cada uno de los contornos convexos obtenidos se creará un objeto de tipo `Hull`, y estos se enlazaran unos con otros de forma que se obtenga una secuencia de contornos convexos.

`void filterContours()`

Método que convierte la secuencia de contornos de tipo `Hull` en una secuencia de elipses de tipo `Ellipse`, filtrando estas en función de la longitud de su radio.
Para ello recorre la secuencia de elementos `Hull` y aplica a cada uno de ellos la función `cvFitEllipse2.` Esta devuelve un elemento CvBox2D, a partir del cual se creará un objeto del tipo `Ellipse`. Si el radio de esta elipse es mayor que el mínimo y menor que el máximo indicados por las barras de desplazamientos, se enlazará con el resto de elipses, formando una secuencia de elementos de tipo `Ellipse`.

`void paintHulls()`

Método que recorre la secuencia de objetos `Hull` y pinta cada uno de ellos en la imagen de contornos correspondiente llamando al método `paintHull` de esa clase.

`void paintEllipses()`

Método que recorre la secuencia de objetos `Ellipse` y pinta cada uno de ellos en la imagen de elipses correspondiente llamando al método `paintEllipse` de esa clase.

`void addSelectedHull(Hull* h)`

Método que selecciona uno de los objetos `Hull` de la secuencia. Este método es utilizado por la función callback asociada al evento de pulsar con el ratón sobre la ventana donde se muestran los contornos, y sirve para poder seleccionar dos contornos que se deseen fusionar. Si ya hay dos contornos seleccionados a la hora de llamar a este método, estos se borrarán y el nuevo contorno ocupará la posición de primer contorno seleccionado.

`void joinSelectedHulls()`

Método que fusiona los dos objetos `Hull` seleccionados previamente. Para ello se creará inicialmente una secuencia mediante el método `cvCreateSeq`, donde se almacenarán todos los elementos de las secuencias de puntos de ambos objetos `Hull`. Esta secuencia resultante, que es un contorno en sí misma, se procesará mediante una llamada a la función `cvConvexHull2` para obtener un contorno convexo. Con este contorno convexo se creará un nuevo objeto de tipo `Hull`, que almacene la nueva secuencia de puntos. Bastará enlazar el recién creado `Hull` con la secuencia de objetos `Hull`.
Por último, será necesario volver a llamar a los métodos `filterContours` y `paintEllipses`, para terminar el procesado de la imagen con la nueva detección de contornos.

`void clearSelectedHulls()`

Método que borra cualquier objeto `Hull` que esté previamente seleccionado, y repinta los contornos en la imagen correspondiente.

`void deleteHullsAndEllipses()`

Método que elimina las secuencias de objetos `Hull` y de objetos `Ellipse` de la memoria de ejecución.

4.5 Ensamblaje de los vídeos

En esta fase se emplearán las clases `VideoGenerator`, `Video` y `VideoFromFileGenerator`. Así mismo se utilizará la clase `Ellipse` descrita en la fase anterior para almacenar y obtener la información de las elipses sobre las que proyectar los vídeos.

4.5.1 VideoGenerator

Clase que encapsula la generación del vídeo global en el que se integran el conjunto de vídeos que se proyectan sobre la superficie de cada elipse de la secuencia de elipses detectada y generada en la fase anterior, con la configuración de giros y tiempos de reproducción especificada en el archivo de configuración.

```
VideoGenerator(Ellipse* firstEllipse)
```

Constructor de la clase. Recibe como parámetro la secuencia de elipses en las que se deben proyectar los vídeos. Crea el objeto `CvVideoWriter` en el que se van a grabar los frames del vídeo resultado, mediante la función `cvCreateVideoWriter`, utilizando el formato DIVX. También lee el archivo video_config.txt, en el que está almacenada la configuración de reproducción y giros de cada uno de los vídeos.
Por último el método recorre la secuencia de elipses y genera un objeto de la clase Video por cada una de ellas, almacenando en este la información de la elipse y del vídeo que le corresponde. Estos vídeos también se almacenarán en secuencia enlazando cada uno con el siguiente para poder recorrerlos fácilmente.

```
virtual ~VideoGenerator()
```

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un

espacio de memoria previamente.

```
void writeVideo()
```

Método que, para cada frame del vídeo final, recorre la secuencia de objetos Video llamando al método `paintVideo` de estos en caso de que el frame actual se encuentre dentro del rango de reproducción de dicho video.
Cada frame generado se graba en el vídeo de salida mediante la función `cvWriteFrame` del objeto `CvVideoWriter`.

4.5.2 Video

Clase que encapsula la información y procesado relativos al vídeo que se corresponde con un DVD.
Esta clase almacena tanto las coordenadas de la elipse, mediante un objeto de tipo `Ellipse`, como la configuración del vídeo a reproducir leída del archivo de "video_config.txt". Dispone de los siguientes métodos:

```
Video(Ellipse* ellip, char* fileLine)
```

Constructor de la clase. Recibe como parámetros de entrada la elipse sobre la que debe proyectarse el vídeo y la línea referente al presente vídeo, leída del archivo de configuración de vídeos.
Con ayuda de las funciones `strtok` y `atoi`, las cuales fragmentan una cadena de caracteres según un separador y convierten una cadena de caracteres a entero, respectivamente, se extrae la información sobre el vídeo a reproducir de la línea de texto y se almacena en las correspondientes variables. Estas indican el nombre del archivo, los frames de inicio y finalización de la reproducción con respecto al vídeo global, el número de giros a reproducir, y un array de tamaño `num_giros` x 3 donde se almacena la información de cada uno de estos giros (frame de inicio y fin del giro y número de vueltas).
También se inicializa el objeto `CvCaptures` que permitirá la lectura del fichero de

vídeo a ajustar en la elipse mediante una llamada a la función `cvCreateFileCapture`, utilizando el nombre de archivo obtenido de la cadena de caracteres de configuración.

Además se llama al método `getEllipseTable`, el cual almacena la información del perímetro de la elipse en una tabla para su posterior utilización, y el método `getVideoDimensions`, el cual calcula las dimensiones a las que debe reescalarse el vídeo para que se muestre en la elipse cuanta mayor superficie de él sea posible a la par que ninguna zona aparezca vacía al realizar los giros de la imagen.

```
virtual ~Video()
```

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

```
void getEllipseTable()
```

Método que, aplicando las ecuaciones paramétricas de la elipse, recorre los puntos del perímetro de la elipse y los almacena en una tabla para que estén accesibles de forma rápida y sencilla durante el procesado de los vídeos. La tabla tiene tamaño 360 x 1, donde el índice de cada posición indica el ángulo en grados del punto evaluado y el valor almacenado indica la distancia de este punto al centro. Estos dos datos son todo lo que se necesitan conocer de la elipse para saber si un punto cualquiera de la imagen del vídeo cae dentro del perímetro de la elipse, y por tanto debe pintarse, o no.

```
void getVideoDimensions()
```

Método que calcula las dimensiones a las que debe reescalarse el vídeo para que se muestre en la elipse cuanta mayor superficie de él sea posible a la par que ninguna zona aparezca vacía al realizar los giros de la imagen.

El tamaño de la dimensión más corta del vídeo, ya sea altura o anchura, debe

coincidir con la longitud del eje mayor de la elipse para que se cumpla la condición anterior durante los giros. Posteriormente se reescalará la otra dimensión del vídeo de forma proporcional.

Para realizar estos cálculos es necesario conocer las dimesiones originales del vida, las cuales se obtienen llamando a la función `cvGetCaptureProperty`, utilizando los flags `CV_CAP_PROP_FRAME_WIDTH` y `CV_CAP_PROP_FRAME_HEIGHT`.

```
void paintVideo(IplImage* totalFrame)
```

Método que, para un frame obtenido del vídeo asociado a ese objeto, pinta en el frame del vídeo final los píxeles pertenecientes a ese vídeo original que caigan dentro del perímetro de la elipse.

Inicialmente calcula cuál es el giro que debe mostrar el frame actual en el vídeo final mediante una llamada al método `getRotationAngle.` Con el valor obtenido y la ayuda de las funciones `cv2DRotationMatrix` y `cvWarpAffine`, se generará la matriz de transformación que permita la rotación y se aplicará dicha transformación a la imagen del frame actual.

A continuación se recorrerán los píxeles de la imagen ya rotada y, cuando un píxel caiga dentro del área de la elipse, este se pintará con el mismo valor en la posición adecuada de la imagen del frame del vídeo final. Para saber si un píxel cae dentro de perímetro de la elipse se utilizará el método `isInside` de esta misma clase.

```
int getAngle(double x, double y)
```

Método que devuelve el ángulo en el que se encuentra situado el punto (x, y) recibido como parámetro de entrada, en grados. Este es obtenido realizando la arcotangente de la división de y entre x, ajustando el valor obtenido en función del cuadrante en el que se encuentre el punto y teniendo en cuenta las singularidades producidas cuando x = 0.

`bool isInside(int x, int y)`

Método que devuelve `true` si el punto recibido como parámetro de entrada se encuentra dentro del perímetro de la elipse, y `false` si no.
Para conocer esto se utiliza la tabla de puntos de la elipse calculada en el constructor de la clase. Basta calcular el ángulo y la distancia al centro de la elipse del punto recibido como parámetro, y comparar esta última con el valor almacenado en la tabla para el mismo valor de ángulo. Si la distancia del punto es mayor que la de la tabla, el punto se encuentra fuera, mientras que si es menor, el punto se encuentra dentro de la elipse.

`double getRotationAngle()`

Método que obtiene el ángulo en grados que se debe rotar el frame actual del vídeo antes de copiar sus píxeles al frame del vídeo final. Para calcular este se tiene en cuenta la información almacenada en la matriz de giros en el constructor de la clase. Cuando el frame actual esté dentro del momento de giro definido por los instantes de inicio y fin, la velocidad de giro angular será la división de 360*`num_vueltas` entre la duración del giro. A partir de esta velocidad se obtiene el giro en el instante actual multiplicándola por el número de frames transcurridos desde que se inició dicho giro.

4.5.3 VideoFromFileGenerator

Clase que encapsula la generación del vídeo partiendo de un archivo donde están almacenadas las elipses, llamado ellipses.txt, en lugar de de una secuencia de objetos `Ellipse`. Esto es útil a la hora de generar vídeos donde los DVDs sobre los que se desea proyectar no han sido modificados desde la última ejecución y por tanto ya han sido detectados y almacenados en tal archivo de texto. Esta clase posee una variable `VideoGenerator` que realiza propiamente la generación del vídeo.

`void writeVideo()`

Método que, para cada frame del vídeo final, recorre la secuencia de objetos `Video` llamando al método `paintVideo` de estos en caso de que el frame actual se encuentre dentro del rango de reproducción de dicho video.
Cada frame generado se graba en el vídeo de salida mediante la función `cvWriteFrame` del objeto `CvVideoWriter`.

`VideoFromFileGenerator()`

Constructor de la clase. Inicialmente lee desde el archivo “ellipses.txt” la descripción del conjunto de elipses detectadas mediante un flujo de tipo `ifstream`. A partir de esta información, se genera la secuencia de objetos `Ellipse` correspondiente, para, a partir de ella, crear un objeto de tipo `VideoGenerator`. Las funciones `strtok`, `atoi` y `atof` facilitarán la lectura y conversión de las líneas del archivo.

`virtual ~VideoFromFileGenerator()`

Destructor de la clase. Libera aquellos objetos para los que se ha reservado un espacio de memoria previamente.

`void writeVideo()`

Método que genera y graba el conjunto de frames del vídeo final en el objeto `CvVideoWriter`. Para ello llama al método `writeVideo` de la clase `VideoGenerator` contenida en esta clase.

4.6 Funciones callback

En el proyecto se han implementado siete métodos callback, incluidos en el mismo archivo donde se encuentra el método `main` del programa, ya que son llamados

por los eventos de ratón y de cambio en barras deslizantes, y por tanto no pueden situarse dentro de ninguna clase.

`mouseCallbackHull`

Este método es ejecutado cuando ocurre el evento de pulsación de botón izquierdo sobre una de las ventanas “Contours”, las cuales muestran los contornos detectados en la fase III. Permite seleccionar un contorno detectado, con el objetivo de poder fusionarlo con otro.
Para ello recorre la lista de contornos (secuencia de objetos de tipo `Hull`), y para cada uno de ellos llama al método `pointDistance` de esta clase para calcular la distancia del contorno al punto donde se pulso sobre la imagen. Se escoge el contorno más cercano, guardándolo en el objeto de tipo `EllipsesDetection` mediante el método `addSelectedHull` de este, y repintando la imagen de forma que el contorno seleccionado aparezca en negro.

`mouseCallbackEllipse`

Este método es ejecutado cuando ocurre el evento de pulsación de botón izquierdo sobre una de las ventanas “Ellipses”, las cuales muestran las elipses detectados en la fase III. Permite seleccionar una elipse con el objetivo de almacenarla en la máscara definitiva que generará los vídeos.
Para ello recorre la lista de elipses (secuencia de objetos de tipo `Ellipse`), y para cada uno de ellos calcula la distancia del centro de la elipse al punto donde se pulso sobre la imagen. Se escoge la elipse con el centro más cercano, guardándola en el objeto de tipo `EllipsesDetection` en la variable correspondiente, y repintando la imagen de forma que la elipse seleccionada aparezca en cyan.

`void cbTbPerspThres(int position)`

Método llamado al detectarse un cambio en la posición de alguna de las barras de

deslizamiento que fijan los umbrales en el procesamiento de la clase `PerspectiveDetection`. El parámetro de entrada informa de la nueva posición de la barra. El método se encarga de reprocesar la imagen inicial con los nuevos valores de umbral, y mostrar el resultado.

```
void cbTbPerspWindow(int position)
```

Método llamado al detectarse un cambio en la posición de la barra de deslizamiento que indica qué ventana debe mostrarse durante la fase II del sistema. El parámetro de entrada informa de la nueva posición de la barra. El método se encarga de cambiar la imagen principal a mostrar.

```
void cbTbEllipThres(int position)
```

Método llamado al detectarse un cambio en la posición de alguna de las barras de deslizamiento que fijan los umbrales en el procesamiento de la clase `EllipsesDetection`. El parámetro de entrada informa de la nueva posición de la barra. El método se encarga de reprocesar la imagen inicial con los nuevos valores de umbral, y mostrar el resultado.

```
void cbTbEllipWindow(int position)
```

Método llamado al detectarse un cambio en la posición de la barra de deslizamiento que indica qué ventana debe mostrarse durante la fase III del sistema. El parámetro de entrada informa de la nueva posición de la barra. El método se encarga de cambiar la imagen principal a mostrar.

```
void cbTbPerspMethod(int position)
```

Método llamado al detectarse un cambio en la posición de la barra de deslizamiento que indica qué método de filtrado debe mostrarse durante la fase III

del sistema. El parámetro de entrada informa de la nueva posición de la barra. El método se encarga de cambiar la imagen principal a mostrar.

5 MANUAL DE USUARIO

Al comenzar la ejecución del software, el usuario puede escoger entre tres opciones iniciales, según muestra el menú siguiente.

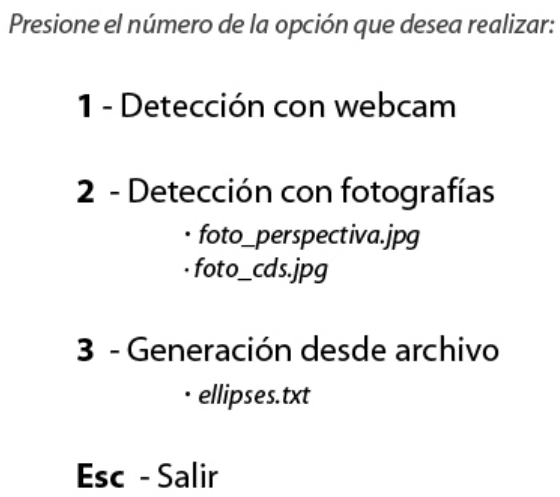

Figura 4.1 – Menú inicial

Pulsando 1, se inicia el proceso de detección utilizando la webcam como dispositivo de entrada al sistema.
Pulsando 2, se inicial el proceso de detección utilizando las fotografías “foto_perspectiva.jpg” y “foto_cds.jpg” como entrada al sistema.
Pulsando 3, se genera el montaje del vídeo final, tomando la información de la posición de los DVDs, almacenada en una ejecución anterior del programa, del archivo “ellipses.txt”.

Si se han seleccionado las opciones 1 ó 2, se pasa a la fase II del sistema, en la cuál se realiza la detección del área de proyección. El aspecto de la pantalla durante esta fase es la mostrada en la figura.

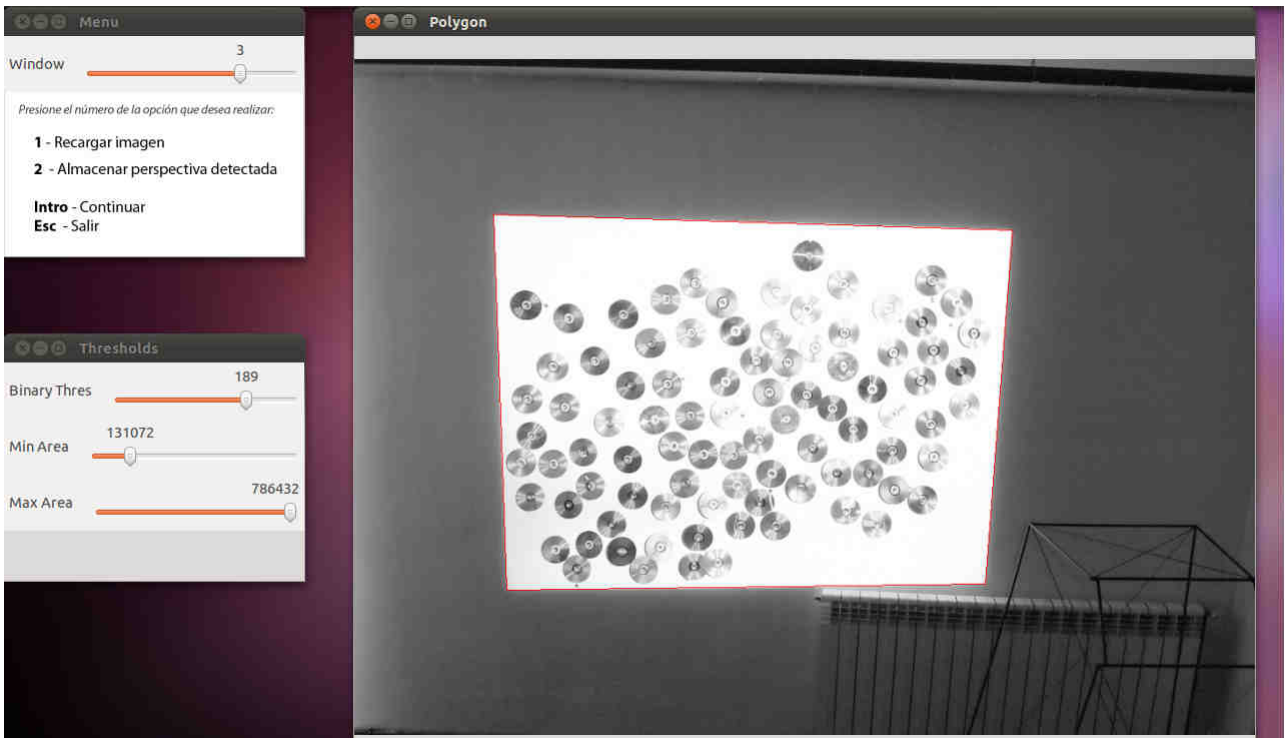

Figura 4.2 – Interfaz de la detección de perspectiva

La barra deslizante de la ventana “Menu”, arriba a la izquierda, permite al usuario moverse por las imágenes que muestran las partes del proceso de detección. Así la posición 0 de esta barra muestra la imagen inicial, la posición 1 tras la aplicación del filtro binario, la posición 2 tras la detección de contornos, la posición 3 el polígono que forman las cuatro esquinas detectadas en el contorno, y la posición 4 la imagen inicial transformada, de forma que sólo se observe el área de proyección.

Cuando se ha accedido por la opción 1 del menú inicial, en la posición 0 de la barra de deslizamiento de las ventanas también se observa una pequeña ventana que muestra la imagen en tiempo real que está captando la webcam. Así se permite seleccionar el instante en el que se desea tomar la imagen de esta para su procesamiento, pulsando la tecla 1.

En la ventana “Thresholds”, abajo a la izquierda, se permite modificar las variables que afectan al procesamiento de la detección. Estas variables se modifican deslizando las barras correspondientes. La primera de estas barras

define el umbral del filtro binario de la ventana 1, mientras que las otras dos fijan el área mínima y máxima que debe tener el contorno seleccionado del conjunto de contornos que se muestran en la ventana 2.

Cuando el polígono que aparece en la ventana 3 coincide correctamente con el área de proyección, el usuario debe pulsar la tecla 2 para seleccionar dicha detección de área y procesar la transformación de la imagen inicial. En ese momento se mostrará la ventana 4 con la imagen transformada, y se podrá pulsar Intro para pasar a la fase III del programa.

El aspecto del sistema durante la fase III es el mostrado a continuación.

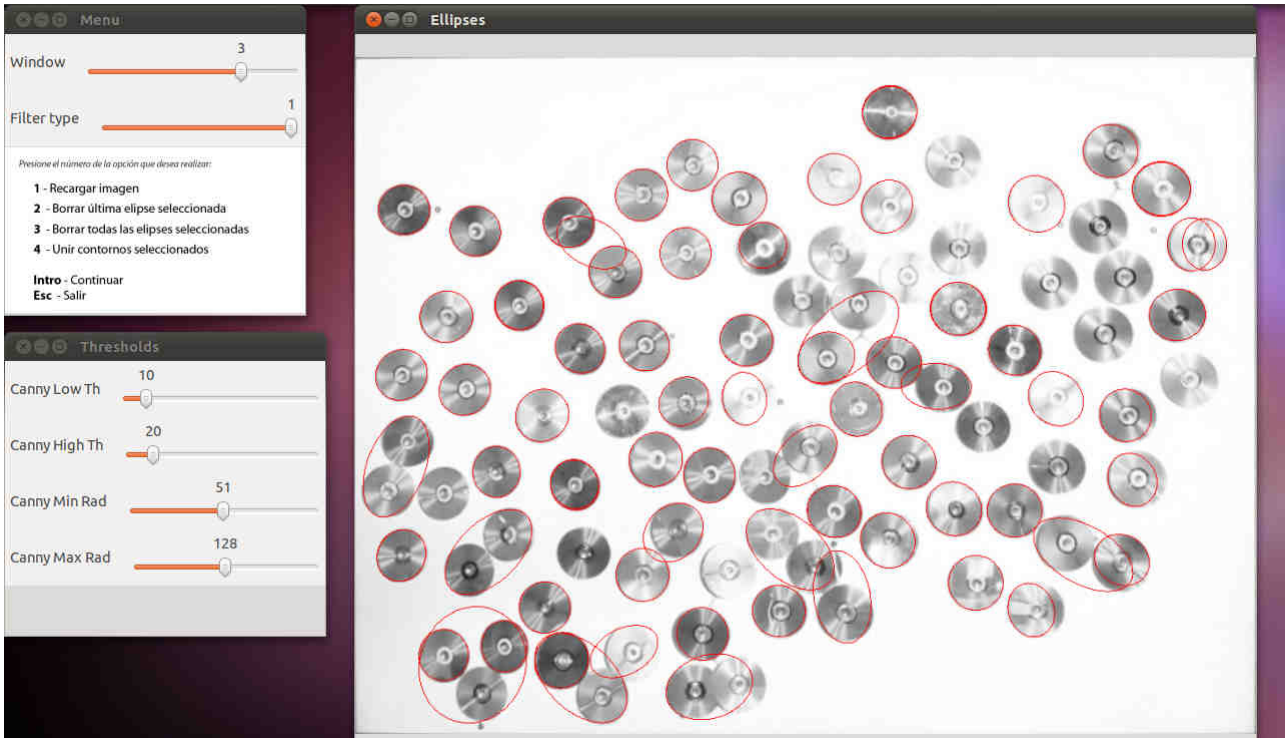

Figura 4.3 – Interfaz de la detección de DVDs

De igual manera, la primera barra deslizante de la ventana “Menu”, arriba a la izquierda, permite al usuario moverse por las imágenes que muestran las partes del proceso de detección, en este caso de los DVDs. Así la posición 0 de esta barra muestra la imagen inicial (imagen inicial + vídeo en el caso de que se tomase la opción 1 en el menú inicial), la posición 1 tras la aplicación del filtro que corresponda, la posición 2 tras la detección de contornos, la posición 3 la

aproximación de esos contornos a elipses, y la posición 4 la máscara que forman las elipses seleccionadas, sobre la cuál se montarán los vídeos en la fase final.

En esta fase III existen dos ramas de procesado, las cuales se ejecutan en paralelo. Cada una de ellas corresponde a un método de filtrado, y se puede escoger cuál es la opción que muestran las imágenes mediante la segunda barra de desplazamiento de la ventana “Menu”. La posición 0 corresponde con el filtrado binario, mientras que la posición 1 corresponde con el filtrado Canny.

En la ventana “Thresholds”, abajo a la izquierda, se permite modificar las variables que afectan al procesamiento de la detección. Estas variables dependen de la opción de filtro seleccionada, y se modifican deslizando las barras correspondientes. En el caso del filtrado binario, la primera de estas barras define el umbral de dicho filtro binario mostrado en la ventana 1, mientras que las otras dos fijan el radio mínimo y máximo que debe tener una elipse para mostrarse en la ventana 3. En el caso del filtrado Canny, las dos primeras barras definen los umbrales superior e inferior de dicho filtro, mostrado en la ventana 1, mientras que las otras dos fijan igualmente el radio mínimo y máximo que debe tener una elipse para mostrarse en la ventana 3.

En la ventana 2, donde se muestran los contornos detectados, existe la posibilidad de unir dos de estos contornos en uno solo. Para ello basta con seleccionarlos pulsando sobre ellos con el ratón. Estos aparecerán resaltados en la imagen con un color negro. Pulsando la tecla 4 tras haberlos marcados, estos contornos pasarán a ser uno.

Para seleccionar las elipses que se han detectado correctamente y sobre las que se desean montar los vídeos, el usuario debe pulsar sobre ellas en la ventana 3. Cada vez que pulse sobre una elipse, ésta se almacenará como seleccionada y se mostrará en otro color sobre la imagen de elipses, así como en la máscara de la imagen 4. Para detectar todas las elipses correctamente,

debido a la diferencia de tonos y contrastes que estas muestran, el usuario debe ir modificando los umbrales de la ventana “Thresholds” e ir seleccionando elipses en el momento en el que note que cada una de ellas está correctamente detectada. Se puede borrar la última elipse seleccionada pulsando la tecla 2, o todas ellas pulsando la tecla 3.

Cuando todas las elipses hayan sido seleccionadas, pulsando Intro se llevará a cabo el montaje del vídeo final, a partir de la configuración de vídeos guardada en el archivo “video_config.txt”. Se mostrará el siguiente mensaje hasta que la generación del vídeo haya finalizado.

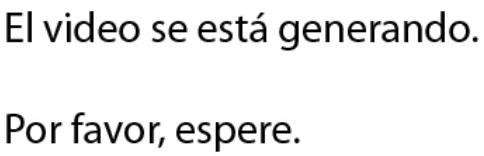

Figura 4.4 – Mensaje durante la generación del vídeo

El vídeo resultado se guardará en formato divx, en la carpeta data/out con el nombre de “video.avi”. Igualmente, cuando en el menú inicial se haya seleccionado la opción 3, el mensaje mostrado en pantalla será el mismo y el vídeo se almacenará este mismo lugar.

6 EVALUACIÓN Y PRUEBAS

El sistema implementado se ha probado con imágenes de entrada de diversos tipos para evaluar su funcionamiento. Se ha comprobado que el resultado de la ejecución depende enormemente de las imágenes captadas por el dispositivo de entrada.

El tipo de luz existente es probablemente la variable que más influye. La detección mejora considerablemente cuando durante la fase de detección de perspectiva la sala se encuentra totalmente a oscuras, de forma que el contraste entre la zona de proyección y el resto de la pared es muy alto, y durante la fase de detección de DVDs existe una luz ambiente no direccional suficiente, de forma que los cd's no tengan sombras y estén bastante iluminados.

La luz direccional sobre los DVDs produce un mayor número de brillos y estos son más intensos, lo cual dificulta la detección durante la fase III. Además produce una sombra importante detrás de cada uno de los DVDs y esta sombra puede perjudicar la correcta detección de los contornos al confundirse con el propio DVD en la imagen.

Sin embargo, en algunos casos no es posible tomar las imágenes de ambas fases con el dispositivo de entrada perfectamente fijo. Es imprescindible que ambas imágenes estén tomadas desde exactamente la misma posición, por lo que cuando esto ocurre es preferible utilizar una única imagen para las dos fases. Por lo explicado anteriormente, la imagen óptima para la detección de la fase II es completamente opuesta a la óptima para la fase III. Sin embargo es preferible usar la imagen de sala oscura con luz bidireccional en ambas fases, ya que la detección de la zona a proyectar es crítica, y aunque este tipo de iluminación complique la detección de los DVDs, con un buen conjunto de combinaciones de umbrales en ambos métodos de detección de la fase III se puede conseguir un nivel adecuado de detección.

También se han realizado pruebas con distintos tamaños de imágenes, llegándose a la conclusión de que la resolución óptima se encuentra en torno a 1024x768 píxeles. Con resoluciones mayores, la calidad de la imagen es mayor y la detección mejora algo, aunque de forma poco significativa. Sin embargo, el tiempo de ejecución del procesado es mucho mayor y el software se ralentiza considerablemente. Con resoluciones inferiores, la calidad de la imagen empeora y la detección es menos precisa y más complicada, ya que el ruido afecta en mayor medida.

Es importante a la hora de tomar las imágenes de entrada que estas se obtengan desde una posición cercana al cañón del proyector de salida. Esto tiene que ver con el hecho de que los DVDs no se encuentran directamente adheridos a la pared, sino que tienen cierta separación y están colocados con cierta inclinación. Al tomar una imagen desde un ángulo significativamente diferente al de aquel con el que incide la luz del proyector, la posición captada de la superficie vista de cada DVD en relación con la zona de proyección que se observa sobre la pared, es distinta a aquella que “ve” el proyector.

Finalmente, en la siguiente imagen se muestra una fotografía de la pared de DVDs durante la proyección del vídeo resultado de la ejecución del software.

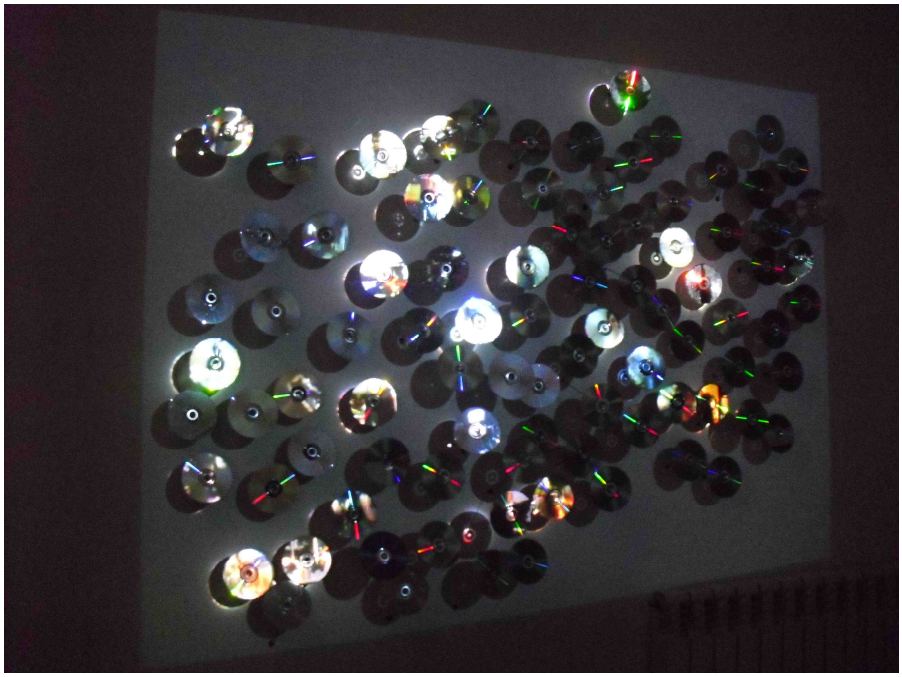

Figura 5.1 – Proyección del vídeo resultado

7 PRESUPUESTO

Este presupuesto recoge las tareas, recursos y costes directos e indirectos derivados del desarrollo del un sistema software de visión por ordenador aplicado a la instalación de una pieza artística, el cual ha sido descrito en el presente proyecto.

7.1 Resumen de recursos y roles

Para el desarrollo del proyecto se tendrán en cuenta los siguientes perfiles:

- 1 Ingeniero Senior (Tutor). Ingeniero experto en sistemas. Se responsabilizará de la supervisión del proyecto y de la revisión de la documentación asociada.
- 1 Analista Programador (Alumno). Se trata de un perfil técnico con conocimientos C++. Se encargará del análisis del problema, del diseño del sistema a implementar, así como del desarrollo y programación del mismo. También se encargará de realizar las pruebas pertinentes a la aplicación.

7.2 Planificación del proyecto

El proyecto aborda el análisis y desarrollo de un sistema que detecta la forma y posición de un conjunto de objetos anclados a una pared y genera un vídeo, a proyectar sobre dicha pared mediante un proyector de salida, de forma que se muestren una serie de vídeos coincidiendo de forma precisa con la superficie de los objetos anclados. Estos vídeos deben seguir un patrón de reproducción configurable, consistente en una composición de giros a distintas velocidades y sentidos, y con diversos instantes de inicio y finalización.

Para la ejecución del proyecto se han tenido en cuenta las siguientes etapas:

- Adquisición de conocimiento. En ella se recopilará y asimilará aquella información relativa al estado del arte y las tecnologías disponibles, así como

una profundización en la tecnología escogida para el problema planteado, en este caso C++ y la librería OpenCV.

- Análisis. Durante esta etapa se realizará el análisis funcional y de requisitos de la aplicación a desarrollar.
- Implementación y codificación. En esta etapa se realizará la implementación de la herramienta. La aplicación consta de 4 fases diferenciadas: calibración, detección de la zona de proyección, detección de los contornos de los objetos y generación del vídeo final a proyectar.
- Pruebas. Se realizarán una serie de pruebas a la aplicación, para comprobar su correcto funcionamiento y detectar posibles errores a corregir.
- Documentación. Creación de la documentación del proyecto.

En la siguiente figura se muestra la duración y el escalonado de las tareas del proyecto mediante un diagrama de Gantt, en el que el tiempo se mide en semanas.

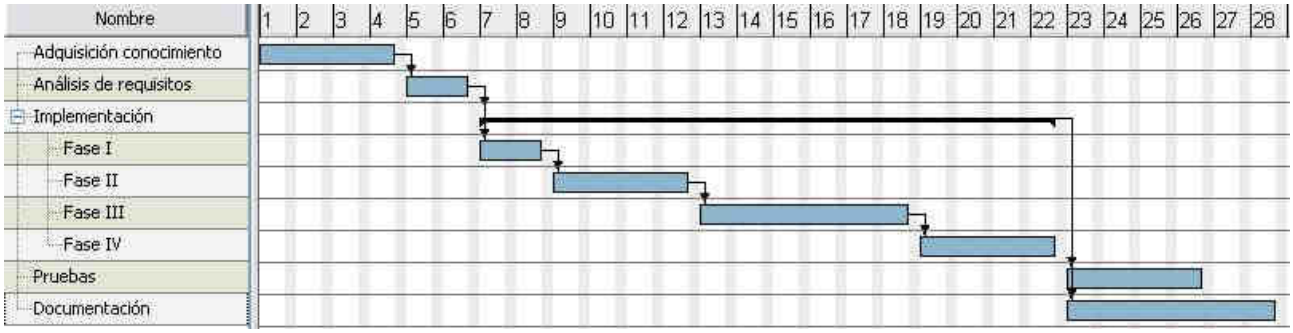

Figura 7.1 – Diagrama de Gantt

7.3 Costes directos

En la siguiente tabla se muestra el resumen de la dedicación de recursos al proyecto desglosado por tareas, para ambos perfiles.

Tarea	Dedicación total (días)	Dedicación Ing. Senior (días)	Dedicación AP (días)
Adquisición conocimiento	20	10%	90%
Análisis de requisitos	10	10%	90%
Implementación fase I	10	0%	100%
Implementación fase II	20	0%	100%
Implementación fase III	30	0%	100%
Implementación fase III	20	0%	100%
Pruebas	10	0%	100%
Documentación	20	20%	80%
TOTAL	140	7	133

Para el cálculo del coste de cada perfil por jornada laboral se ha utilizado una ponderación basada en las tarifas de varias consultoras para los perfiles de Analista-Programador e Ingeniero Senior durante el año 2010. El coste total de los recursos humanos dedicados al proyecto se muestra en la siguiente tabla.

Rol	Jornadas	Coste/Jornada (€)	Coste total (€)
Ingeniero Senior	7	360	2520
Analista-Programador	133	180	23940
TOTAL			26460

Dado que las librerías de software utilizadas para el desarrollo del proyecto son de libre distribución, el único coste adicional al de los recursos humanos es el de un ordenador portátil, el utilizado por el Analista-Programador. El coste estimado del portátil, de marca Dell Latitude D620 con el sistema operativo Ubuntu 10.10 instalado, es de 800€, y la dedicación del mismo al proyecto es del 90%.

Se utilizará la siguiente fórmula para calcular la amortización:

$$\frac{A}{B} \cdot C \cdot D$$

donde

A – Número de meses en que el equipo es utilizado

B – Periodo de depreciación (60 meses)

C – Coste del equipo (sin IVA)

D – % del uso que se dedica al proyecto

Utilizando esta fórmula, y teniendo en cuenta que el proyecto tiene una duración de 7 meses, el coste del equipo es de 84€. Por tanto el total de costes directos del proyecto asciende a 26544€.

7.4 Costes indirectos

Los costes indirectos derivan de la gestión y seguimiento del proyecto, y se estiman a priori en un 20% de los costes directos. Por tanto los costes indirectos ascienden a 2654€.

7.5 Cuadro resumen del presupuesto

Costes directos	26544€
Costes indirectos	2654€
Presupuesto TOTAL del proyecto	**29198€**

8 CONCLUSIONES Y TRABAJOS FUTUROS

8.1 Conclusiones

En este proyecto se ha llevado a cabo el análisis de un problema de aplicación artística, definido en términos de visión por ordenador y manipulación de fotos y vídeos tanto de entrada como de salida.

Se ha podido observar cómo las labores relacionadas con la visión por ordenador son complejas de abordar, aún contando con una librería tan potente y versátil como OpenCV. La cantidad de filtrados, transformaciones y procesados que ha sido necesario aplicar a las imágenes de entrada para conseguir un resultado satisfactorio son amplias y variadas, y aún así el sistema necesita de cierta realimentación manual por parte del usuario para fijar los parámetros que dependen de la claridad y nivel de ruido de las imágenes de entrada al sistema, y para confirmar qué detecciones se ajustan a lo deseado. La automatización de estos procesos manuales habría sido una tarea mucho más ardua y habría complicado ampliamente el código, sobrecargando el propósito central de aplicar de forma sencilla la programación y las nuevas tecnologías a un objetivo artístico concreto.

El principal problema que se ha encontrado durante el diseño del sistema ha ocurrido durante la fase de detección de los bordes de los DVDs, la fase III, y se ha debido a las propiedades reflectantes de la superficie de estos. La detección de esas mismas formas estando los DVDs cubiertos por un material que no reflejase los rayos de luz del proyector, habría sido mucho más sencilla y el resultado final aún más preciso.

El sistema desarrollado será de gran utilidad práctica para el estudio artístico creador de la pieza “Spin”, ya que les facilitará en gran medida el montaje de la misma en distintos entornos. Por tanto es un ejemplo de cómo la introducción de las nuevas tecnologías en el mundo de las artes puede traer grandes ventajas para este tipo de ámbitos.

Acerca de la relación entre el arte y la tecnología, es digno de resaltar que, en el desarrollo de la relación actual entre el arte y la tecnología y la aplicación de esta al arte, uno de los principales impedimentos se debe a la falta de conocimientos tecnológicos por parte de los artistas. La formación de estos acostumbra a ser poco técnica, necesitando de la colaboración de ingenieros y programadores para poder aprovecharse de las herramientas y posibilidades que la tecnología les brinda.

8.2 Trabajos futuros

El sistema diseñado, aunque ha sido aplicado a la detección de formas elípticas correspondientes a DVDs en posiciones variadas, podría ser aplicado a elementos de cualquier tipo o forma de manera sencilla. La detección de formas sobre las que proyectar determinadas imágenes o vídeos es una aplicación bastante frecuente en instalaciones de arte tecnológico en general, y de forma más concreta podría resultar de utilidad en diversas piezas del estudio artístico con el que se ha colaborado durante el desarrollo del presente proyecto. Por tanto su utilidad no se limita al montaje y configuración de la pieza “Spin” con la que se ha trabajado, sino que abre la puerta a facilitar diversos montajes en un futuro próximo, efectuando mínimas variaciones en el código.

Queda abierta a futuras ampliaciones la posibilidad de llegar a una mayor automatización del proceso de detección. Una opción de automatización podría consistir en realizar un barrido automático de todos los valores que pueden tomar los umbrales que se utilizan durante el procesamiento de las imágenes, tratando de estimar cuáles de estos valores obtienen unos mejores resultados. Para ello sería útil partir de alguna confirmación de detección correcta de alguno de los DVDs por parte del usuario, para realizar la estimación por ejemplo a partir de un radio aproximado de la elipse o de alguna otra variable confirmada.

También cabría la posibilidad de utilizar algoritmos de aprendizaje o redes

neuronales, de forma que cuantos más DVDs correctamente detectados haya seleccionado el usuario, mayor información tendrá el sistema para aprender el patrón aproximado al que responden los DVDs que se pretenden detectar y mejor podrá estimar el valor óptimo de los umbrales de procesamiento.

La automatización absoluta del sistema de detección sería complicada de realizar, ya que las condiciones bajo las que se monta la pieza artística pueden variar ampliamente de una instalación a otra. Los objetos a detectar pueden estar a diferentes distancias, por lo que se desconoce el tamaño de estos. Además la claridad del espacio en el que se sitúa la pieza y la direccionalidad de la luz con la que se ilumina este, así como la calidad del dispositivo de entrada con el que se capten las imágenes de la pieza, puede influir enormemente en el resultado y en el valor de los umbrales óptimos para los distintos procesos que se llevan a cabo en el sistema.

9 BIBLIOGRAFÍA

[Bra08] – Bradsky, G., A. Kaelher - “Learning OpenCV, computer vision with the OpenCV library”. O’Reilly, 2008

[Free67] – Freeman, H., “On the classification of line-drawing data,” Models for the Perception of Speech and Visual Form (pp.408–412), 1967.

[Hur67] – Hurwitz, A., J. P. Citron, J. B. Yeaton - “GRAF: Graphic Additions to FORTRAN”. Spring Joint Computer Conf., 1967

[Klu72] – Klüver, B., J. Martin, B. Rose – “Pavilion. Experiments in art and technology” E. P. Dutton & Co. Inc, 1972

[Nob09] – Noble, J. - “Programming Interactivity”. O’Reilly, 2009

[Rea10] – Reas, C., C. McWilliams - “Form + Code in design art and architecture”. Princeton Architectural Press, 2010

[Sha09] – Shanken, E. A. – “Art and electronic media”. Ed. Phaidon, 2009

[Sua10] – Suárez Guerrini, F., B. Gustavino, M. Noel Correbo, N. Matewecki - “Usos de la ciencia en el arte argentino contemporáneo”. Papers Editores, 2010

[Teh89] – Teh, C. H., R. T. Chin - “On the detection of dominant points on digital curves,” IEEE Transactions on Pattern Analysis and Machine Intelligence 11, 1989

[Tri06] – Tribe, M., R. Jana – “Arte y nuevas tecnologías”. Ed. Taschen, 2006

LINKS

[L1] http://es.wikipedia.org/wiki/Trivium (accedido en Junio 2011)

[L2] http://es.wikipedia.org/wiki/Quadrivium (accedido en Junio 2011)

[L3] http://es.wikipedia.org/wiki/Royal_Society (accedido en Junio 2011)

[L4] http://es.wikipedia.org/wiki/Academia_de_las_Ciencias_francesa (accedido en Junio 2011)

[L5] http://es.wikipedia.org/wiki/Real_Academia_de_Pintura_y_Escultura (accedido en Junio 2011)

[L6] http://es.wikipedia.org/wiki/Royal_Academy (accedido en Junio 2011)

[L7] Wilson, Stephen – "Science and Art – Looking backward / looking forward". Publicación digital. http://userwww.sfsu.edu/~swilson/ (accedido en Junio 2011)

[L8] http://es.wikipedia.org/wiki/Seymour_Papert (accedido en Junio 2011)

[L9] http://en.wikipedia.org/wiki/Cybernetic_Serendipity (accedido en Junio 2011)

[L10] http://www.moma.org/ (accedido en Junio 2011)

[L11] http://en.wikipedia.org/wiki/Jack_Burnham (accedido en Junio 2011)

[L12] http://es.wikipedia.org/wiki/Hans_Haacke (accedido en Junio 2011)

[L13] http://www.ccca.ca/artists/work_detail.html?languagePref=en&mkey=49894&link_id=5442 (accedido en Junio 2011)

[L14] http://www.slideshare.net/hexakali/compressed (accedido en Junio 2011)

[L15] http://en.wikipedia.org/wiki/Mel_Bochner (accedido en Junio 2011)

[L16] http://es.wikipedia.org/wiki/John_Cage (accedido en Junio 2011)

[L17] http://es.wikipedia.org/wiki/Allan_Kaprow (accedido en Junio 2011)

[L18] http://es.wikipedia.org/wiki/Sol_LeWitt (accedido en Junio 2011)

[L19] http://es.wikipedia.org/wiki/Yoko_Ono (accedido en Junio 2011)

[L20] http://es.wikipedia.org/wiki/La_Monte_Young (accedido en Junio 2011)

[L21] http://www.youtube.com/watch?v=BzB31mD4NmA (accedido en Junio 2011)

[L22] http://www.beflix.com/beflix.php (accedido en Junio 2011)

[L23] http://en.wikipedia.org/wiki/HyperTalk (accedido en Junio 2011)

[L24] http://es.wikipedia.org/wiki/HyperCard (accedido en Junio 2011)

[L25] http://www.adobe.com/products/director/ (accedido en Junio 2011)

[L26] http://processing.org/ (accedido en Junio 2011)

[L27] http://www.arduino.cc/es/ (accedido en Junio 2011)

[L28] http://www.openframeworks.cc/ (accedido en Junio 2011)

[L29] http://opencv.willowgarage.com/wiki/ (accedido en Junio 2011)

[L30] http://maps.google.es/intl/es/help/maps/streetview/ (accedido en Junio 2011)

[L31] http://es.wikipedia.org/wiki/Elipse (accedido en Junio 2011)

[L32] http://www.danielcanogar.com/ (accedido en Junio 2011)