Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Diseño y realización de un sistema de realidad aumentada para dispositivos de sobremesa

Autor: Jaime Palomo Iranzo
Tutor: Manuel Ángel Perales Esteve

Dpto. Teoría de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Diseño y realización de un sistema de realidad aumentada para dispositivos de sobremesa

Autor:
Jaime Palomo Iranzo

Tutor:
Manuel Ángel Perales Esteve
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Diseño y realización de un sistema de realidad aumentada para dispositivos de sobremesa

Autor: Jaime Palomo Iranzo

Tutor: Manuel Ángel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mis maestros, por darme los conocimientos para abrirme camino en la vida.

A mis amigos y compañeros, por apoyarme y ayudarme cuando lo he necesitado.

Y a mis padres, por haberme hecho ser quien soy.

Resumen

Las tecnologías de realidad virtual (VR) y realidad aumentada (AR) comenzaron a desarrollarse conjuntamente en la década de 1970, aunque su popularidad, especialmente la de la VR, se ha disparado en los últimos años, favoreciéndose de la aparición de dispositivos procesadores de tamaño cada vez más reducido y pantallas de mayor resolución.

La principal diferencia entre ambas tecnologías consiste en que, mientras la VR busca sustituir el entorno del usuario, o al menos la percepción de uno de sus sentidos, normalmente la vista, por un entorno puramente virtual, la AR busca integrar elementos virtuales en el entorno real del usuario.

El objetivo de este proyecto es desarrollar una rama de la AR que fue ignorada en favor de una más vistosa VR, pero que podría presentar mejores prestaciones en determinados ámbitos, especialmente profesionales, y al mismo tiempo ofrecer un menor coste económico.

Abstract

Virtual reality (VR) and augmented reality (AR) technologies began developing together in the 1970s, although their popularity, especially VR, has skyrocketed in recent years, benefiting from the emergence of increasingly smaller size of processing devices and higher resolution screens.

The main difference between both technologies is that, while VR seeks to replace the user's environment, or at least the perception of one of his senses, usually sight, with a purely virtual environment, AR seeks to integrate virtual elements into the real environment around the user.

The object of this project is to develop a branch of AR that was ignored in favor of a more attractive VR, but that could present better benefits in certain areas, especially professional, and at the same time offer a lower economic cost.

Índice

Índice de Tablas

Índice de Figuras

Notación

VR	Realidad Virtual (Virtual Reality)
AR	Realidad Aumentada (Augmented Reality)
HUD	Visualizador frontal (Heads Up Display)
RGB	Rojo-Verde-Azul (Red-Green-Blue, sistema de representación de colores)
VM	Máquina virtual (Virtual Machine)

1 INTRODUCCIÓN

"These things are quite improbable, to be sure; but are they impossible?"

(Estas cosas son improbables, claro; pero ¿son imposibles?)

-The Master Key, L. Frank Baum. 1901

La realidad aumentada y la realidad virtual son conceptos estrechamente ligados, especialmente en su origen, y evolucionan en paralelo, aprovechando una los avances de la otra.

La idea de superponer datos al entorno mediante el uso de un dispositivo electrónico aparece por primera vez en una historia corta titulada "la llave maestra", escrita en 1901 por L. Frank Baum, conocido por su novela "el maravilloso mago de Oz".

En esta historia, el protagonista invoca accidentalmente un "Demonio de Electricidad" que le proporciona una serie de artefactos electrónicos entre los que se incluyen unas gafas capaces de revelar la naturaleza de las personas, mostrando una letra sobre su frente, siendo esta considerada la primera referencia a un sistema de realidad aumentada.

Unos 60 años después, entre 1957 y 1962, el pionero y cineasta Morton Heilig construyó el Sensorama, el primer dispositivo de lo que tiempo después se llamaría realidad virtual. Se trataba de un dispositivo mecánico que incluía un display estereoscópico, ventiladores, emisores de olor, un sistema de sonido estéreo y una silla móvil.

No obstante, el usuario de este tipo de tecnologías sería un mero espectador hasta 1975, con la aparición del Videoplace de Myron Krueger, que creaba un entorno bidimensional en el que la silueta del usuario, detectada mediante una cámara, interactuaba con elementos virtuales proyectados en una pantalla, incluyendo en algunas ocasiones la silueta de un segundo usuario en una sala separada. Este sistema fue llamado entonces "realidad artificial".

A lo largo de la década de 1980 aparecen por primera vez los términos de "realidad aumentada", referida a aquellas tecnologías que añaden (o modifican) contenido del mundo real a través de elementos virtuales, y "realidad virtual", que son aquellas tecnologías que sustituyen la percepción de la realidad del usuario por una generada virtualmente.

A pesar de la multitud de tecnologías tanto de realidad aumentada como de realidad virtual desarrolladas desde entonces, se ha observado la ausencia de sistemas de software y hardware abiertos y bajo coste, que facilite el acceso a estas tecnologías a desarrolladores y usuarios en general, especialmente en el ámbito de la realidad aumentada y en dispositivos de control e interacción.

A continuación, se propone un análisis rápido del estado tecnológico actual, acompañado de un breve repaso histórico, y de las principales aplicaciones de ambas tecnologías, centrándose especialmente los métodos de interacción del usuario y los tipos de displays visuales, al ser estos los elementos que se pretenden desarrollar en el presente trabajo. Tras el análisis, se establecen unos objetivos a tener en cuenta durante el desarrollo del dispositivo.

1.1. Realidad virtual

La VR existe desde la aparición del Sensorama de Morton Heilig en 1962.

En 1968, Ivan Sutherland y sus estudiantes desarrollaron la "Espada de Damocles", un dispositivo electrónico considerado el primer headset VR, y llamado así por su aspecto; el usuario debía sostener y mover el headset empleando unas asas situadas a ambos lados del dispositivo, el cual estaba conectado por cable a un ordenador y unido a una estructura rígida, que permitía detectar su posición en todo momento.

Figura 1-1: Ivan Sutherland probando la Espada de Damocles. (Sutherland, 1968)

Entre 1970 y 1990 fue aplicada a un gran número de campos, entre los que se incluye la medicina, la simulación de vuelo o el diseño en la industria automovilística, entre otros, pero los intentos de llevar displays de este tipo al público general para uso personal tuvieron un éxito mucho más moderado, al tratarse de dispositivos con un coste muy elevado.

La década de 1990, no obstante, marcó la primera oleada de dispositivos de consumo basados en VR, principalmente máquinas de videojuegos para salones arcade, tanto headsets como basados en "cabinas" mecánicas o salas completas, aunque también surgieron los primeros headsets y accesorios para consolas y ordenadores personales.

A partir del año 2000, el interés por este tipo de tecnologías se redujo de nuevo, hasta la aparición del headset "Oculus Rift" en el 2010, cuyas versiones más recientes se consideran aún un referente. La aparición de Google Cardboard en 2015, que proporcionó un headset de muy bajo coste, basado en un chasis de cartón que sostiene un par de lentes y el Smartphone del propio usuario, empleado como pantalla, unidad de procesamiento y conjunto de sensores, terminó de avivar el nuevo interés por esta tecnología.

Dado que se pretende desarrollar un sistema para uso personal de bajo coste, el análisis de la VR se centrará en aquellas que emplean headsets, ya que el desarrollo y construcción de una cabina VR suele implicar una inversión mayor, tanto económicamente como en tiempo y espacio, que suele quedar fuera del alcance del usuario medio, y definitivamente fuera de los objetivos del trabajo.

1.1.1 El headset VR

Un headset VR, también conocido como "gafas" o "casco" de realidad virtual, especialmente los dispositivos más modernos, por su forma, consiste en un dispositivo, diseñado para sostenerse sobre la cabeza, que incluye como mínimo una pantalla, habitualmente estereoscópica, y unas lentes que permitan a los ojos del usuario enfocar la imagen de la misma, a pesar de su cercanía.

Este dispositivo suele incluir altavoces estéreo y algún sistema de seguimiento de movimiento (que puede incluir giroscopios, acelerómetros, magnetómetros, cámaras, sensores de luz, etc.). Algunos headset incluyen también un sistema de seguimiento de los ojos del usuario y soporte para dispositivos de interacción.

Su principal uso actualmente es como accesorio para videojuegos, aunque también se emplean en muchos más campos, incluyendo el entrenamiento médico o la cirugía asistida por ordenador. Sus usos profesionales, no obstante, se ven limitados por los riesgos que conlleva un uso prolongado de esta tecnología.

1.1.2 Controladores VR

Es habitual que los headset VR vengan acompañados por dispositivos adicionales destinados a detectar la interacción del usuario. Es habitual el uso de dos mandos inalámbricos, uno para cada mano, dotados de diversos sensores que pueden incluir un sistema de posicionamiento, habitualmente basado en acelerómetros, varios botones, sensores de presión y actuadores de vibración, entre otros.

Estos controladores suelen buscar ayudar a la sensación de inmersión del usuario, transmitiendo los movimientos a un avatar virtual.

1.1.3 Riesgos y limitaciones de la VR

A pesar de su popularidad y su extenso uso, la VR es aún una tecnología en desarrollo, con notables limitaciones e incluso riesgos para la salud de sus usuarios.

El riesgo más obvio relacionado con su uso deriva de su función principal. La realidad virtual basada en headsets anula por completo la capacidad del usuario para ver su entorno de forma directa, por lo que existe riesgo de golpes y lesiones al chocar con los objetos que lo rodean, especialmente al emplearse en videojuegos de acción o que requieran movimientos rápidos. Algunos headsets minimizan el riesgo añadiendo una función de “zona segura”, que permite establecer unos límites a la zona de juego. Acercarse demasiado a los bordes de la zona hace aparecer una “pared” virtual visible para el usuario, y en algunos modelos activa cámaras situadas alrededor del headset para proporcionar una visión del entorno.

Otros riesgos derivan del uso de pantallas lisas, que requieren un enfoque fijo por parte del ojo, junto a la ilusión de profundidad estereoscópica, lo que puede causar problemas para enfocar de nuevo con normalidad tras haber utilizado el dispositivo por un tiempo prolongado.

También se ha observado un efecto llamado en inglés “virtual reality sickness” o “cybersickness”, derivado de contradicciones entre la percepción visual y el resto de sentidos, por ejemplo al encontrarse quieto mientras el entorno virtual se desplaza a gran velocidad o al observarse un determinado retardo entre las acciones del usuario y las del avatar, debido a los tiempos de procesamiento.

Los síntomas de la cybersickness incluyen mareos, nausea, desorientación y migraña, entre otros, y se calcula que afecta a entre un 25 y un 40% de la población, por lo que los esfuerzos de los desarrolladores se centran actualmente en minimizar estos efectos, especialmente frente a usos prolongados.

No se recomienda el uso de esta tecnología por parte de niños y adolescentes, ya que aún se desconocen los efectos a largo plazo que pueda tener, incluyendo posibles efectos en su desarrollo físico y psicológico.

1.2. Realidad aumentada

Aunque el Sensorama y la Espada de Damocles se consideran los inicios de la VR, también sientan las bases para el desarrollo de la tecnología AR. Posteriormente, se desarrollaron nuevas ramas de esta tecnología:

En 1975, Myron Krueger crea el Videoplace, un sistema de realidad aumentada, por entonces llamada realidad artificial, basado en elementos bidimensionales sobre una pantalla, sobre el que se habló en la introducción de este capítulo.

A partir de 1980, se desarrollan los primeros HUD, utilizados ampliamente en aeronáutica, consistentes en una pantalla transparente sobre la que se proyectan los datos, de forma que el usuario no necesita desviar la vista para consultar sus instrumentos. Estos HUD son también precursores de las gafas de realidad virtual (no confundir con los headsets VR opacos), las cuales utilizan el reflejo de una pantalla sobre una lente

transparente para mostrar datos al usuario.

Figura 1-2: HUD empleado en aviación, desde el punto de vista del piloto. (Shawn, 2008)

Poco después, en 1981, Dan Reitan aplica por primera vez la realidad aumentada a la televisión, al superponer imágenes de radares y satélites meteorológicos a un mapa. Variaciones de esta tecnología incluyen los titulares de noticias, subtítulos en directo o trayectorias proyectadas sobre vídeos.

En 1986, Ron Feigenblatt, de IBM, describe la aplicación más común de la AR a día de hoy: el uso de una pequeña pantalla, que se llevase y orientase con la mano, como se hace con los dispositivos smartphones actualmente. Un ejemplo de aplicación muy conocido es el videojuego Pokémon Go, aunque también se consideran realidad aumentada las aplicaciones de navegación GPS, entre otras de uso común.

Centrando el análisis en las tecnologías derivadas del Videoplace, por ser la más cercana al tipo de dispositivo que se pretende diseñar, se encuentran algunos ejemplos en dispositivos de consumo, especialmente en el ámbito de las videoconsolas. El sistema más habitual emplea una pantalla, habitualmente de tv, y sistemas de tracking similares a los empleados por la VR. La antigua silueta 2D del usuario es sustituida en la actualidad por un avatar 3D o por una imagen captada por cámara del entorno del usuario.

1.2.1 Sistemas de seguimiento del usuario

Los sistemas de seguimiento para AR de sobremesa actuales utilizan en su mayoría un mando capaz de realizar el seguimiento por sí mismo o una cámara 3D capaz de realizar reconocimiento de objetos.

El primer ejemplo que gozó de una gran popularidad para el método basado en mando se encuentra en el Wii remote, o Wiimote, el mando original para la consola Wii de Nintendo, lanzada al mercado en 2006, mientras que el segundo método se ve reflejado en el accesorio Kinect de Microsoft, lanzado en 2010 para la consola Xbox de la misma compañía y posteriormente adaptado para PC.

En 2007, Johnny Chung Lee, posteriormente uno de los principales miembros del equipo de desarrollo de Kinect, publicó en su página y en YouTube un video en el que presentaba un software desarrollado por él que aprovechaba los sensores de un wiimote para realizar el seguimiento de la cabeza de un usuario. Este proyecto, junto con el Videoplace, es una de las principales fuentes de inspiración del presente trabajo.

1.2.1.1 Wiimote

Figura 1-3: Wiimote(Dominio público, 2010)

El Wiimote utiliza dos métodos para el seguimiento de la posición, uno basado en un acelerómetro de 6 ejes (y giroscopio, si el mando cuenta con el accesorio "motion plus") y otro basado en una cámara infrarroja.

Existe también un segundo mando, llamado nunchuk, de menor tamaño, acoplable de forma alámbrica al mando principal. Este mando adicional cuenta únicamente con un acelerómetro, idéntico al del wiimote. El nombre de nunchuk deriva del parecido físico entre el conjunto de ambos mandos y el arma nunchaku, a pesar de que el cable entre mandos mide aproximadamente 1m, en lugar de unos centímetros como en el caso del arma.

El sistema basado en acelerómetro y giroscopio del wiimote proporciona datos de velocidad y aceleración instantáneas bastante precisos, pero el cálculo de la posición sufre de una deriva que requiere su recalibrado cada cierto tiempo de uso, especialmente si se realizan movimientos bruscos, al no poderse obtener datos de la velocidad y posición entre muestras.

Para compensar esto, y añadir una forma sencilla de calibración, se emplea el sistema basado en la cámara. Dispositivos anteriores al wiimote ya utilizaban un sensor de luz para detectar la posición de la pantalla respecto al dispositivo, pero el wiimote mejora este sistema añadiendo un nuevo accesorio, la "barra de sensores".

La barra de sensores consiste en una barra de longitud fija, con un grupo de LEDs infrarrojos en cada extremo. Estos LED permiten detectar su posición relativa con precisión gracias a la cámara situada en el extremo del mando, pero el sistema no funciona bien en cuanto al seguimiento de movimiento, ya que tiene un tiempo de muestreo mayor, y la cámara empleada tiene un ángulo de visión muy limitado, para evitar fuentes de ruido.

1.2.1.2 Kinect

Figura 1-4: Kinect (Dominio público, 2011)

Kinect utiliza un sistema de detección de profundidad, basado en una rejilla de luz infrarroja y un sensor de luz estructurada en las primeras versiones, sustituido por un sensor basado en tiempo de vuelo de dicha luz en las siguientes, para la detección de profundidad, separando "objetos" de "fondo".

Una vez separados, y asumiendo que los únicos objetos en movimiento serán los usuarios (hasta 4 por dispositivo), aísla los objetos en movimiento y utiliza un algoritmo de inteligencia artificial para generar un "esqueleto" de 20 puntos, utilizable por el software para determinar las acciones del usuario.

Este sistema proporciona bastante precisión en general a la hora de detectar posiciones, pero es más vulnerable al ruido del entorno que el empleado por el wiimote, además de requerir un procesador más potente. La mayor ventaja, no obstante, es la ausencia total de mandos, captando las acciones de los usuarios de forma directa, lo que elimina el uso de baterías.

El sistema empleado por este dispositivo, no obstante, presenta una versatilidad más allá de la realidad aumentada, existiendo aplicaciones para emplearlo como escáner 3D, algunas de las cuales permiten incluso recuperar la textura del objeto real, al disponer también de una cámara de luz visible.

1.2.1.3 WiiDesktopVR de Johnny Lee

Antes de trabajar para Microsoft en el desarrollo de Kinect, Johnny Chung Lee utilizó un wiimote, conectado a un PC a través de bluetooth, para obtener un dispositivo de realidad aumentada programado en C# similar al que aquí se pretende desarrollar.

El dispositivo consistía en acoplar una versión ligera de la barra de sensores a unas gafas, empleando el wiimote como cámara fija, y aprovechando el software preexistente en el wiimote para obtener los datos de posición, mientras una segunda barra de sensores situada detrás del wiimote "cámara" permitía el uso de un segundo wiimote.

Uno de los objetivos de este trabajo podría verse como obtener una réplica de bajo coste, construidas con elementos más genéricos o baratos, de este mismo sistema.

Como inconveniente, se observa el uso de dos dispositivos alimentados por batería, si bien en un sistema construido expresamente uno de ellos podría ser sustituido por una cámara fija.

Comparando el sistema empleado aquí con el empleado por el Kinect, e ignorando el ángulo de visión de las cámaras, este sistema presenta un menor uso de procesador y una mayor resistencia al ruido, siempre que no se produzca acoplamiento entre ambos wiimotes, por ejemplo sustituyéndolos por sistemas que funcionen a frecuencias distintas.

1.2.2 Ventajas y desventajas frente a la VR y ámbitos de aplicación

La principal ventaja de este tipo de AR frente a la VR supone la eliminación de los inconvenientes de esta última relacionados con el uso de pantallas a escasa distancia de los ojos. El usuario de este tipo de AR puede percibir su entorno, al utilizarse una pantalla convencional, y debe enfocar a distintas distancias con la misma regularidad que el usuario de cualquier tipo de pantalla convencional, lo que elimina los efectos de la cybersickness.

El principal inconveniente es que, incluso si se emplea algún tipo de pantalla con efecto 3D que conserve la ilusión de visión estereoscópica, la limitación del ángulo de visión a la pantalla limita la sensación de inmersión.

En general, la AR parece más apropiada en ámbitos donde resulte útil utilizar material físico además del virtual, como suele ocurrir en ámbitos laborales, mientras la VR es más inmersiva y apropiada en ámbitos lúdicos, como los videojuegos, siempre que se utilice adecuadamente.

1.3. Propuesta de proyecto

Se observa que, a pesar de lo extendidas que se encuentran las tecnologías de realidad aumentada y realidad virtual, y de la existencia de versiones de bajo coste de algunas de sus aplicaciones, no existe un modelo de realidad aumentada para dispositivos de escritorio, de bajo coste y que proporcione todas las prestaciones que

puede brindar esta tecnología, al menos no sin recurrir a soluciones hardware especializadas.

Además, se comprueba que la mayoría de esfuerzos de la realidad virtual, mucho más extendida, se dirigen a dispositivos eminentemente lúdicos, preparados más para el disfrute de videojuegos que para ámbitos laborales o creativos, si bien parece razonable la aplicación de estas tecnologías en estos ámbitos, al permitir la visualización de objetos virtuales, por ejemplo diseños 3D, como si se tratase de objetos físicos.

Por tanto, se propone comprobar la viabilidad de un sistema AR que o bien utilice hardware que ya esté disponible a nivel doméstico o bien sea de fácil adquisición, por su bajo precio o por disponer de otras aplicaciones de uso común. También se contemplará la viabilidad de producir un dispositivo comercializable que integre las características del sistema diseñado, que ofrezca facilidades de uso e instalación al usuario.

Este sistema deberá proporcionar a las aplicaciones asociadas información suficiente para permitir un cierto grado de interacción del usuario, considerándose adecuado, como mínimo, posición de cabeza y mano, aunque la detección de la orientación de las mismas puede ser opcional, dependiendo de las aplicaciones finales.

Además, para que sea realmente útil, debe proporcionarse en un formato para el que sea fácil realizar nuevas aplicaciones software que aprovechen las características del sistema diseñado.

1.3.1 Sistema de seguimiento

Para el sistema de seguimiento se han barajado tres posibilidades básicas:

- Emplear un sistema basado en mandos que realicen por si mismos el seguimiento, procesando o no la información, similar al wiimote, al que en adelante se llamará “mando activo”.
- Emplear un sistema basado en visión artificial, similar al Kinect, al que se llamará “sin mandos”.
- Emplear un híbrido, en el que un sistema de visión capte únicamente un elemento prediseñado, que puede o no realizar otras funciones pero que no posea sensores de posición propios, al que se llamará “mando pasivo”.

Estas posibilidades pueden utilizarse por separado, o combinarse entre sí. Por ejemplo, si la cámara del wiimote estuviera en la barra de sensores y los LED infrarrojos en el wiimote, se estaría empleando un método activo con los acelerómetros y giroscopios del mando, y un método pasivo al detectar el mando con la cámara.

A continuación se presenta cada una de ellas con mayor detalle, y al final se ofrece una tabla comparativa de sus características.

1.3.1.1 Sistema de mando activo

Un sistema de mando activo requiere la instalación de un conjunto de sensores en un dispositivo que pueda ser movido por el usuario, y un método de conexión entre este dispositivo y el dispositivo de sobremesa en el que se aplica la AR.

Al pretender realizar el sistema con un bajo coste y fácilmente replicable por la población, resulta conveniente contemplar los distintos dispositivos que un usuario pueda tener ya con anterioridad al uso de esta tecnología, siendo el Smartphone el único dispositivo suficientemente extendido con un conjunto de sensores y sistemas de envío de información preinstalados.

Quedan descartados inicialmente los mandos fabricados exclusivamente con este fin, al observarse la existencia de alternativas que no requieran la adquisición de este tipo de accesorios y la existencia previa a este trabajo de versiones comerciales de coste relativamente bajo con las que sería difícil competir.

El uso de un Smartphone presenta algunas ventajas e inconvenientes.

Como ventajas, el conjunto de sensores disponibles es lo bastante amplio como para añadir un gran número de posibilidades a la interacción del usuario. La precisión de los acelerómetros y giroscopios actuales es bastante elevada, y la pantalla táctil ofrece al mismo tiempo un display adicional y un método adicional de interacción con el dispositivo. También se observa un gran número de formas habituales de comunicación entre el móvil y un dispositivo de sobremesa.

Además, es posible compartir la información de varios dispositivos simultáneamente, permitiendo el acceso de múltiples usuarios, y el Smartphone puede actuar como una unidad de procesamiento previa, por lo que los

datos enviados requerirían procesado mínimo según su aplicación en el sistema de sobremesa.

Como inconvenientes, este sistema permitiría un único dispositivo por usuario, ya que no es tan habitual la posesión de varios Smartphones, por lo que no se podría hacer seguimiento de cabeza y mano simultáneamente. Además, existen un gran número de aplicaciones AR basadas en Smartphone, por lo que el uso de uno como mando activo sólo traería como novedad la adición de una pantalla de mayor tamaño, lo que implicaría una mayor competencia inicial.

Además, los servicios de difusión de aplicaciones móviles, como la Play Store para Android, implican un coste para el desarrollador, independiente de si la aplicación es gratuita o no.

1.3.1.2 Sistema sin mandos

Un sistema similar al de Kinect requiere como mínimo el uso de una cámara y algún medio para distinguir al usuario de su entorno.

La disponibilidad de una cámara es muy elevada, ya que podrían emplearse tanto webcam como la integrada en un Smartphone, empleando aplicaciones gratuitas que ya existen.

No obstante, el sistema de reconocimiento del usuario es más complejo. Puede emplearse un sistema basado en detección de movimiento que compare fotogramas anteriores con el actual, pero este método es muy sensible al ruido.

Otra opción es un sistema similar al del Kinect, basado en un emisor de luz sincronizado con la cámara, pero se requeriría una cámara de luz estructurada o capaz de medir el tiempo de vuelo de la luz empleada, lo que disminuiría drásticamente la disponibilidad y aumentaría los costes.

Un tercer método podría basarse en la intensidad luminosa de la luz reflejada por una fuente controlada, pero esto se vería afectado enormemente por los materiales del entorno o la ropa del usuario, así como la presencia de otras fuentes de luz, lo cual, dependiendo de la frecuencia empleada, puede tener una disponibilidad menor, contraindicaciones de uso o requerir un entorno concreto, lo que dificulta su instalación.

Además, estos métodos presentan mayor dificultad a la hora de calcular la distancia exacta a la que se encuentra el usuario, a menos que empleen cámaras especiales como las del Kinect.

1.3.1.3 Sistema de mando pasivo

Una opción que facilita la implementación del sistema anterior es añadir elementos conocidos y fácilmente identificables, que actúen como mandos pero no realicen ninguna acción de posicionamiento por sí mismos. Conocer su tamaño puede ayudar a calcular la distancia a partir de su escala en la imagen obtenida.

Además, estos “mandos” no tienen por qué ser dispositivos electrónicos. Por ejemplo, puede utilizarse una cámara de luz visible y una pelota de un color concreto que no se encuentre en el fondo. Utilizar varios elementos de distintos colores con la misma cámara permitiría obtener más información de uno o varios usuarios.

Este sistema, además, permite filtrar la imagen para centrarse únicamente en estos objetos, en lugar de tener que procesar toda la imagen.

1.3.1.4 Elección inicial e idea previa

Como conclusión del análisis previo, se extrae la siguiente tabla cualitativa. Las celdas marcadas en verde son aquellas en las que se considera que el sistema correspondiente presenta ventajas sobre los otros dos, mientras que las marcadas en rojo indican un inconveniente.

La valoración de cada elemento corresponde a una estimación a priori, basada en la información anterior.

Sistema analizado:	Mando activo	Mando pasivo	Sin mando
Disponibilidad	Alta	Alta	Alta/Baja según método
Usuarios máximos por dispositivo	Varios	Varios	Varios
Capacidad de procesado externa	Nativa	Depende de la aplicación	Depende de la aplicación
Capacidad de procesado necesaria	Baja	Media	Alta
Posicionamiento	Relativo	Absoluto	Absoluto
Sensibilidad a ruidos y derivas	Alta	Baja	Alta
Facilidad de instalación	Muy alta	Alta	Baja
Presencia de ejemplos en el mercado	Muy alta	Baja	Muy alta

Tabla 1-1: Análisis de los posibles sistemas de posicionamiento a emplear

Como conclusión, parece razonable comprobar la viabilidad del proyecto utilizando un sistema de mandos pasivos. En concreto, se utilizarán imágenes de luz visible captadas con cámara en conjunto con elementos de los colores y formas adecuadas para facilitar su detección.

1.3.2 Sistema de visualización

Dado que se pretende utilizar el sistema de realidad aumentada con un dispositivo de sobremesa, el sistema de visualización será el del propio dispositivo, normalmente una pantalla fija, como por ejemplo un monitor.

1.3.3 Sistema de procesado

Se dispone para la elaboración del trabajo de una Raspberry Pi, cedida en préstamo por la Universidad de Sevilla. No obstante, se realizará una prueba inicial utilizando el mismo ordenador empleado para el diseño como único dispositivo.

Dependiendo del comportamiento observado con este diseño inicial, se propondrá emplear un dispositivo externo para el procesado de la imagen, liberando esta carga del procesador principal para ejecutar aplicaciones de mayor peso, realizando una prueba empleando la Raspberry como unidad de procesado externa.

También es posible, si el ordenador principal presenta una gran facilidad para superar la prueba propuesta, que se realice una segunda prueba empleando la Raspberry como único dispositivo, al presentar unas características más limitadas, con el fin de probar las limitaciones del algoritmo diseñado.

1.3.4 Software

Dado que se pretende realizar un sistema aprovechable para múltiples aplicaciones, resulta indispensable que los datos proporcionados por el mismo puedan ser leídos e interpretados por el software generado a tal efecto. Para conseguir esto, existen tres soluciones habituales: el uso de un "driver" asociado a un hardware completo, que realice las funciones propias del sistema y envíe los datos a las aplicaciones, habitualmente acompañado de una librería específica; el uso de un protocolo de comunicación entre programas software, donde uno realice el procesamiento y se transfiera al otro, también implementable mediante una librería; o el uso de una librería que integre las funciones directamente en la aplicación final.

Dada la simplicidad de este último método frente a los anteriores, que además puede incluir una mejora de rendimiento al minimizar el número de elementos implicados, se ha optado por esta implementación.

Por otro lado, el lenguaje a emplear deberá ser analizado en mayor profundidad, por lo que se reserva este análisis para el capítulo 3, dedicado al diseño del software.

No obstante, sí que es posible establecer un interés por los lenguajes con un mayor uso en las aplicaciones en las que esta librería pueda resultar de utilidad, especialmente lenguajes multiplataforma, ya que esto podría facilitar el acceso a un mayor número de desarrolladores y usuarios.

Además, se acompañará la librería de una serie de aplicaciones de demostración que aprovechen las funciones desarrolladas, a modo de ejemplo para futuros desarrolladores.

1.3.5 Resumen de objetivos

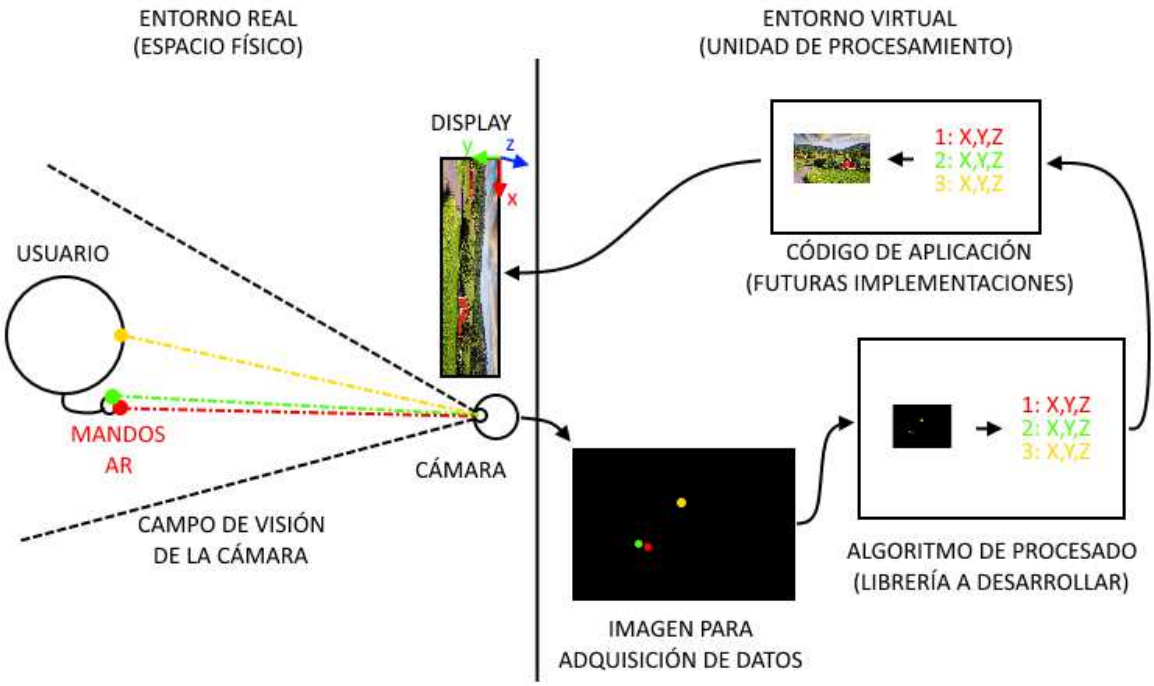

Figura 1-5: Esquema de los elementos del sistema AR a realizar. Elaboración propia.

En base al análisis anterior, es posible concretar una serie de características que tendrá el sistema a diseñar, así como objetivos de diseño. La lista completa puede consultarse con detalle en el Anexo A, pero, a modo de resumen, se establece que:

- Se desea establecer un sistema de bajo coste, fácilmente replicable por futuros usuarios.
- Se empleará un sistema basado en luz visible, objetos prediseñados (mandos) y una cámara.
- Se detectará al menos la posición de la cabeza y una mano respecto a un sistema cartesiano fijo.
- Se realizará el procesado de la información a través del dispositivo de sobremesa o un dispositivo externo. En el esquema se ha representado un entorno virtual que incluiría a ambos, en su caso.

- El sistema podrá emplearse para el desarrollo de nuevas aplicaciones. Para facilitar esto, el cálculo de las coordenadas de posición se implementará en forma de librería, que proporcionará funciones para obtener los valores en una forma aprovechable por el desarrollador.

2 DISEÑO TEÓRICO DEL SISTEMA DE POSICIÓN

"Geometry has two great treasures: one is the theorem of Pythagoras, the other the division of a line into extreme and mean ratio. The first we may compare to a mass of gold, the second we may call a precious jewel."

(La geometría tiene dos grandes tesoros: uno es el teorema de Pitágoras, el otro la división de una línea en razón extrema y media. El primero se puede comparar con un montón de oro, a la segunda se la puede llamar una joya preciosa)

-Atribuida a Kepler en "A brief history of mathematics", de Karl Fink

El único elemento hardware empleado por el sistema que no puede ser considerado preexistente, y que por tanto ha de ser diseñado específicamente, son los mandos pasivos, que deberán permitir un seguimiento adecuado empleando únicamente una cámara.

Dado que se busca un cierto nivel de interacción por parte del usuario, se ha considerado adecuado realizar el seguimiento de, al menos, la cabeza y una mano del mismo. De esta forma, se podrán construir aplicaciones que utilicen información de la perspectiva del observador, y al mismo tiempo le permitan interactuar de forma cómoda.

Además, el dispositivo de la mano deberá permitir detectar algunos gestos, para facilitar esta interacción, por lo que se sugiere la detección de al menos dos dedos, proponiéndose índice y pulgar para esta tarea.

Así, deberán construirse y probarse al menos 3 elementos individuales de detección, cómodos para el usuario que los lleve en la cabeza y la mano, y que no limiten el movimiento. Queda abierta la posibilidad de crear nuevos elementos en un futuro, para aplicaciones distintas a las propuestas en este trabajo pero que aprovechen su tecnología, por ejemplo emulando herramientas físicas que el usuario pueda usar en el entorno virtual como lo haría en un entorno real.

Dado que el dispositivo de detección de los mandos pasivos será una cámara de luz visible, las principales características a analizar serán su color y su forma, al ser estas las que una cámara percibe, además de algún método de sujeción cómodo para el usuario.

Antes de comenzar con el diseño de los mandos, resulta conveniente hacer un repaso al funcionamiento de una cámara, en especial a la proyección empleada para obtener una imagen plana a partir del entorno tridimensional frente al dispositivo, ya que se busca revertir esta proyección para obtener un juego de coordenadas tridimensionales que dependa de la posición del objeto real, a ser posible coincidiendo con las coordenadas reales en algún sistema de referencia fijo.

También es de interés conocer cómo se pueden filtrar los colores para distinguir los objetos del fondo y entre sí, permitiendo un cierto margen de tolerancia que permita seguir los colores aunque se vean expuestos a sombras o a luces ligeramente coloreadas, o si la cámara presenta una ligera desviación en el valor de alguno de los colores.

2.1. Obtención de la posición

El sistema de obtención de imagen de una cámara digital actual se basa en el mismo principio que una cámara oscura o que el ojo humano: se hacen pasar haces de luz por un orificio pequeño, dotado de una lente para ayudar al enfoque, y se hacen impactar sobre una rejilla de sensores capaces de detectar el color de cada uno de los haces.

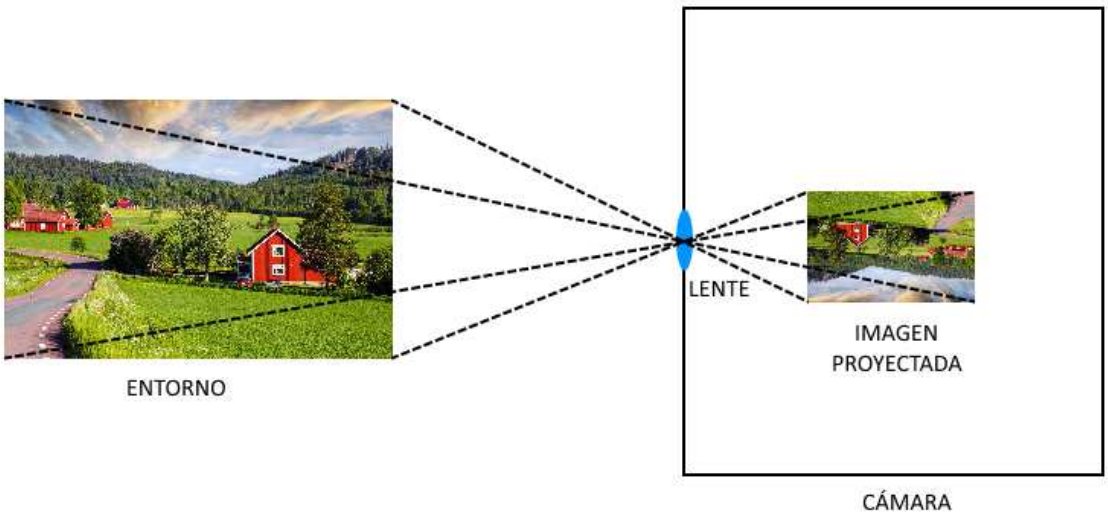

Figura 2-1: Principio de funcionamiento de una cámara. Elaboración propia.

De esta forma, si se toma como origen de referencia el punto de la lente en el que confluyen los haces, y se miden los ángulos horizontal y vertical a cada haz individual desde el correspondiente a la esquina superior izquierda de la imagen, se obtiene un sistema de coordenadas esféricas en el que los ángulos son directamente proporcionales a las coordenadas en píxeles sobre la imagen, con una proporción que dependerá de los ángulos de visión de la cámara, que es el ángulo máximo en cada dirección que se representa en la imagen, y de la resolución de la propia imagen.

Sabiendo esto, el cálculo de las coordenadas esféricas del objeto respecto al sistema de referencia de la cámara puede ser un buen punto de partida, para luego desplazar el origen y cambiar el sistema de referencia según convenga.

2.1.1 Coordenadas esféricas

Si los objetos son perfectamente identificables por color, o si se consigue una aproximación lo bastante cercana, bastará con calcular la posición media de los píxeles de ese color en la imagen para obtener los ángulos al centro del objeto.

En cuanto a la distancia entre el objeto y la cámara, el cálculo resulta algo menos trivial. Es posible calcularla a partir de su escala, siempre y cuando esta no se vea afectada por la rotación del objeto, ya que distintos puntos de un objeto alejado aparecerán más próximos entre sí que en un objeto cercano a la cámara, como se aprecia en la ilustración, donde los ángulos α y β representan el tamaño de cada objeto (línea de color) en la cámara, situada a la izquierda.

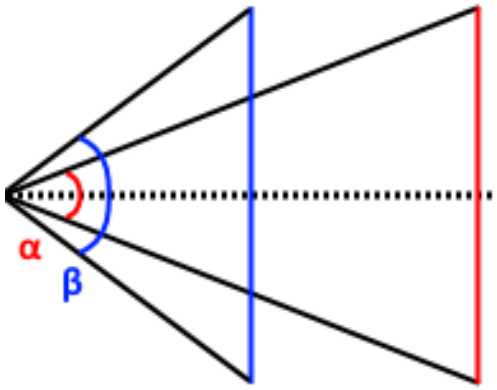

Figura 2-2: Representación bidimensional de la percepción de distancia en una cámara. Elaboración propia.

No obstante, para poder emplear esta propiedad, deben poderse detectar al menos dos puntos del objeto, cuya

distancia entre sí sea conocida. Una opción, que además permitiría conocer la orientación del objeto, es emplear dos elementos de colores distintos separados una distancia fija por cada mando, como hacía el sistema diseñado por Johnny Chung Lee a partir de wiimotes con la barra de sensores, pero este sistema requiere emplear y detectar dos colores por cada objeto, lo que reduce a la mitad el espectro de colores disponibles para la detección simultánea de objetos y duplica el tiempo de procesado, al tener que buscar ambos colores a lo largo de toda la imagen. Además, se sigue necesitando alguna forma de distinguir cuándo el cambio de distancia entre objetos observados se debe a un cambio de orientación y cuándo a un cambio de distancia a la cámara. El wiimote solucionaba este problema variando la orientación de los LEDs en los extremos de la barra, lo que provocaba un cambio en la intensidad luminosa, lo que resulta complicado de implementar en este caso.

Otra opción consiste en buscar el borde del objeto. Si se emplean objetos esféricos, este borde siempre estará a una distancia fija del centro, mientras que si se emplean círculos planos, el radio mayor de la elipse proyectada tendrá también la misma longitud aparente, independiente de la orientación. Una forma gráfica de comprobar este hecho es hacer girar una moneda sobre su canto: el eje de giro siempre tendrá el mismo tamaño aparente. La siguiente imagen representa, arriba, los triángulos que formarían la cámara y los dos puntos en un marcador esférico y plano. Abajo, un círculo plano en distintas orientaciones, representado sobre una esfera.

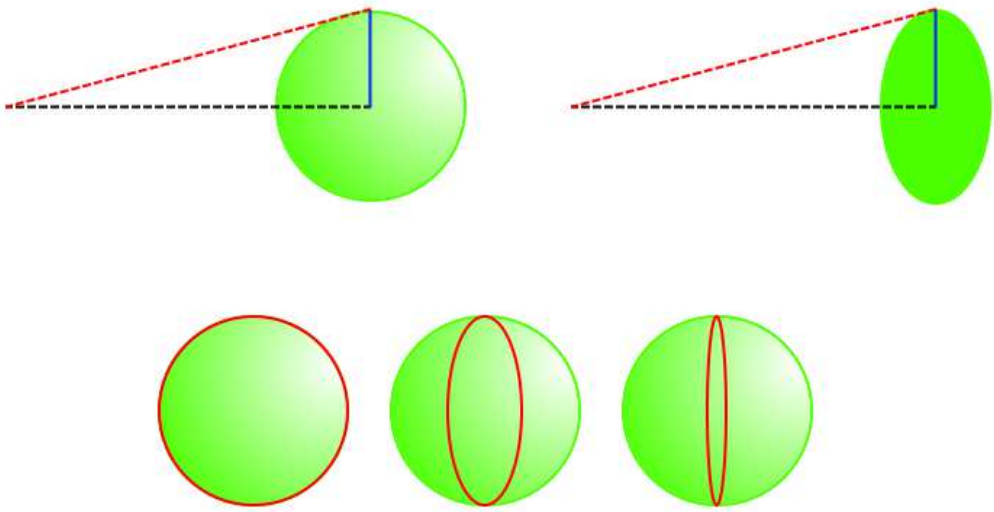

Figura 2-3: Detección del radio de formas circulares. Elaboración propia.

No obstante, esta forma no permite conocer la orientación del objeto, precisamente por las mismas propiedades que se buscan aprovechar. En muchas aplicaciones AR actuales, esto no es problema, pero si se desea obtener la orientación (cabeceo y guiñada) en la aplicación a desarrollar, deberán emplearse dos "objetos" por mando, obteniendo la orientación 3D a partir de la diferencia de posición entre objetos. Si la librería desarrollada en este trabajo devuelve el valor de las coordenadas 3D en un sistema cartesiano, el cálculo de orientación del segmento entre dos objetos consiste en una simple resta. Si además se desea obtener el alabeo, deberá añadirse un tercer objeto al mando.

Una ventaja de este sistema es la sencillez del algoritmo para encontrar los puntos, ya que lo realmente importante del punto en el borde es su distancia al centro. En lugar de volver a recorrer la imagen entera, bastaría con recorrer algunos de los píxeles alrededor del centro de cada objeto, tomando como parámetros un radio y un ángulo desde este. Si el pixel analizado es del color del objeto, se incrementa el radio, mientras el ángulo se incrementa continuamente. Si se recorre una circunferencia completa sin encontrar ningún pixel, el valor actual del radio es el radio del objeto en la imagen. La siguiente imagen representa esta búsqueda, marcando en blanco los pixeles analizados, así como el centro mediante una cruz.

Figura 2-4: Ejemplo de búsqueda de radio de un objeto rojo. Elaboración propia.

A partir del radio en píxeles, la relación entre los píxeles y el ángulo, y el radio real, es posible calcular la distancia a la cámara, ya que se obtiene un triángulo rectángulo donde un cateto (altura) es el radio real del objeto, el otro cateto (base) es la distancia y el ángulo entre la base y la hipotenusa es proporcional al radio en píxeles, en una proporción que depende únicamente del tamaño de imagen y del ángulo de visión de la cámara:

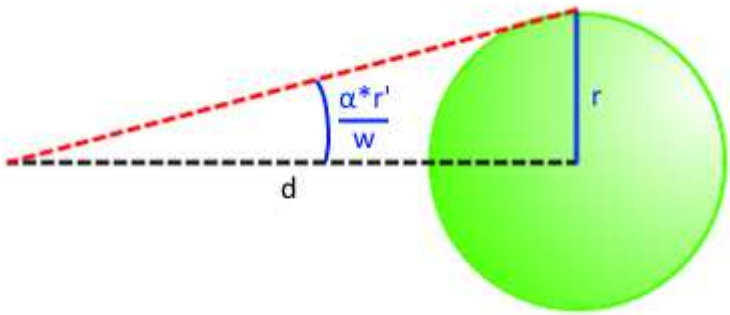

Figura 2-5: Cálculo de la distancia del objeto. Elaboración propia.

Donde d es la distancia a medir, r es el radio real del objeto, r' es el radio en píxeles en la imagen, α es el ángulo de visión de la cámara y w es la longitud en píxeles de la imagen correspondiente a dicho ángulo. En la mayoría de cámaras, la relación entre un ángulo de visión y la longitud correspondiente en la imagen es aproximadamente la misma tanto en horizontal como en vertical, ya que de otra forma la imagen se vería deformada, por lo que se va a considerar una única relación para ambas direcciones. De esta forma, las ecuaciones de las coordenadas esféricas para un objeto a partir de una imagen quedan:

$$\theta = \frac{\alpha}{w} * x \tag{2-1}$$

$$\varphi = \frac{\alpha}{w} * y \tag{2-2}$$

$$d = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} \tag{2-3}$$

No obstante, los cálculos empleados aquí proporcionan la posición basada en unos ejes esféricos relativos a la esquina superior izquierda de la cámara. Si bien estos ejes pueden ser útiles en algunas circunstancias, la mayoría de motores de programación incorporan sistemas de coordenadas basados en ejes cartesianos, que además suelen ser a los que más acostumbrados están tanto programadores como usuarios.

2.1.2 Cambio de sistema

A este respecto, las pantallas de PC incorporan un sistema cartesiano 2D para identificar las posiciones de los elementos en las mismas, que por razones históricas tiene origen en la esquina superior izquierda, un eje X hacia la derecha y un eje Y hacia abajo, con todas las medidas en píxeles. Como propuesta inicial, parece razonable añadir un eje Z según la regla de la mano derecha, por lo que quedaría hacia "dentro" de la pantalla, y emplear este sistema como referencia.

En la siguiente imagen se proporciona un esquema en el que aparecen un monitor de PC, una cámara USB y un punto arbitrario, sobre el que se han dibujado ambos sistemas de referencia y una representación gráfica de las coordenadas del punto en ellos.

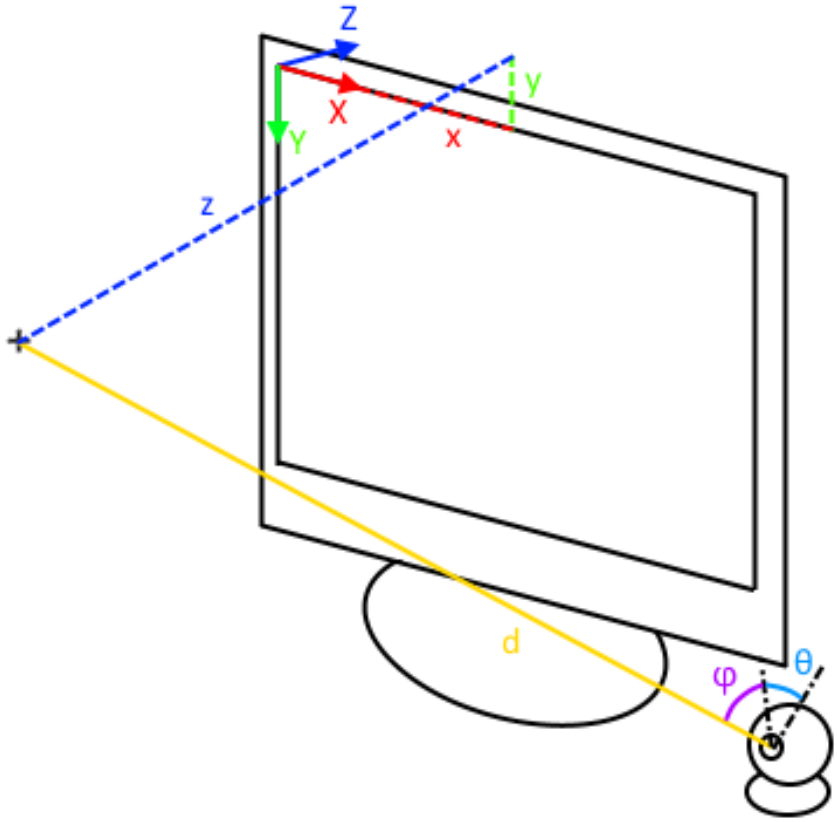

Figura 2-6: Sistemas de coordenadas. Elaboración propia.

El cambio entre ambos sistemas es relativamente sencillo, aunque conviene dividirlo en varios pasos.

Dado que los giros en el origen son más sencillos en coordenadas esféricas, mientras que los desplazamientos son más sencillos en coordenadas cartesianas, el primer paso sería colocar el eje de referencia del sistema esférico paralelo a uno de los ejes cartesianos, por ejemplo el eje Z. La siguiente imagen representa el giro en θ, pero es análogo para φ:

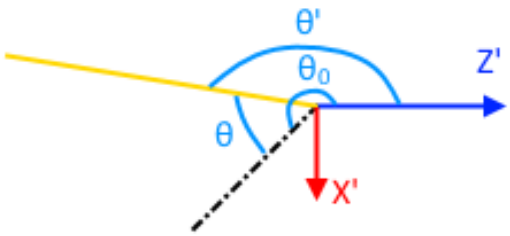

Figura 2-7: Giro de origen. Elaboración propia.

Donde θ' es el valor respecto al nuevo eje y θ_0 es el ángulo de giro. En ambos casos, además, el ángulo de giro puede calcularse para una cámara paralela a la pantalla como la mitad del ángulo de visión, lo que deja el eje

en el centro de la imagen, más 180º, al apuntar el eje Z en dirección contraria a la del usuario. Para un caso más general, bastaría añadir el ángulo entre la cámara empleada y una paralela a la pantalla.

En cuanto a los signos de los ángulos, tal como se representan en la imagen, son positivos en el sentido de las agujas del reloj, por lo que θ es positivo, θ' es negativo y las componentes de θ_0 serán positivas al medirse en este sentido y negativas en el contrario. Para φ, se obtiene el mismo esquema de signos al mirar el sistema desde la derecha, siendo positivos los ángulos "hacia arriba" desde el punto de vista del usuario.

Por tanto, las coordenadas tras el giro quedan:

$$\theta' = \theta - \theta_0 \tag{2-4}$$

$$\varphi' = \varphi - \varphi_0 \tag{2-5}$$

$$d' = d \tag{2-6}$$

El segundo paso es el cambio de coordenadas esféricas a coordenadas cartesianas, empleando unos ejes paralelos a los de la pantalla pero sin desplazar el origen. Tal como están orientados los sistemas tras el giro anterior, el cambio es inmediato:

$$x' = d' * \sin\theta' * cos\varphi' \tag{2-7}$$

$$y' = d' * sin\varphi' \tag{2-8}$$

$$z' = d' * \cos\theta' * cos\varphi' \tag{2-9}$$

Por último, debe realizarse un desplazamiento del origen del sistema a la esquina de la pantalla, para lo que bastaría con sumar las coordenadas del origen de la cámara en el nuevo sistema a las del objeto:

$$x = x' + x_0 \tag{2-10}$$

$$y = y' + y_0 \tag{2-11}$$

$$z = z' + z_0 \tag{2-12}$$

A lo largo de todo este proceso se han utilizado las mismas unidades de longitud que se emplearon para medir el radio del objeto. Sin embargo, dado que el sistema de la pantalla en 2D empleado por el ordenador utiliza píxeles como unidad de medida, puede resultar de interés multiplicar cada una de las coordenadas por el factor de conversión obtenido de dividir la longitud de la pantalla en las unidades empleadas entre la resolución de la

pantalla.

2.1.3 Coordenadas cartesianas

Añadiendo este factor y despejando en las ecuaciones 2-1 a 2-12 se obtiene el sistema de ecuaciones completo para la obtención de coordenadas 3D a partir de una imagen plana, donde se ha utilizado el sufijo "p" para distinguir las coordenadas en la imagen:

$$x = \left(\frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \sin\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + x_0 \right) * \frac{Resolución}{Longitud} \quad (2\text{-}13)$$

$$y = \left(\frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \sin\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + y_0 \right) * \frac{Resolución}{Longitud} \quad (2\text{-}14)$$

$$z = \left(\frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + z_0 \right) * \frac{Resolución}{Longitud} \quad (2\text{-}15)$$

2.2. Filtrado del color

Para poder emplear las ecuaciones anteriores, es necesario poder identificar el color del objeto y distinguirlo del resto de la imagen. En este caso, las cámaras codifican los colores como una combinación de luces roja, verde y azul.

El modelo estándar para las imágenes a color, denominado "color verdadero" por su cercanía a la profundidad de color del ojo humano, utiliza una profundidad de color de 24 bit, asignando valores enteros entre 0 y 255 (8 bit) a la potencia de cada una de las luces, en orden rojo, verde y azul (modelo RGB, por sus siglas en inglés). De esta forma, por ejemplo, (0, 0, 0) representa el color negro, (255, 255, 255) el blanco y (255, 0, 0) el rojo intenso. En total se obtienen 16.7 millones de colores distintos empleando este modelo.

Aunque existen otros modelos que pueden emplearse para la codificación del color, su funcionamiento es análogo, por lo que se empleará la codificación en color verdadero a modo de ejemplo en el desarrollo teórico.

Se busca diferenciar varios colores del resto y entre sí, con una cierta tolerancia, de forma fiable. Además, dado el elevado número de píxeles que deberán procesarse en cada imagen, el algoritmo debe emplear la menor complejidad temporal posible, es decir, deberá emplear el menor número de operaciones y estas deberán requerir el menor número de instrucciones al procesador.

Para representar de forma gráfica los colores filtrados, puede emplearse un modelo tridimensional donde cada eje corresponde a un color primario (rojo, verde y azul) y cada punto dentro de ese cubo corresponde a un color definido por sus valores RGB. Este cubo se denomina espacio de color RGB:

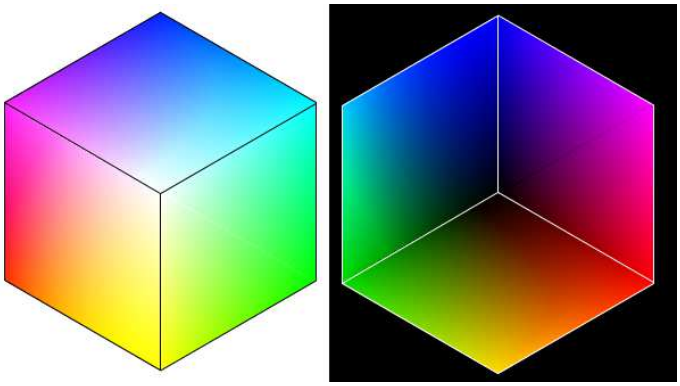

Figura 2-8: Vistas opuestas del espacio de color RGB. Elaboración propia.

A lo largo de esta sección se empleará una representación del espacio de color a la que se le han eliminado los colores de las caras y las aristas más cercanas, permitiendo "ver" el interior con claridad. En las aristas representadas, además, se dibujarán los ejes correspondientes a cada color, permitiendo observar la representación como una vista isométrica de unos ejes cartesianos.

Se han considerado varias opciones para el filtrado de colores.

2.2.1 Cubo arbitrario

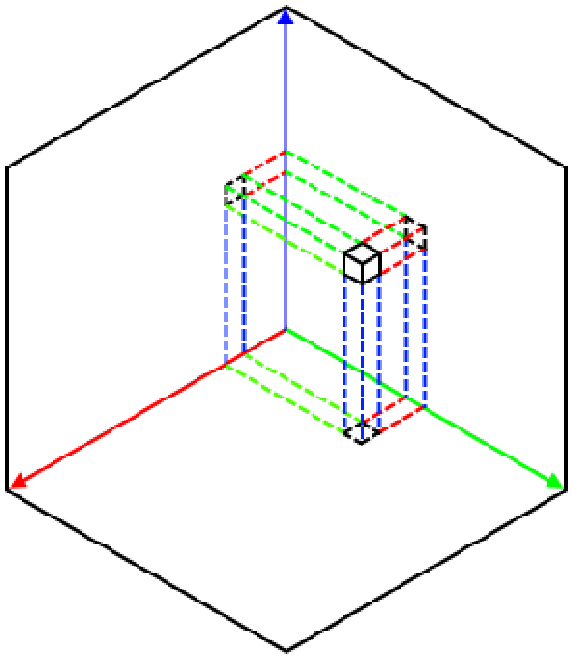

Figura 2-9: Representación de un cubo arbitrario en el espacio de color. Elaboración propia.

Este modelo de filtrado emplea un valor máximo y un valor mínimo para cada color. Para comprobar si un determinado color se encuentra dentro del cubo, deben hacerse 6 comparaciones, ya que por cada color el valor debe ser mayor que el mínimo y menor que el máximo. Una forma de obtener estos valores máximos y mínimos es emplear un color base, dado en sus coordenadas RGB, y una serie de tolerancias.

La principal ventaja de este algoritmo es su capacidad para emplear cualquier color como base, pero su complejidad es mayor a la de otros algoritmos probados. Además, mediante posterior experimentación con una librería parcialmente desarrollada, se ha observado que seleccionar colores en un volumen cúbico no siempre funciona bien en presencia de ruido, como por ejemplo sombras o luz ligeramente coloreada, detectando en ocasiones colores menos parecidos entre sí visualmente como el mismo, mientras ignora otros más parecidos al color objetivo.

2.2.2 Cubo en una esquina

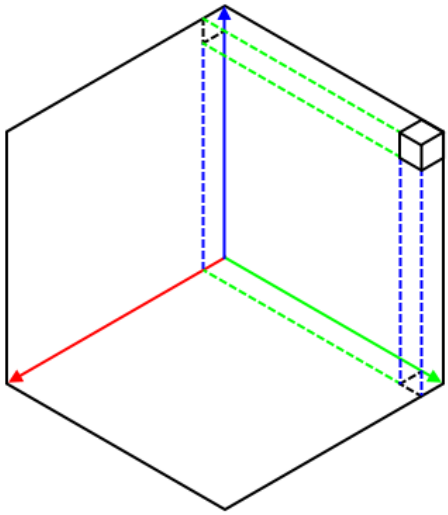

Figura 2-10: Representación de un cubo en una esquina del espacio de color. Elaboración propia.

Situar el cubo en una de las esquinas permite reducir el número de comparaciones a la mitad, ya que solo haría falta saber si cada componente queda por encima o por debajo de un único límite, pero también limita el número de colores a 8, un color por cada esquina del espacio de color: rojo, verde, azul, amarillo, cian, magenta, blanco y negro.

Dado que en la imagen cualquier sombra lo bastante oscura aparecerá en negro y cualquier luz lo bastante clara aparecerá en blanco, además de ser colores muy habituales, se debería prescindir de estos colores y utilizar únicamente los otros 6.

No obstante, dado que este algoritmo también emplea cubos, se observa experimentalmente el mismo problema que con el anterior.

2.2.3 Tetraedro en una esquina

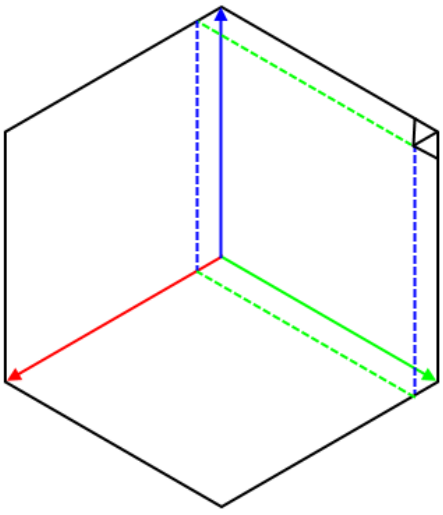

Figura 2-11: Representación de un tetraedro en una esquina del espacio de color. Elaboración propia.

Un dato interesante a tener en cuenta es que muchos procesadores actuales realizan operaciones de suma/resta y de multiplicación/división con enteros más rápido que operaciones de comparación. La diferencia en tiempo es muy pequeña, pero suficiente para plantear la posibilidad de reducir aún más el número de comparaciones empleando operaciones matemáticas, y comprobar el rendimiento.

Para obtener un valor numérico que mida cómo de cerca está un pixel a la esquina seleccionada, una opción es multiplicar por -1 el valor de las componentes del pixel que estén a 0 en la esquina y por 1 las que estén a 255

y a continuación sumar los resultados. De esta forma, para los colores primarios (rojo, verde y azul) el resultado variará entre -510 y 255, para los secundarios entre -255 y 510, para el negro entre -765 y 0 y para el blanco entre 0 y 765.

Una vez obtenido el valor del pixel para el color a analizar, basta con compararlo con un valor de referencia. Si se encuentra por encima, el pixel pertenece al color que se busca.

Para la comparación, puede ser interesante dar un rango fijo de valores común a primarios y secundarios. Una opción es dividir el valor de los secundarios a la mitad, lo que se puede hacer mediante una operación de "multiplicación entera" por 0.5 en lugar de por 1, y fijando el rango de tolerancias empleables al rango [0, 255], ya que de todas formas los valores inferiores a 0 suponen una distancia a la esquina tan grande que el color podría encontrarse en una esquina adyacente, lo que lo convertiría en un color distinto, y aún así considerarse válido.

El uso de multiplicación entera con 0.5, aunque a priori parece erróneo, es válido en muchos lenguajes, ya que el "entero" no hace referencia a las constantes que intervienen si no al tipo de dato a la salida. Una multiplicación entera trunca el resultado, lo que limita su precisión al emplearse con decimales, pero en este caso al emplearse en una comparación con una variable entera no es una limitación que afecte al resultado de forma apreciable.

Este método, por tanto, emplea de nuevo 6 operaciones, pero las tres multiplicaciones pueden hacerse en paralelo y la complejidad de las dos sumas es normalmente mucho menor a la de la comparación, por lo que la complejidad temporal es muy parecida a la del algoritmo anterior, sin poder afirmar con seguridad si es mayor o menor ya que esto dependerá del procesador empleado.

Se ha comprobado experimentalmente, no obstante, que este algoritmo detecta mejor los colores parecidos entre sí, separándolos del resto, que los algoritmos basados en cubos, por lo que será el algoritmo a emplear.

También puede comprobarse este efecto observando una de las caras en el espacio de colores, como en la siguiente imagen:

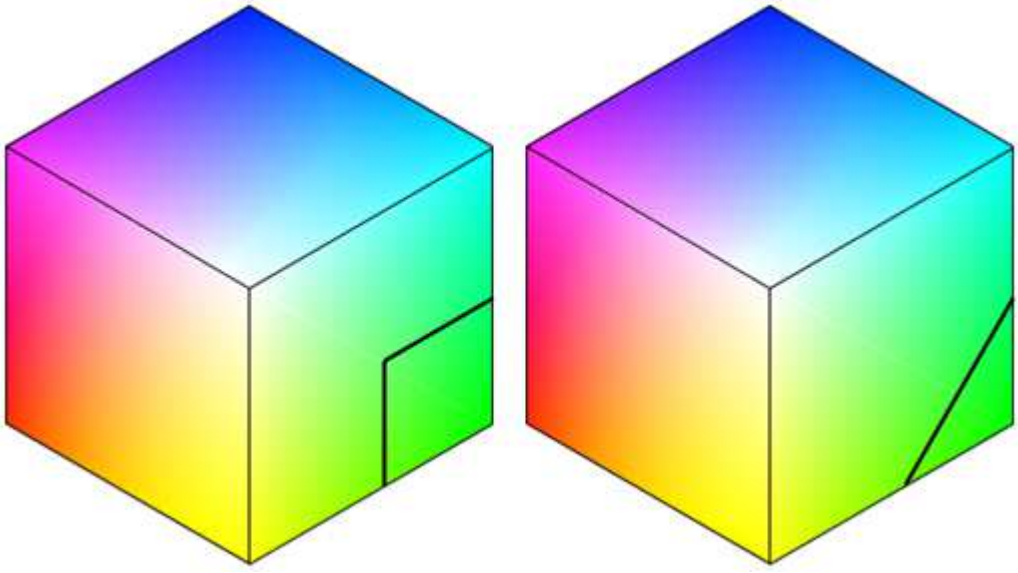

Figura 2-12: Comparación de métodos basado en cubo y basado en tetraedro. Elaboración propia.

Si se extraen los colores de las esquinas del cubo, se colocan en un cuadrado y se superpone el color objetivo, se obtiene el siguiente dibujo:

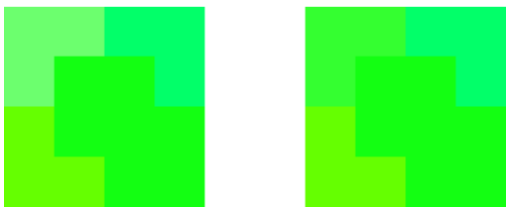

Figura 2-13: Colores en los límites detectados por el método del cubo y el tetraedro. Elaboración propia.

Donde se observa que el color de la esquina superior derecha del método del cubo se distingue del color objetivo mucho mejor que el resto de esquinas. En el método del tetraedro, donde se ha buscado el color del punto medio de la arista al no existir esa esquina, es algo más difícil de distinguir que las otras dos, pero el cambio del tono es mucho más parecido al que ocurre en las otras esquinas.

2.3. Resolución del sistema y rango de uso

Las cámaras tienen una resolución limitada, especialmente cuando transmiten datos en tiempo real. Las fórmulas desarrolladas en el primer apartado de este capítulo tienen una precisión ilimitada, pero sólo si se aplican a una imagen de resolución infinita.

Por otro lado, el algoritmo de búsqueda de colores necesita recorrer todos los píxeles de la imagen, los cuales componen una rejilla bidimensional sobre esta. Cuanta más alta sea la resolución, más pequeños y numerosos serán los píxeles y más tiempo llevará su procesado.

Partiendo de las fórmulas calculadas anteriormente puede calcularse la resolución que tendrán las coordenadas obtenidas, tanto esféricas como cartesianas, en función de la resolución de la cámara.

2.3.1 Coordenadas esféricas

Recuperando las ecuaciones antes calculadas:

$$\theta = \frac{\alpha}{w} * x \tag{2-16}$$

$$\varphi = \frac{\alpha}{w} * y \tag{2-17}$$

$$d = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} \tag{2-18}$$

Se puede calcular la resolución de las coordenadas como la variación de las mismas frente a la variación mínima de cada variable. De esta forma, el primer paso es identificar las variables y separarlas de parámetros constantes, para a continuación estudiar la función del incremento mediante la fórmula:

$$\Delta f(x) = f(x + h_x) - f(x) \tag{2-19}$$

Siendo f la coordenada a estudiar, x la variable relacionada (que no tiene por qué coincidir con la x de las ecuaciones anteriores) y h el incremento mínimo de la variable. En este caso, las variables son r', x, e y, para las coordenadas θ, φ y d, respectivamente.

En cuanto a los incrementos, el de x e y será 1, al tratarse de las coordenadas en píxeles enteros. En cuanto a r', será el incremento empleado por el algoritmo, ya que aunque también depende de los píxeles, puede medirse en diagonal, lo que da lugar a valores decimales. Aún así, puede considerarse 1 como valor típico de incremento.

De esta forma, las resoluciones quedan:

$$\Delta\theta = \frac{\alpha}{w}(x + 1) - \frac{\alpha}{w} * x = \frac{\alpha}{w} \tag{2-20}$$

$$\Delta\varphi = \frac{\alpha}{w}(y+1) - \frac{\alpha}{w} * y = \frac{\alpha}{w} \tag{2-21}$$

$$\Delta d = \frac{r}{\tan\left(\frac{\alpha}{w}(r'+1)\right)} - \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} \tag{2-22}$$

Las dos primeras ecuaciones muestran una relación directa entre la resolución de los ángulos y la relación entre el ángulo de visión y la resolución de la imagen: a mayor resolución de imagen y menor ángulo de visión, menor incremento del ángulo y, por tanto, mayor resolución.

En cuanto a la ecuación 2-22, plantea una cierta complejidad y conviene analizarla con más detenimiento. Operando se obtiene:

$$\Delta d = \frac{r * \left(1 + \tan\left(\frac{\alpha}{w} r'\right)\tan\left(\frac{\alpha}{w}\right)\right)}{\tan\left(\frac{\alpha}{w} r'\right) + \tan\left(\frac{\alpha}{w}\right)} - \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} \tag{2-23}$$

Si se despeja en 2-23 empleando 2-18 para sustituir el radio de la imagen por la distancia a la cámara, que es un valor más fácil de comprender, se obtiene:

$$\Delta d = \frac{r * \left(1 + \frac{r}{d}\tan\left(\frac{\alpha}{w}\right)\right)}{\frac{r}{d} + \tan\left(\frac{\alpha}{w}\right)} - d \tag{2-24}$$

Operando:

$$\Delta d = \frac{r * d + r^2 \tan\left(\frac{\alpha}{w}\right)}{r + \tan\left(\frac{\alpha}{w}\right) * d} - d \tag{2-25}$$

Para analizar esta ecuación en más detalle, conviene representarla en función de la distancia, al ser el parámetro variable durante el uso, y r, al ser un parámetro a diseñar, dando unos valores típicos al resto de parámetros.

Una resolución típica para cámaras USB es 640x480, por lo que puede partirse de este valor. En cuanto al ángulo de visión, el horizontal suele estar entre 50º y 140º, llegando en ocasiones a los 180º. Como ejemplo, el módulo de cámara V2 de una Raspberry Pi tiene un ángulo de 62.2º, por lo que se puede tomar un valor típico en torno a este valor.

A continuación, puede emplearse una hoja de cálculo para obtener los valores y representarlos. Para limitar la gráfica, se ha decidido fijar un alcance máximo a 5m, aunque este no tiene por qué ser el alcance máximo real del sistema, un tamaño máximo del objeto a 3cm, ya que se ha considerado que un radio mayor podría ser incómodo para el usuario, y un paso de resolución máximo de 1cm, ya que se considera que esta puede ser una precisión de partida relativamente baja para aplicaciones de carácter general.

Estas decisiones se han tomado de forma subjetiva y con carácter orientativo, con el fin de realizar gráficas generales que permitan ver la evolución del sistema. Es posible que aplicaciones específicas requieran otras condiciones de resolución o tamaño que no se hayan contemplado en estas gráficas, en cuyo caso debería repetirse el cálculo teniendo en cuenta estas condiciones específicas.

Se observa una mejora de precisión en objetos más grandes y cercanos a la cámara. También se ha repetido el cálculo con ángulos más grandes (180º) y resoluciones más pequeñas (320), lo que empeora el resultado.

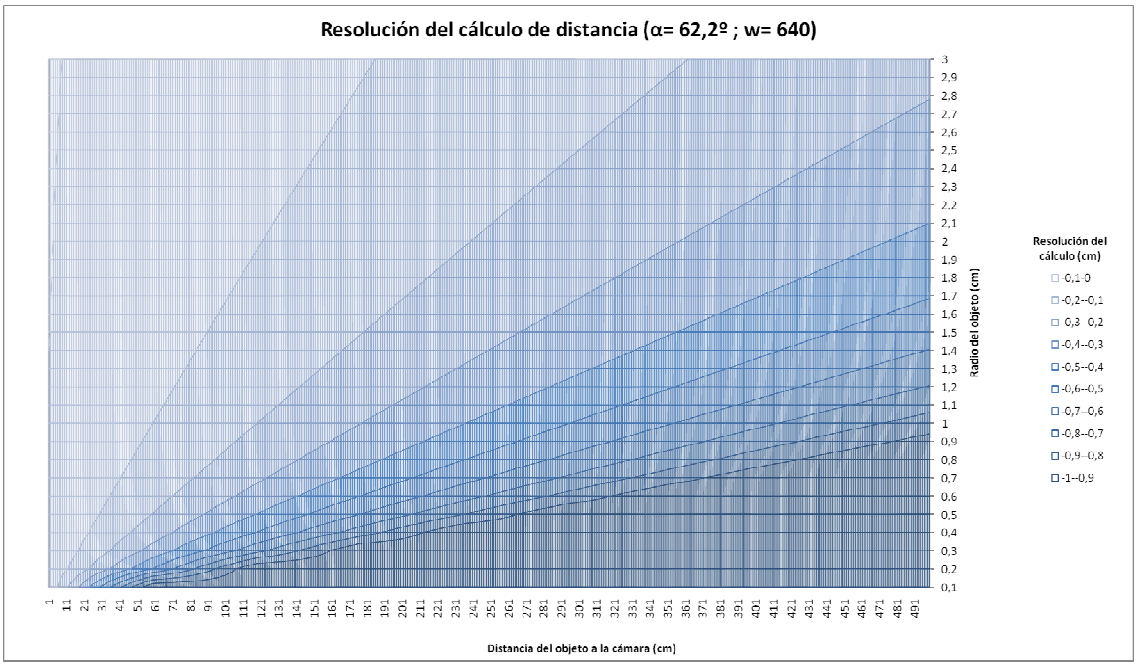

Figura 2-14: Resolución en función de la distancia y el radio. Elaboración propia.

Con esta cámara, además, se tiene una resolución de 0.0972 en ambos ángulos. Es la configuración recomendada en términos de resolución, obteniendo buenos resultados hasta los 5m con objetos a partir de 1cm de radio. Además, a 1m de distancia de la cámara, se dispone de 1.2m de distancia entre un extremo y otro de la imagen, con una resolución relativamente alta en la mayoría de tamaños de objeto.

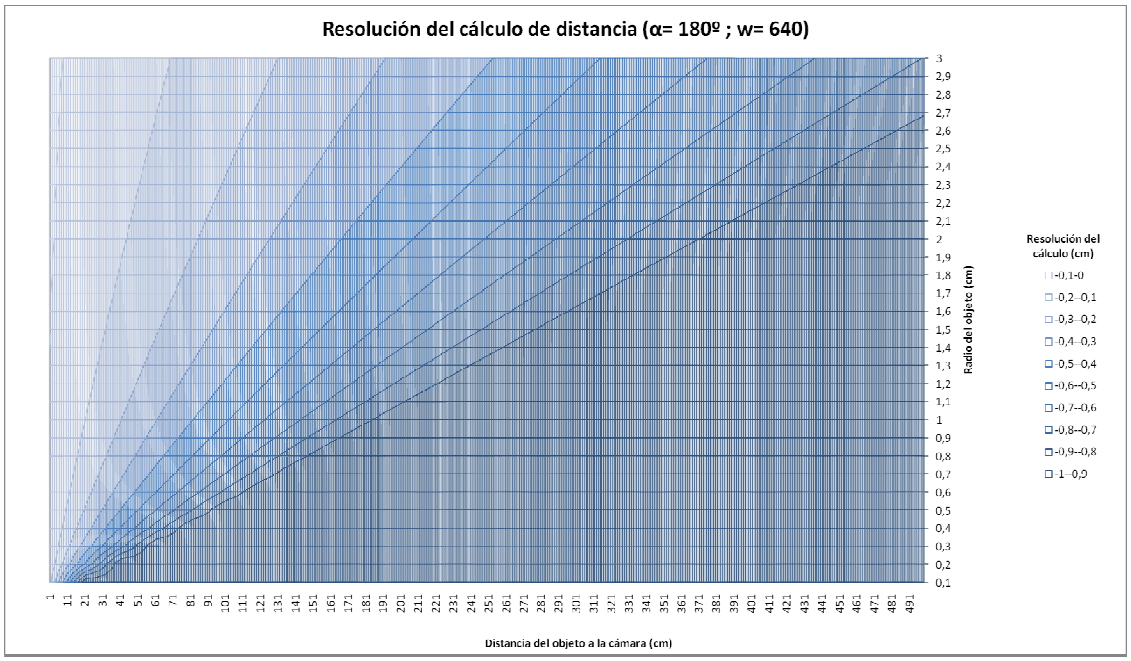

Figura 2-15: Resolución en cámaras de ángulo amplio. Elaboración propia.

Con una cámara de 180º, se tiene la ventaja de que son capaces de captar cualquier punto por delante de la cámara, pero la resolución angular es mucho menor (0.3º). En este caso, y con la tolerancia exigida, no se puede garantizar una buena detección, con una resolución de menos de 1cm, de un objeto de menos de 1cm de radio a más de 2m.

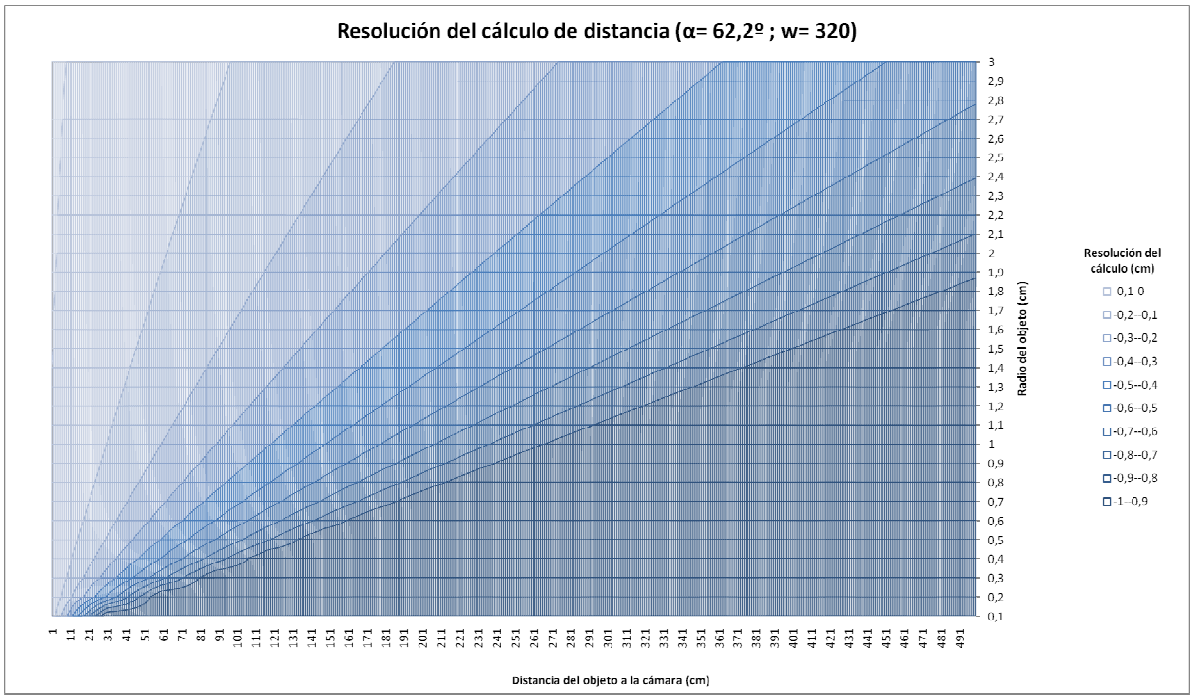

Figura 2-16: Resolución en cámaras de baja resolución. Elaboración propia.

En cuanto a cámaras de baja resolución, el resultado es similar a las de mayor ángulo, ya que una disminución de la resolución es equivalente a un aumento del ángulo en la misma proporción. Para una cámara de 320x240, un objeto de 1cm puede captarse con una resolución de menos de 1cm mientras esté a menos de 290cm, con una resolución angular de 0.2º.

Como conclusión, se ha confeccionado la siguiente tabla que indica qué acciones pueden aumentar la resolución y qué efecto adverso puede limitar su uso.

Aumento de resolución	**Efecto limitante asociado**
Aumento de la resolución de la cámara	Aumento del tiempo de procesado Puede ir asociado a un aumento del tiempo de refresco si se emplea la misma cámara con distinta configuración
Aumento del tamaño del objeto	Incomodidad de manejo para el usuario Puede requerir un aumento del ángulo de visión de la cámara Sin efecto en la resolución angular, sólo afecta a la de distancia
Disminución de la distancia a la cámara	Puede requerir un aumento del ángulo de visión de la cámara Sin efecto en la resolución angular, sólo afecta a la de distancia
Empleo de una cámara con menor ángulo de visión	Disminuye el espacio de detección

Tabla 2-1: Acciones que afectan a la resolución de las coordenadas esféricas. Elaboración propia.

2.3.2 Coordenadas cartesianas

Recuperando las ecuaciones 2-13 a 2-15, empleadas para obtener las coordenadas cartesianas a partir de una imagen plana:

$$x = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \sin\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + x_0 \quad (2\text{-}26)$$

$$y = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \sin\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + y_0 \quad (2\text{-}27)$$

$$z = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + z_0 \quad (2\text{-}28)$$

En las que se ha omitido el cambio de unidades a píxeles, ya que es más fácil entender los incrementos en centímetros que en píxeles. Además, todas las coordenadas dependen simultáneamente de r' y de y_p, y tanto x como z dependen además de x_p, ya que realmente la "rejilla" de valores de las coordenadas se corresponde a la de las coordenadas cartesianas, por lo que será necesario analizar los incrementos según cada una de estas variables.

De esta forma, para x_p, con un incremento mínimo de 1, se obtienen los incrementos:

$$\Delta x_{x_p} = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) * \sin\left(\frac{\alpha}{w} * \left(x_p + 1\right) - \theta_0\right) + x_0 - \mathrm{x} \quad (2\text{-}29)$$

$$\Delta y_{x_p} = 0 \quad (2\text{-}30)$$

$$\Delta z_{x_p} = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) * \cos\left(\frac{\alpha}{w} * \left(x_p + 1\right) - \theta_0\right) + z_0 - \mathrm{z} \quad (2\text{-}31)$$

Operando en la primera ecuación, se obtiene:

$$\Delta x_{x_p} = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) * \left(\sin\left(\frac{\alpha}{w} * x_p - \theta_0\right)\cos\left(\frac{\alpha}{w}\right) + \cos\left(\frac{\alpha}{w} * x_p - \theta_0\right)\sin\left(\frac{\alpha}{w}\right)\right) + x_0 - \mathrm{x} \quad (2\text{-}32)$$

Despejando los valores de x y z procedentes de las ecuaciones 2-26 y 2-28, respectivamente:

$$\Delta x_{x_p} = (x - x_0)\cos\left(\frac{\alpha}{w}\right) + (z - z_0)\sin\left(\frac{\alpha}{w}\right) + x_0 - \mathrm{x} \quad (2\text{-}33)$$

Operando:

$$\Delta x_{x_p} = (x - x_0)\left(\cos\left(\frac{\alpha}{w}\right) - 1\right) + (z - z_0)\sin\left(\frac{\alpha}{w}\right) \quad (2\text{-}34)$$

Tomando como referencia una cámara de 62.2º y 640x480 como la empleada en el apartado anterior, se

observa que la resolución es mayor cerca de la cámara, y que el incremento de x crece más rápidamente al alejarse en la dirección del eje z.

Operando de forma análoga en la tercera ecuación:

$$\Delta z_{x_p} = \frac{r}{\tan\left(\frac{\alpha}{w} * r'\right)} * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) * \left(\cos\left(\frac{\alpha}{w} * x_p - \theta_0\right)\cos\left(\frac{\alpha}{w}\right) - \sin\left(\frac{\alpha}{w} * x_p - \theta_0\right)\sin\left(\frac{\alpha}{w}\right)\right) + z_0 - z \quad (2\text{-}35)$$

$$\Delta z_{x_p} = \left((z - z_0)\cos\left(\frac{\alpha}{w}\right) - (x - x_0)\sin\left(\frac{\alpha}{w}\right)\right) + z_0 - z \quad (2\text{-}36)$$

$$\Delta z_{x_p} = (z - z_0)\left(\cos\left(\frac{\alpha}{w}\right) - 1\right) - (x - x_0)\sin\left(\frac{\alpha}{w}\right) \quad (2\text{-}37)$$

Se obtienen conclusiones similares, pero esta vez el incremento es mayor al alejarse en la dirección del eje x.

Para incrementos de y_p, dada la analogía en las ecuaciones, se obtienen resultados similares: la resolución disminuye al alejarse el objeto, aumentando más el incremento de y al alejarse en una dirección horizontal, y aumentando más los incrementos de x y de z al alejarse en la dirección del eje y:

$$\Delta x_{y_p} = \sqrt{(x - x_0)^2 + (z - z_0)^2}\left(\cos\left(\frac{\alpha}{w}\right) - 1\right) - (y - y_0)\sin\left(\frac{\alpha}{w}\right) \quad (2\text{-}38)$$

$$\Delta y_{y_p} = (y - y_0)\left(\cos\left(\frac{\alpha}{w}\right) - 1\right) + \sqrt{(x - x_0)^2 + (z - z_0)^2}\sin\left(\frac{\alpha}{w}\right) \quad (2\text{-}39)$$

$$\Delta z_{y_p} = \sqrt{(x - x_0)^2 + (z - z_0)^2}\left(\cos\left(\frac{\alpha}{w}\right) - 1\right) - (y - y_0)\sin\left(\frac{\alpha}{w}\right) \quad (2\text{-}40)$$

Estas ecuaciones tienen un sentido físico cuando se tiene en cuenta que x_p e y_p se corresponden con ángulos, siendo el valor mínimo de 1 equivalente a un ángulo de valor α/w, como se observa en la siguiente imagen:

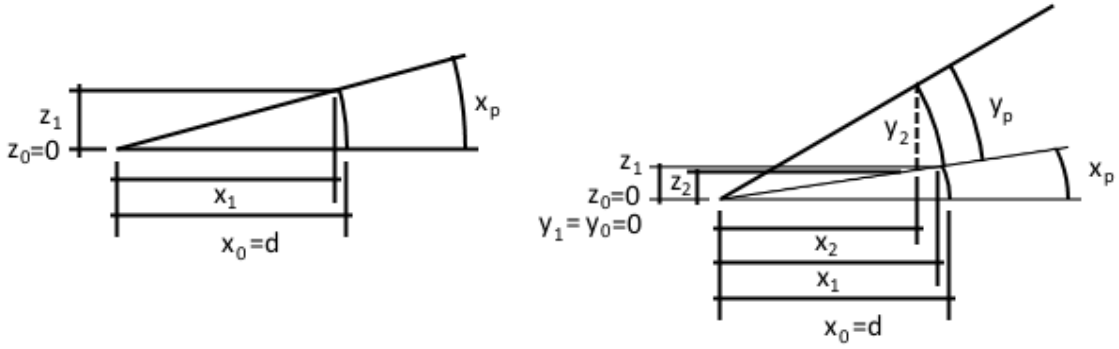

Figura 2-17: Incrementos de x_p e y_p. Elaboración propia.

En cuanto a r', como ya se explicó en el cálculo de la resolución de las coordenadas esféricas, se puede tomar un incremento de valor 1 como valor típico:

$$\Delta x_{r\prime} = \frac{r}{\tan\left(\frac{\alpha}{w} * (r' + 1)\right)} * \sin\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + x_0 - x \tag{2-41}$$

$$\Delta y_{r\prime} = \frac{r}{\tan\left(\frac{\alpha}{w} * (r' + 1)\right)} * \sin\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + y_0 - y \tag{2-42}$$

$$\Delta z_{r\prime} = \frac{r}{\tan\left(\frac{\alpha}{w} * (r' + 1)\right)} * \cos\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) + z_0 - z \tag{2-43}$$

Operando en la ecuación 2-26 se obtiene:

$$\sin\left(\frac{\alpha}{w} * x_p - \theta_0\right) * \cos\left(\frac{\alpha}{w} * y_p - \varphi_0\right) = \frac{(x - x_0) * \tan\left(\frac{\alpha}{w} * r'\right)}{r} \tag{2-44}$$

Y despejando en 2-41 y operando:

$$\Delta x_{r\prime} = \frac{(x - x_0) * \tan\left(\frac{\alpha}{w} * r'\right)}{\tan\left(\frac{\alpha}{w} * (r' + 1)\right)} + x_0 - x \tag{2-45}$$

$$\Delta x_{r\prime} = \frac{(x - x_0) * \tan\left(\frac{\alpha}{w} * r'\right)\left(1 - \tan\left(\frac{\alpha}{w} * r'\right)\tan\left(\frac{\alpha}{w}\right)\right)}{\tan\left(\frac{\alpha}{w} * r'\right) + \tan\left(\frac{\alpha}{w}\right)} - (x - x_0) \tag{2-46}$$

Recuperando la ecuación 2-18:

$$\tan\left(\frac{\alpha}{w} * r'\right) = \frac{r}{d} \tag{2-47}$$

$$\Delta x_{r\prime} = \frac{(x - x_0) * \frac{r}{d}\left(1 - \frac{r}{d}\tan\left(\frac{\alpha}{w}\right)\right)}{\frac{r}{d} + \tan\left(\frac{\alpha}{w}\right)} - (x - x_0) \tag{2-48}$$

$$\Delta x_{r\prime} = \frac{(x - x_0) * r\left(1 - \frac{r}{d}\tan\left(\frac{\alpha}{w}\right)\right)}{r + d * \tan\left(\frac{\alpha}{w}\right)} - (x - x_0) \tag{2-49}$$

Donde, por definición:

$$\mathrm{d} = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \tag{2-50}$$

Aunque resulta más cómodo no realizar esta última sustitución para la representación de la función.

Por analogía, el incremento del resto de coordenadas al incrementar r’ en una unidad quedan:

$$\Delta y_{r\prime} = \frac{(y - y_0) * r\left(1 - \frac{r}{d}\tan\left(\frac{\alpha}{w}\right)\right)}{r + d * \tan\left(\frac{\alpha}{w}\right)} - (y - y_0) \tag{2-51}$$

$$\Delta z_{r\prime} = \frac{(z - z_0) * r\left(1 - \frac{r}{d}\tan\left(\frac{\alpha}{w}\right)\right)}{r + d * \tan\left(\frac{\alpha}{w}\right)} - (z - z_0) \tag{2-52}$$

Donde se observa que el incremento es, por un lado, directamente proporcional a la coordenada, y por otro, directamente proporcional a una función de r, d y los parámetros de la cámara. Para poder analizar este incremento, se va a suponer un desplazamiento en la dirección del eje x de un objeto situado sobre el mismo, de forma que las otras dos coordenadas se anulen y la función quede:

$$\Delta x_{r\prime} = \frac{(x - x_0) * r\left(1 - \frac{r}{(x - x_0)}\tan\left(\frac{\alpha}{w}\right)\right)}{r + (x - x_0) * \tan\left(\frac{\alpha}{w}\right)} - (x - x_0) \tag{2-53}$$

Para representar la función, dado su crecimiento más rápido que en las funciones anteriores, se ha optado por recurrir a una "resolución relativa", multiplicando el resultado por 100 y dividiendo por la posición, y tomando el valor absoluto:

$$\Delta x_{r\prime}^{\%} = \left| \left(\frac{(x - x_0) * r\left(1 - \frac{r}{(x - x_0)}\tan\left(\frac{\alpha}{w}\right)\right)}{r + (x - x_0) * \tan\left(\frac{\alpha}{w}\right)} - (x - x_0) \right) * \frac{100}{(x - x_0)} \right| \tag{2-54}$$

Y la representación, para una cámara de 62.2º de ángulo de visión y 640 píxeles de resolución, queda:

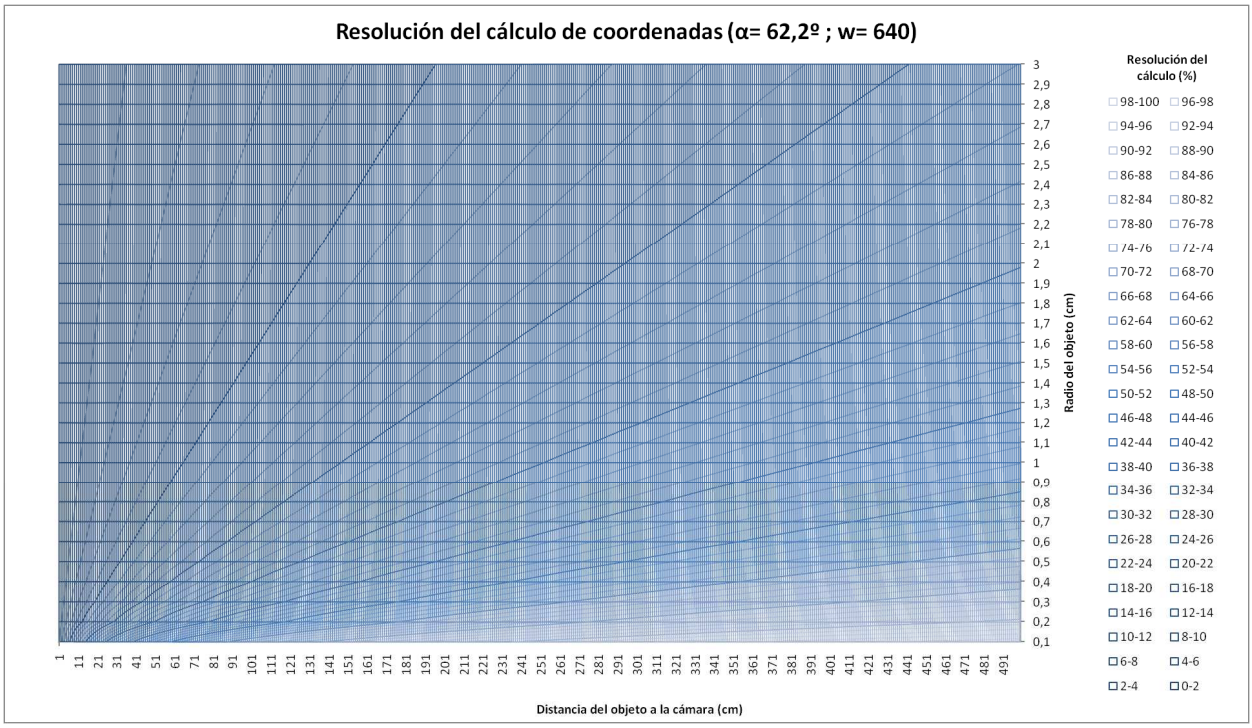

Figura 2-18: Resolución en coordenadas cartesianas. Elaboración propia.

Donde se han representado en un color más oscuro las líneas que marcan múltiplos de 10%, para facilitar la lectura.

De nuevo, y como cabe esperar, aumentar la relación α/w disminuye la resolución obtenida:

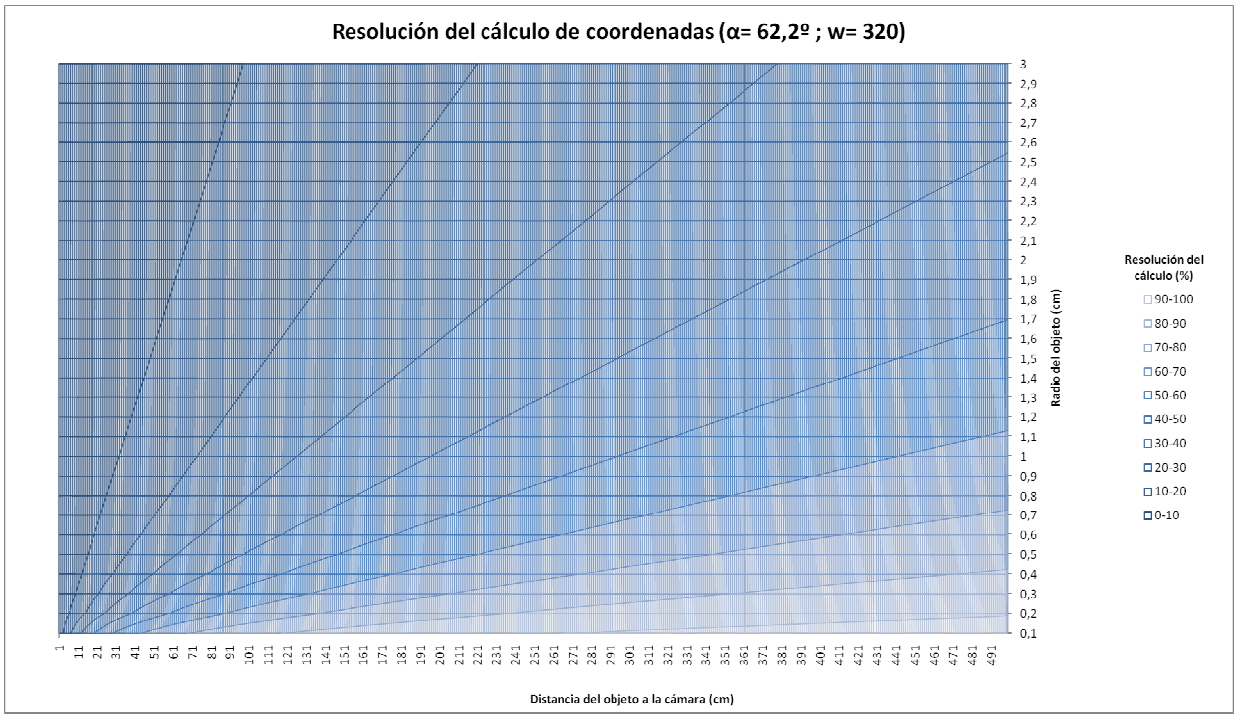

Figura 2-19: Resolución en coordenadas cartesianas con cámaras de baja resolución. Elaboración propia.

Se suele considerar que una resolución de un 10% de la medida es el valor máximo admisible para un instrumento con carácter general, por lo que los límites de distancia de uso recomendados para una cámara dada dependerán del tamaño del objeto a identificar.

Para una cámara de 62.2º de ángulo de visión y 640 píxeles de resolución, por ejemplo, se recomienda un límite de 33cm para un objeto de 0.5cm de radio, 66cm para un objeto de 1cm de radio o 99cm para un objeto de 1.5cm de radio.

Para una cámara con la mitad de resolución, estas distancias se reducirán a la mitad.

2.4. Conclusiones y diseño del objeto a detectar

Del apartado 2.1 se extrae que las formas óptimas para el algoritmo de detección diseñado son esféricas o, en su defecto, la de un círculo plano.

El apartado 2.2 proporciona información acerca de los colores óptimos a emplear. En este caso, se ha concluido que son 6: rojo, verde, azul, amarillo, cian y magenta. Los objetos óptimos, además, serán de color mate, para evitar ruido debido a reflejos y brillos. También podrían ser translucidos y disponer de iluminación interna propia, evitando la aparición de la sombra del objeto.

Además, dado que se busca que la cámara perciba objetos de estos colores con la mayor precisión posible, el entorno óptimo dispondrá de luz blanca, para evitar la coloración de los objetos por el color de la luz, y difusa, para minimizar el efecto de la sombra, siempre que no se disponga de objetos con iluminación propia. La procedencia recomendada de esta luz será desde detrás de la cámara, siempre que no deslumbre al usuario, o en su defecto desde un lateral. La mejor fuente de luz entre las probadas es la luz solar, especialmente si se emplean cortinas blancas para conseguir una luz más difusa. El sistema puede funcionar con luz artificial, pero aparecerá más ruido cuanto menos blanca sea.

También, para evitar ruidos, los colores de los objetos a detectar deberán escogerse de forma que no estén presentes en el fondo de la imagen o en elementos que puedan aparecer durante el funcionamiento, como ropa, reflejos procedentes de objetos móviles, etc.

Del apartado 2.3 se extrae información acerca de los tamaños óptimos de los objetos. Estos tamaños dependerán de la cámara a emplear, y, dado que se busca realizar un sistema abierto y fácilmente replicable, de los materiales disponibles.

Dado que se busca permitir un cierto grado de interacción, una opción es captar la posición de al menos dos dedos de una mano (preferiblemente índice y pulgar) y la cabeza del usuario. También pueden captarse mandos con botones y otros elementos interactivos, pero se ha preferido emplear elementos que se limiten a la posición del usuario para demostrar el funcionamiento del sistema elaborado.

Respecto a la posición de la mano, una opción es emplear un guante con la punta de los dedos a detectar pintada del color correspondiente, pero dependiendo de la forma de la mano del usuario esto puede dificultar obtener una forma aproximadamente esférica o un círculo plano, y el tamaño puede no ser suficiente dependiendo de la calidad de la cámara empleada.

Otra opción, empleada en la elaboración de los prototipos con los que se ha probado el trabajo, es construir "dedales" del tamaño, color y forma adecuados. En este caso se ha utilizado gomaespuma, con una capa de pintura acrílica para proporcionar el color adecuado.

En cuanto al elemento de detección de la cabeza, puede emplearse un gorro, bandana o similar de un color no empleado por el algoritmo y coser, sujetar o dibujar un elemento circular. En el caso de los prototipos, se ha empleado una bandana negra, a la que se ha cosido un círculo de goma EVA (etileno-vinil-acetato) como elemento detectable.

Los colores escogidos para los prototipos han sido rojo para el dedo índice, verde para el pulgar y amarillo para la cabeza, aunque pueden intercambiarse entre sí o con cualquiera de los otros 3 colores disponibles.

Originalmente se creó también un elemento azul para la detección de la cabeza, pero fue descartado al descubrir a través de una aplicación de prueba en las primeras fases del proyecto que en este caso había una fuente de ruido azul, procedente del reflejo de la pantalla del ordenador en el filtro ultravioleta de unas gafas de vista, que impediría el correcto funcionamiento del sistema, como se aprecia en la siguiente imagen, donde la parte izquierda es la imagen captada por la cámara y la derecha muestra los colores reconocidos por el algoritmo.

Figura 2-20: Prueba de prototipos. Elaboración propia

3 DISEÑO DE LA LIBRERÍA

"The function of good software is to make the complex appear to be simple"

(La función de un buen software es hacer que lo complejo parezca ser simple)

- Grady Booch

Para implementar el sistema que se acaba de diseñar, se va a emplear una librería en algún lenguaje de programación de uso extendido, para facilitar el acceso a futuros desarrolladores que puedan buscar acceso a una tecnología AR.

Para ello, antes de comenzar a diseñar la librería, conviene hacer un análisis de los lenguajes más utilizados, con el fin de escoger el más adecuado.

Se dispone de conocimientos previos de algunos lenguajes, lo que puede favorecer el uso de alguno de ellos en caso de duda: C y sus variantes C++ y C#, Java y su variante simplificada para proyectos multimedia Processing, Haxe (que puede ser considerado una variante de Java) y GDScript (variante de Python escrita en C para el motor de videojuegos Godot), además de algunos lenguajes relacionados con programación web como PHP o HTML, que a priori parecen poco adecuados. En menor medida, se dispone también de conocimientos básicos de JavaScript y Python.

En cualquier caso, si tras el análisis previo se descubriera un lenguaje distinto que pudiera ser más adecuado que cualquiera de estos, se valoraría la posibilidad de emplearlo en el diseño de la librería.

Además, es habitual la "traducción" de librerías a otros lenguajes, en caso de necesidad. La búsqueda de un lenguaje adecuado y extendido se basa, por un lado, en la facilidad para la creación y uso de la librería, y por otro en la facilidad para atraer desarrolladores en las etapas iniciales, en caso de publicarse, pero no se descarta la elaboración de librerías específicas para otros lenguajes en un futuro si la idea se extiende.

3.1. Selección del lenguaje de programación

Dado el vasto número de lenguajes de programación existentes, se hace necesario obtener un listado reducido con los más empleados.

Una posibilidad es emplear el índice TIOBE, que registra las búsquedas en internet relacionadas con los distintos lenguajes de programación y los ordena por número de búsquedas. Puede entenderse que, con una cierta desviación, los lenguajes más buscados suelen ser los más empleados.

De esta forma, se pueden analizar los 10 primeros lenguajes de la lista, y escoger el más adecuado entre ellos. Dado que el índice TIOBE no filtra por tipo de lenguaje o ámbito de aplicación, algunos de ellos quedarán descartados automáticamente por ser demasiado específicos y pertenecer a un ámbito distinto del desarrollo de aplicaciones para dispositivos de sobremesa.

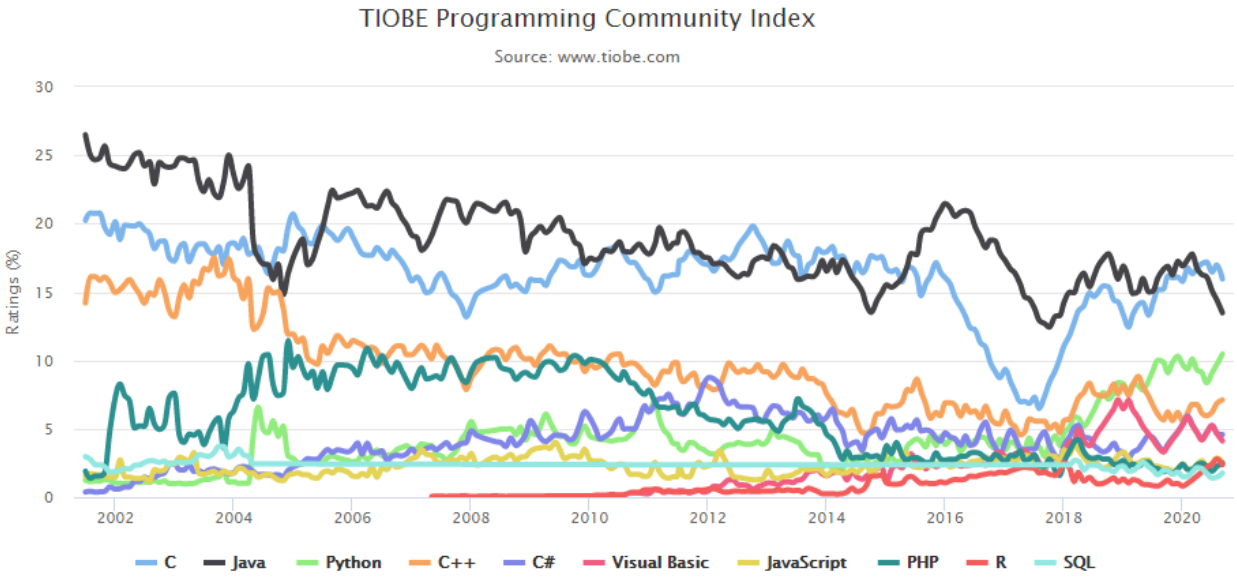

Figura 3-1: Índices TIOBE de los últimos años. (TIOBE, 2020)

3.1.1 C

C es un lenguaje estructurado que permite el desarrollo de aplicaciones a nivel de sistema, casi al mismo nivel permitido trabajando directamente en ensamblador, con la ventaja de ofrecer un código a nivel relativamente alto que facilita trabajar en el desarrollo para distintos sistemas operativos y equipamiento hardware de forma simultánea, lo que explica que su popularidad permanezca relativamente alta pese al auge de otros lenuajes.

Es un lenguaje compilado, lo que proporciona una mayor velocidad de ejecución que en un lenguaje interpretado, y su compilador permite generar aplicaciones para un amplio abanico de sistemas operativos.

De hecho, muchos sistemas operativos, entre ellos Linux, emplean C como lenguaje de desarrollo.

No obstante, el principal uso de C en la actualidad como lenguaje para el desarrollo de aplicaciones se orienta principalmente a la creación de herramientas para científicos y desarrolladores, como por ejemplo algunos módulos de MATLAB o Mathematica, los motores de videojuegos Unity y Godot o la creación de otros lenguajes como el propio C++. Las aplicaciones más visuales aprovechan características de C++ o C#, variantes analizadas por separado más adelante.

Además, aunque es posible la obtención de datos de una cámara a través de C, resulta bastante más complejo que en sus variantes, para las que además existen librerías prediseñadas con esta función.

3.1.2 Java

Aunque la popularidad de Java lleva varios años en lenta decadencia, a día de hoy sigue siendo el segundo lenguaje más utilizado según el índice TIOBE. Esta pérdida de popularidad se debe en gran medida a la aparición de JavaScript, que ganó fuerza rápidamente en el desarrollo web, prácticamente erradicando a java de este terreno, y a desacuerdos entre las compañías de oracle y google, que han limitado su uso en aplicaciones para android, aunque continúa siendo compatible. En aplicaciones para ordenador, no obstante, continúa siendo ampliamente utilizado.

Su principal ventaja es el uso de una máquina virtual como un segundo sistema operativo, para la cual se compila el código y desde la que se ejecuta; esta implementación es lo que hace que Java sea considerado a la vez interpretado y compilado: se compila para la ejecución de la máquina virtual, pero la máquina actúa como un “intérprete”, si bien es más cercano a un emulador. Como resultado, las aplicaciones se ejecutan más rápido que en otros lenguajes interpretados, pero consume algunos recursos más que un lenguaje compilado al uso.

Otra característica importante de esta implementación es su capacidad para la creación de aplicaciones multiplataforma: al emplear la máquina virtual, una vez compilado un programa podrá utilizarse en cualquier hardware compatible y se garantiza (con contadas excepciones) que el funcionamiento será análogo,

independientemente del sistema operativo asociado.

Se trata además de un lenguaje orientado a objetos, lo que puede facilitar el desarrollo implementando un objeto que agrupe todas las funciones necesarias y gestione la cámara internamente.

Otra ventaja es la existencia previa de una librería gratuita para Java, llamada Webcam Capture y desarrollada por Bartosz Firyn ("SarXos"), que facilita la obtención y tratamiento de imágenes procedentes de cámaras integradas o conectadas vía USB, por lo que únicamente sería necesario crear las funciones específicas del sistema AR a diseñar.

También se dispone de una herramienta, IKVM.NET, que permite trasladar librerías y aplicaciones desarrolladas en Java al entorno .NET Framework de Microsoft, que de forma nativa emplea C# y Visual Basic y es uno de los entornos empleados por los motores de videojuegos Unity, Unreal Engine y Godot, actualmente considerados los tres motores de videojuegos más extendidos.

En consecuencia, Java parece una buena opción para la realización de esta librería.

3.1.3 Python

La popularidad de Python se ha incrementado notablemente en los últimos años, sobre todo gracias a su facilidad de uso.

Se trata de un leguaje interpretado basado en script que se ha abierto paso rápidamente en desarrollo de aplicaciones web, machine learning e investigación científica.

Además, es frecuente su uso en aplicaciones de reconocimiento de imágenes mediante cámara, muchas de ellas con una complejidad mucho más elevada que la que se pretende implementar aquí, por lo que la documentación al respecto es fácil de encontrar.

No obstante, y a pesar de existir un compilador para generar ejecutables a partir de Python encapsulando el interprete junto al código, su uso en desarrollo de aplicaciones de uso común fuera del entorno web es aún muy reducido, lo que limita su popularidad en el grupo de desarrolladores que podrían sacar más provecho al sistema desarrollado.

3.1.4 C++

Variante de C que originalmente buscaba añadir la manipulación de objetos y actualmente incorpora paradigmas de programación estructurada, orientada a objetos y genérica, siendo uno de los lenguajes multiparadigma más empleado actualmente.

Cuenta con las ventajas de C, al ser una ampliación de este, pero gracias a su funcionamiento como lenguaje multiparadigma cuenta con un uso mucho más extendido en el ámbito de aplicaciones de escritorio con carácter general, aunque carece de algunas funciones útiles en el desarrollo de videojuegos disponibles en C#, más extendido en este ámbito.

Además, existe una librería que permite su uso para la lectura de imágenes desde cámaras, por lo que el principal problema de C en este ámbito queda solucionado.

Existe un método para trasladar librerías de C++ a C#, y viceversa, por lo que también es compatible con el entorno de .NET Framework.

En consecuencia, parece una opción tan buena como Java para la realización de la librería, aunque en este caso se disponen de menos conocimientos previos.

3.1.5 C#

Variante de C orientada a objetos que emplea una máquina virtual propia, de forma parecida a como lo hace Java, por lo que las aplicaciones realizadas requieren esta máquina virtual. Es empleado por los principales motores de videojuegos, como ya se mencionó antes, y sus prestaciones son similares a las de C++ o java.

El uso de una máquina virtual limita el acceso del lenguaje a las funciones relacionadas con el hardware, por lo que su uso es más seguro pero no permite la realización de drivers o similares.

Su uso está mucho más extendido que el de C++ en el diseño de videojuegos, especialmente gracias a Unity,

pero menos extendido en el resto de aplicaciones.

De nuevo, es una opción válida, pero de la que en este caso se disponen de menos conocimientos previos. El hecho de que una librería Java sea compatible con este lenguaje dentro del .NET Framework favorece la idea de emplear Java para desarrollar la librería y ofrecerla también a los desarrolladores de C#.

3.1.6 Visual Basic

Lenguaje de programación dirigido por eventos que actualmente formaría parte del .NET Framework, bajo el nombre Visual Studio. No se actualiza de forma independiente desde 1998, y para realizar la librería bajo .NET Framework parece más adecuado emplear un lenguaje orientado a objetos, por lo que se emplearía C#.

3.1.7 JavaScript

A pesar del nombre y de ser un lenguaje orientado a objetos, no existe relación entre este lenguaje y Java. JavaScript está orientado al diseño de páginas web dinámicas, conteniendo instrucciones a ejecutar por el navegador.

Una estructura típica para una página web actual emplea SQL para gestionar una base de datos del servidor, PHP para ejecutar instrucciones en el servidor y modificar el archivo HTML, HTML para describir el aspecto de la página y JavaScript para realizar acciones desde el navegador, como recargar una página o realizar animaciones en el contenido.

También se utiliza en la realización de widgets, pequeñas aplicaciones de escritorio, y es posible utilizarlo dentro de algunos archivos no web, como PDF, pero queda fuera del ámbito de este trabajo.

3.1.8 PHP

Aunque es un programa multiparadigma de ámbito general, el ámbito al que más se adapta es el del desarrollo web, especialmente a la ejecución de instrucciones desde el lado del servidor, como se ha mencionado durante el análisis de JavaScript, por lo que también queda fuera del ámbito de este proyecto.

3.1.9 R

Lenguaje multiparadigma de software libre con un ámbito de aplicación muy parecido a Python, con el que de hecho es compatible, siendo habitual el uso de funciones de R desde scripts de Python. Destaca, especialmente, en las aplicaciones de machine learning.

A pesar de su indiscutible potencia, los ámbitos de aplicación habituales de este lenguaje se alejan de los objetivos de este trabajo, especialmente teniendo en cuenta que el objetivo de R no es generar aplicaciones finales compiladas, si no código fácilmente editable para diversas funciones.

3.1.10 SQL

SQL es un lenguaje de gestión de bases de datos que no admite otras funciones, por lo que no puede utilizarse en el desarrollo de este sistema.

3.1.11 Resumen y conclusiones del análisis

Para facilitar la lectura de los análisis se ha realizado la siguiente tabla. El orden de la tabla se corresponde con el orden en el índice TIOBE. No se han incluido Visual Basic, JavaScript, PHP, R y SQL, por ser poco adecuados por sus características y ámbitos de aplicación.

Lenguaje	Nivel de conocimientos previos	Tipo de compilación	Ámbito principal de aplicación	Capacidad para lectura de cámaras	Otros factores
C	Medio	Compilado	Desarrollo de sistemas Drivers hardware Desarrollo de ejecutables	Debe implementarse manualmente	Complejidad elevada para esta aplicación Programación estructurada
Java	Alto	Compilado a VM	Desarrollo de ejecutables Desarrollo web (muy ocasional)	Disponible una librería abierta gratuita	Programación orientada a objetos Puede utilizarse como librería de .NET Framework (C#)
Python	Bajo	Interpretado	Desarrollo de aplicaciones web Computación científica y análisis de datos Machine learning	Ampliamente utilizado en visión	Muy poco usado en desarrollo de juegos, aun teniendo motor propio (PyGame) Muy poco usado en aplicaciones de escritorio, aun teniendo compilador (Tkinter)
C++	Medio	Compilado	Desarrollo de ejecutables	Disponible una librería abierta gratuita	Programación multiparadigma Puede utilizarse para hacer librerías de C#
C#	Medio	Compilado a VM	Desarrollo de ejecutables (videojuegos)	Disponible una librería abierta gratuita	Programación orientada a objetos Usado por los principales motores de videojuegos

Tabla 3-1: Lenguajes analizados. Elaboración propia.

En base al análisis, se ha optado por emplear Java, ya que se trata de un lenguaje con el que se parte de buenos conocimientos previos, que es ampliamente utilizado en el desarrollo de aplicaciones de escritorio, por lo que se espera que la librería pueda atraer a más desarrolladores, que por sí mismo es ampliamente utilizado, disfrutando actualmente de un índice TIOBE del 13.48% frente al 10.47% de Python o el 7.11% de C++, y que es compatible con los principales motores de desarrollo de videojuegos actuales, lo que puede aumentar aún más la visibilidad de la librería.

3.2. Creación de la librería: la clase ColorFinder

Escogido el lenguaje, se ha considerado que una implementación que facilita la creación de futuras aplicaciones es mediante un objeto encargado de realizar el seguimiento, el cual se describe a través de una clase a la que se ha llamado ColorFinder, por su algoritmo de búsqueda de objetos basado en color, que será el único contenido de la librería.

Esta clase contiene una serie de métodos, o funciones, creados expresamente durante este trabajo para proporcionar al usuario los datos procesados. El código completo puede consultarse en el anexo B.

A continuación se explica cada una de las partes de la clase. Para mayor claridad de las explicaciones, se han eliminado los comentarios javadoc, utilizados para generar la documentación asociada a la librería, aunque si se desea pueden consultarse en el anexo.

3.2.1 Declaración del paquete y llamada a librerías

Las clases de java pertenecen a paquetes, que funcionan de forma similar a las librerías de cualquier otro lenguaje. En este caso, se ha optado por emplear el nombre de paquete desktopAR para identificar a la librería generada:

```
package desktopAR;
```

A continuación, aparece el listado de elementos a importar de otras librerías. Para esta clase, se va a emplear la clase Webcam de la librería de Sarxos para la gestión de cámaras y una serie de elementos de las librerías básicas de java: Color, Dimension y BufferedImage para gestionar imágenes, el número PI y las funciones trigonométricas básicas en la realización de cálculos y Arrays y listas para gestionar coordenadas y métodos con número de parámetros variable.

```
import com.github.sarxos.webcam.Webcam;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import static java.lang.Math.PI;
import static java.lang.Math.cos;
import static java.lang.Math.sin;
import static java.lang.Math.tan;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

3.2.2 Declaración de la clase

Una vez importado el contenido, se declara la clase, que tendrá el mismo nombre que el objeto que implemente:

```
public class ColorFinder {
```

Se trata de una clase pública, ya que deberá ser accesible para otras clases fuera del paquete para que la librería pueda ser implementada. La llave abierta en esta línea se cierra al final del archivo de código, ya que todo el contenido pertenece a la misma clase.

3.2.3 Constantes y variables del objeto

Un objeto de tipo ColorFinder dispone de una serie de constantes y variables. De entrada, se dispone de una serie de constantes que identifican los colores utilizables según lo establecido en el punto 2.2.3:

```
static public final float[] RED = {1, -1, -1};
static public final float[] GREEN = {-1, 1, -1};
static public final float[] BLUE = {-1, -1, 1};
static public final float[] CYAN = {-0.5f, 0.5f, 0.5f};
static public final float[] MAGENTA = {0.5f, -0.5f, 0.5f};
static public final float[] YELLOW = {0.5f, 0.5f, -0.5f};
static public final float[] WHITE = {0.3f, 0.3f, 0.3f};
static public final float[] BLACK = {-0.3f, -0.3f, -0.3f};
```

Donde el algoritmo devolverá para cada píxel un valor menor a 255 para cada color buscado, excepto para el blanco, que llega hasta 230, y el negro, que va de -230 a 0. Se recomienda, no obstante, evitar el uso de estos dos últimos colores en la medida de lo posible.

Estas constantes son estáticas, de forma que podrán utilizarse aunque no haya ninguna instancia del objeto ColorFinder, públicas, por lo que su acceso desde fuera de la clase está permitido, y final, por ser constantes.

Las siguientes variables son listas relacionadas con las funciones de la librería:

```
private List<float[]> targets = new ArrayList<>();
private List<double[]> positions = new ArrayList<>();
private List<double[]> positionSpheric = new ArrayList<>();
private List<int[]> positions2D = new ArrayList<>();
private List<Integer> size2D = new ArrayList<>();
private List<Integer> tol = new ArrayList<>();
```

Son privadas para evitar modificaciones externas accidentales, y se han empleado como listas ya que de esta forma pueden hacerse de longitud variable, mientras que los array tienen un tamaño fijo, permitiendo así ajustar el número de objetivos desde la aplicación en la que se implemente la librería, sin necesidad de borrarlos datos preexistentes. Las listas contienen, por orden: los colores a identificar, en el formato anterior, las posiciones en coordenadas cartesianas 3D de los objetos, sus posiciones esféricas, sus posiciones en la imagen, los radios en la imagen de cada objeto y la tolerancia en el algoritmo de búsqueda de color.

A continuación aparecen el resto de variables privadas del objeto:

```
private Webcam webcam;

private int width;
private int height;

private boolean debug = false;
private BufferedImage image;
private int[][] readArray;
```

Un objeto Webcam, encargado de obtener las imágenes de la cámara, dos enteros con la altura y anchura de la imagen, un booleano para entrar en “modo debug” y generar una imagen con la captura de la cámara y una representación del algoritmo, y un array entero que almacena información sobre los píxeles leídos, que se reescribirá en cada iteración por lo que no necesita ser una lista.

3.2.4 Constructores

A continuación aparecen los constructores del objeto, que en este caso son dos. Un constructor es un método que permite crear una instancia del objeto, empleando los parámetros necesarios para su inicialización. En Java, debe crearse un constructor por cada conjunto de parámetros empleable, ya que no se permite asignar valores por defecto a los parámetros. Esta técnica se conoce como “sobrecarga de métodos”. Dada la similitud

entre ambos constructores, se va a explicar únicamente el primero, por ser el que más parámetros admite.

```
 public ColorFinder(Webcam webcam, int width, int height, int tol,
float[]... colors) throws IllegalArgumentException {
```

Esta línea corresponde a la declaración del método. Al ser un constructor, no debe llevar tipo de variable a devolver, ya que creará objetos tipo ColorFinder, pero sí debe indicar modificador de acceso, en este caso público, para poder emplearse fuera del paquete de la librería.

A continuación aparecen los parámetros de creación del objeto: una webcam, tres int de anchura, altura y tolerancia, y uno o varios arrays de tipo float para los colores. Java sólo admite argumentos de longitud variable cuando se declaran los últimos y son de tipos distintos, por lo que no puede utilizarse esta técnica en los colores y las tolerancias simultáneamente; se ha optado por proporcionar una tolerancia global en la creación del objeto y modificarla luego individualmente a conveniencia.

Por último, se ha incluido una excepción de tipo IllegalArgument, que puede ser lanzada por el constructor.

```
        for (float[] color : colors) {
            if (color.length != 3) {
                throw new IllegalArgumentException("color format must
be int array of size 3 containing {r,g,b}");
            }
        }
```

La primera acción a realizar por el constructor es comprobar la validez de los parámetros introducidos. Los 4 primeros parámetros aprovechan el tipado para comprobar su validez, lanzando excepciones si se emplea un tipo distinto, pero el parámetro variable colors admite cualquier número de arrays float, y solo debería admitir floats de 3 componentes, por lo que la excepción debe lanzarse manualmente. Además, se ha añadido un mensaje de error personalizado a la excepción, que informa de la causa que la ha provocado. Una excepción siempre causa la salida del método en el que se lanza y, si no es gestionada por un método de orden superior, la salida de la aplicación.

```
        this.targets.addAll(Arrays.asList(colors));
        for (float[] target : targets) {
            this.tol.add(tol);
            this.size2D.add(0);
        }
        this.webcam = webcam;
        this.width = width;
        this.height = height;
        this.readArray = new int[width][height];
    }
```

Por último, se inicializan las variables del objeto: se añaden los colores a la lista targets, se añade una tolerancia con el valor introducido a cada objetivo y se reserva espacio en memoria para el cálculo de tamaños, se asigna la webcam del objeto a la introducida, se fijan el alto y el ancho de la imagen y se reserva espacio en memoria para el array de lectura.

La versión sobrecargada del constructor realiza las mismas operaciones, con valores por defecto excepto para la cámara y los colores, que serán los únicos parámetros a introducir.

3.2.5 Método camStart

El primer método del objeto, que admite una versión con parámetros y otra versión sin parámetros, inicializa el resto de listas y prepara la cámara para la adquisición de datos, actualizando las dimensiones de la cámara. Es de tipo public, por lo que puede emplearse desde fuera de la librería, y void, por lo que no devuelve ninguna variable, pero no es estático, por lo que debe utilizarse asociado a un objeto ColorFinder.

```
    public void camStart(int w, int h) {
        this.width = w;
        this.height = h;
        this.readArray = new int[width][height];
        this.positions = new ArrayList<>();
        this.positionSpheric = new ArrayList<>();
        this.positions2D = new ArrayList<>();
```

Las primeras líneas corresponden a inicializaciones normales como las realizadas por el constructor, pero las dos últimas pueden requerir una breve explicación:

```
        webcam.setViewSize(new Dimension(w, h));
        webcam.open(true);
    }
```

El método de la clase Webcam setViewSize(Dimension d) viene definido por la librería de Sarxos. Permite ajustar el tamaño de la imagen desde origen, según el método empleado por cada cámara.

La mayoría de cámaras USB emplean un escalado por hardware de la imagen obtenida, ignorando la señal de parte de los sensores para obtener una imagen de menor tamaño con el mismo ángulo de visión. Además, muchos modelos varían su tiempo de refresco al emplear este método, ya que empleando menos sensores disminuye la carga de procesado en la cámara y el tiempo de transmisión de la imagen.

Otras cámaras, sin embargo, recortan la imagen ya generada, lo que altera tanto el tamaño de la imagen como el ángulo de visión en la misma proporción, sin obtener una variación apreciable en el tiempo de refresco. En estas cámaras es habitual emplear 640x480 como resolución de la imagen sin recortar.

En cuanto al método open(boolean async), también creado por Sarxos, cumple dos funciones.

La primera consiste en “abrir” la cámara, es decir, reservar el acceso a la misma por parte de esta aplicación, ya que en la mayoría de sistemas operativos actuales cada cámara sólo puede ser utilizada por una única aplicación, con el fin de evitar configuraciones contradictorias.

La segunda función es establecer el modo de sincronismo de la cámara, que en este caso se ha fijado como asíncrono. Este modo crea un buffer interno en el objeto webcam, en el que se almacena la imagen de la cámara, y un segundo “hilo” en el programa, encargado de actualizar dicho buffer cada vez que haya una imagen disponible. De esta forma, la aplicación no tiene que esperar la captura de una imagen nueva cada vez que llama a la cámara, si no que utiliza la última imagen captada de forma ininterrumpida al máximo de velocidad de la cámara, minimizando los tiempos de procesado en este punto.

3.2.6 Método camChange

Este método cierra la cámara actual, interrumpiendo su uso actual y permitiendo que sea usada por otra aplicación, la cambia por una nueva pasada como parámetro, y abre la cámara nueva:

```
public void camChange(Webcam webcam, int w, int h) {
        this.webcam.close();
        this.webcam = webcam;
        camStart(w, h);
    }
```

Admite una versión sobrecargada en la que el ancho y el alto actuales se mantienen con la cámara nueva.

3.2.7 Método find2D

Actualiza la lista de posiciones 2D según la posición actual de los objetos en la imagen de la cámara. Una vez actualizada, se puede utilizar el método getCenters2D() para obtener la lista.

```
public void find2D() {

        readArray = new int[width][height];
```

Inicializa el array de píxeles leídos a 0.

```
        if (targets.size() > positions2D.size()) {
            int tsize = targets.size();
            int psize = positions2D.size();
            for (int i = 0; i < tsize - psize; i++) {
                positions2D.add(new int[]{0, 0});
            }
        }
```

Si la lista de objetivos es mayor que la lista de posiciones en este momento, antes de empezar a calcular, añade posiciones adicionales a la lista hasta igualar las longitudes. De esta forma, se evitan desbordamientos.

```
        int[] count = new int[targets.size()];
        float[] xflo = new float[targets.size()];
        float[] yflo = new float[targets.size()];

        int argbRead;
        int rRead;
        int gRead;
        int bRead;
        Color write = new Color(0, 0, 0);
```

Declaración de variables internas del método. Las tres primeras se emplean en el cálculo de la posición del punto medio del objeto, las cuatro siguientes en el análisis de color de píxel y la última se emplea para pintar en la imagen con los colores adecuados en modo debug.

```
        image = getImage();
```

Este método está definido más adelante en el código. Es un método público que llama al método getImage de la cámara, devolviendo una BufferedImage con la imagen de la cámara en ese momento.

```
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
```

Las siguientes líneas se realizan por cada píxel en la imagen, recorriéndola de izquierda a derecha y de arriba a abajo, hasta el final del método.

```
                argbRead = image.getRGB(x, y);
                rRead = (argbRead >> 16) & 0xFF;
                gRead = (argbRead >> 8) & 0xFF;
                bRead = (argbRead) & 0xFF;
```

Estas líneas extraen el color del píxel y separan las componentes en las variables rRead (rojo), gRead (verde) y bRead (azul).

```
                if (debug) {
                    write = new Color(rRead / 2, gRead / 2, bRead /
2);
                }
```

Si está activo el modo debug, guarda en la variable write el color leído, pero atenuándolo a la mitad de brillo. Esto permitirá pintar encima de la imagen atenuada los colores a detectar, haciendo que resalten visualmente.

```
                for (int i = 0; i < targets.size(); i++) {
                    if ((rRead * targets.get(i)[0] + gRead *
targets.get(i)[1] + bRead * targets.get(i)[2] > 255 - tol.get(i))) {
                        xflo[i] = (xflo[i] * count[i] + x) /
(count[i] + 1);
                        yflo[i] = (yflo[i] * count[i] + y) /
(count[i] + 1);
                        count[i]++;
                        positions2D.set(i, new int[]{(int) xflo[i],
(int) yflo[i]});
                        if (debug) {
                            write = new Color((int)
(targets.get(i)[0] * 127 + 127 + rRead) / 2, (int) (targets.get(i)[1]
* 127 + 127 + gRead) / 2, (int) (targets.get(i)[2] * 127 + 127 +
bRead) / 2);
                        }
                        readArray[x][y] = i + 1;
                    }
```

Por cada color objetivo, si se detecta el objetivo en este píxel, actualiza xflo e yflo para el objetivo encontrado con la media de las coordenadas de todos los píxeles de este objetivo detectados hasta ahora, guarda el valor medio entre el color del píxel y el objetivo en la variable write si está activo el modo debug y guarda el índice del objetivo más 1 en el array de lectura.

De esta forma, si el modo debug está activo, el color almacenado en la variable write dará un píxel de un color más brillante que el de la imagen original, lo que hará que resalte aún más sobre la imagen atenuada de los píxeles no detectados.

En cuanto al índice del array de lectura, se le añade 1 al índice de los colores objetivos para conservar el orden de los objetivos, pero al mismo tiempo reservar el valor 0 inicial para los píxeles no detectados.

```
                    if (debug) {
                        image.setRGB(x, y, write.getRGB());
                    }
                }
            }
        }
    }
```

Por último, si el modo debug está activo, sustituye cada píxel por el color generado, ya sea el de la imagen atenuada del fondo o el color brillante del objetivo.

Debido a la forma de trabajar de java, mediante punteros, la imagen modificada en modo debug es la misma que se almacena en el bufer, por lo que una cámara más lenta que el algoritmo hará que la imagen del debug se oscurezca en cada iteración y los colores objetivo detectados se acerquen cada vez más al objetivo, hasta que se reciba un fotograma nuevo. No obstante, esto no tiene efecto en las coordenadas calculadas.

3.2.8 Método find2Dsizes

Actualiza las posiciones 2D usando el método anterior y calcula los tamaños de los objetos. Para el cálculo del tamaño, se analizan circunferencias centradas en el objeto y con radio cada vez mayor, pasando a la siguiente cuando se encuentra un pixel perteneciente al objeto, hasta que se realice una vuelta completa sin detectar ningún pixel.

```
     public void find2Dsizes(int sizeTol, double radTol) throws
IllegalArgumentException {
         if (sizeTol < 1) {
             throw new IllegalArgumentException("sizeTol must be at
least 1");
         }
         if (radTol == 0) {

             throw new IllegalArgumentException("radTol can't be 0");
         }
```

El método lanza excepciones si el incremento del radio es menor que 1 o si el incremento de ángulo es igual a 0, para evitar bucles infinitos en el algoritmo.

```
         int radius;
         double lastAlfa = 0;
         double alfa = 0;
         boolean found;
         find2D();
```

Se declaran las variables internas del método y se ejecuta el método find2D para actualizar las posiciones en la imagen.

```
         for (int i = 0; i < targets.size(); i++) {
```

A partir de aquí, las acciones se realizan una vez para cada color objetivo, utilizando la variable i para indicar el índice del objetivo actual; lo que sigue es el cálculo del radio del objeto.

```
             found = true;
             radius = sizeTol;
             alfa = 0;
```

En cada nuevo objetivo se reinician las variables relacionadas con la búsqueda del radio, volviendo a buscar desde el centro.

```
             if
(readArray[positions2D.get(i)[0]][positions2D.get(i)[1]] == i + 1) {
```

La búsqueda se realiza únicamente si el centro calculado pertenece al objeto, lo que ayuda a filtrar falsas detecciones debidas a posibles ruidos con el objeto fuera de pantalla. En esta situación, se mantendrá el objeto en la última posición conocida.

```
                 while (found) {
```

Mientras siga encontrándose el objeto,

```
                     found = false;
```

Se reinicia la variable de búsqueda.

```
                    while (!found && (alfa < lastAlfa + 2 * PI)) {
                        int x = positions2D.get(i)[0] + (int) (radius
* cos(alfa));
                        int y = positions2D.get(i)[1] + (int) (radius
* sin(alfa));
```

Y mientras no se haya encontrado en esta iteración y no se haya recorrido una vuelta completa, se actualiza la posición del pixel a comprobar según el radio y el ángulo actuales.

```
                        if (x >= 0 && x < width && y >= 0 && y <
height) {
                            if (readArray[x][y] == i + 1) {
                                radius = radius + sizeTol;
                                lastAlfa = alfa;
                                found = true;
                                size2D.set(i, radius);
                            }
```

Y, si el punto está dentro de la imagen, se comprueba en el array de lectura si pertenece al color objetivo, en cuyo caso se amplía el radio, se marca el objeto como encontrado y se fija el último ángulo de búsqueda al ángulo actual. Además, se fija el radio del objeto al actual.

```
                            if (debug) {
                                image.setRGB(x,                    y,
Color.WHITE.getRGB());
                            }
```

Si está activo el debug, se pinta el píxel analizado en blanco, de forma que en la imagen aparecerá una espiral desde el centro al borde del objeto, y un círculo rodeándolo, permitiendo observar el funcionamiento.

```
                        }
                        alfa = alfa + radTol;
                    }
                }
            }
        }
    }
```

Al final de cada iteración, se aumenta el ángulo para la siguiente, se haya encontrado o no el objeto. El ciclo continuará hasta que se dé una vuelta completa, momento en que se pasará al siguiente objetivo, dejando registrado el último radio para el que se encontró el objeto. Cuando no se encuentren nuevos objetivos, se habrá terminado la ejecución del método.

3.2.9 Método find3Dspherics

Este método implementa las ecuaciones calculadas en el apartado 2.1.1, para obtener las coordenadas esféricas del objeto a partir de la imagen de la cámara. Emplea el método anterior para obtener los datos de tamaño y posición de los objetos en la imagen.

Antes de actualizar la lista de coordenadas, comprueba que la lista incluye un juego de coordenadas para cada objetivo, y la amplía en caso contrario. A continuación, se ejecuta find2Dsizes con el fin de actualizar la librería. Por último, por cada objetivo, obtiene las coordenadas esféricas y las almacena en la lista, en el puesto correspondiente.

```
    public void find3Dspherics(int sizeTol, double radTol, double
angleOfView, double originTheta, double originPhi, double...
targetSize) {
        if (targets.size() > positionSpheric.size()) {
            int tsize = targets.size();
            int psize = positionSpheric.size();
            for (int i = 0; i < tsize - psize; i++) {
                positionSpheric.add(new double[]{0, 0, 0});
            }
        }
        find2Dsizes(sizeTol, radTol);
        for (int i = 0; i < targets.size(); i++) {
            positionSpheric.set(i, new double[]{(angleOfView / width)
*  positions2D.get(i)[0]  -  originTheta  -  PI  -  angleOfView  /  2,
(angleOfView  /  width)  *  positions2D.get(i)[1]  -  originPhi  -
angleOfView * height / (2 * width), targetSize[i] / (tan(size2D.get(i)
* angleOfView / width))});
        }
    }
```

3.2.10 Método find3D

Implementa las ecuaciones calculadas en el apartado 2.1.3, llamando al método anterior para actualizar toda la información necesaria, y ampliando la lista de posiciones en caso de necesidad. Su funcionamiento es análogo al del método anterior.

```
    public  void  find3D(int  sizeTol,  double  radTol,  double
angleOfView, double originTheta, double originPhi, double xCamera,
double  yCamera,  double  zCamera,  double  pixelPerUnit,  double...
targetSize) {
        if (targets.size() > positions.size()) {
            int tsize = targets.size();
            int psize = positions.size();
            for (int i = 0; i < tsize - psize; i++) {
                positions.add(new double[]{0, 0, 0});
            }
        }
        find3Dspherics(sizeTol,  radTol,  angleOfView,  originTheta,
originPhi, targetSize);
        for (int i = 0; i < targets.size(); i++) {
            positions.set(i,  new  double[]{(positionSpheric.get(i)[2]
*  sin(positionSpheric.get(i)[0])  *  cos(positionSpheric.get(i)[1])  +
xCamera)     *     pixelPerUnit,     (positionSpheric.get(i)[2]     *
sin(positionSpheric.get(i)[1])    +    yCamera)    *    pixelPerUnit,
(positionSpheric.get(i)[2]    *    cos(positionSpheric.get(i)[0])    *
cos(positionSpheric.get(i)[1]) + zCamera) * pixelPerUnit});
        }
    }
```

3.2.11 Método setTarget

Cambia el objetivo indicado mediante su índice por un objetivo pasado por parámetros.

```
    public void setTarget(int index, float[] target) {
        this.targets.set(index, target);
    }
```

3.2.12 Getters

Los getters son métodos empleados en java para leer variables privadas sin permitir el acceso directo a las mismas. Esta clase dispone de varios de ellos, aunque por su simplicidad no se van a comentar individualmente:

```
public List<double[]> getPositions() {
    return positions;
}

public List<double[]> getSpheric() {
    return positionSpheric;
}

public List<Integer> getSizes2D() {
    return size2D;
}

public List<int[]> getCenters2D() {
    return positions2D;
}

public String getCamName() {
    return webcam.getName();
}

public BufferedImage getImage() {
    return webcam.getImage();
}

public boolean getDebug() {
    return debug;
}
```

3.2.13 Método imageWithCenters

Devuelve la imagen de la cámara con los centros de los objetos marcados por una cruz blanca.

```
public BufferedImage imageWithCenters() {
    for (int[] position2D : positions2D) {
        drawCross(position2D[0],      position2D[1],      image,
Color.WHITE.getRGB());
    }
    return image;
}
```

3.2.14 Métodos para obtención de coordenadas como String

Devuelven un String con las coordenadas del tipo indicado en el nombre (cartesianas 2D, esféricas o cartesianas 3D). El String de coordenadas 2D, además, incluye los tamaños. Estos métodos son útiles en el debug de aplicaciones.

```
     public String position2DString(String coordinatesSeparator,
String separator) {
          StringBuilder sb = new StringBuilder();
          for (int[] position2D : positions2D) {
              sb.append(String.valueOf(position2D[0]));
              sb.append(coordinatesSeparator);
              sb.append(String.valueOf(position2D[1]));
              sb.append(separator);
          }
          for (int size : size2D) {
              sb.append(size);
              sb.append(separator);
          }
          return sb.toString();
     }

     public String sphericString(String coordinatesSeparator, String
separator) {
          StringBuilder sb = new StringBuilder();
          for (double[] position : positionSpheric) {
              sb.append(String.valueOf(position[0]));
              sb.append(coordinatesSeparator);
              sb.append(String.valueOf(position[1]));
              sb.append(coordinatesSeparator);
              sb.append(String.valueOf(position[2]));
              sb.append(separator);
          }
          return sb.toString();
     }

     public String cartesian3DString(String coordinatesSeparator,
String separator) {
          StringBuilder sb = new StringBuilder();
          for (double[] position : positions) {
              sb.append(String.valueOf(position[0]));
              sb.append(coordinatesSeparator);
              sb.append(String.valueOf(position[1]));
              sb.append(coordinatesSeparator);
              sb.append(String.valueOf(position[2]));
              sb.append(separator);
          }
          return sb.toString();
     }
```

3.2.15 Método drawCross

Dibuja cruces sobre la imagen proporcionada, en las coordenadas y del color indicados. Útil en el debug de aplicaciones. Devuelve la imagen generada.

```
    public static BufferedImage drawCross(int xpos, int ypos,
BufferedImage image, int color) {
        for (int dist = 0; dist < 10; dist++) {
            if (xpos + dist < image.getWidth()) {
                image.setRGB(xpos + dist, ypos, color);
            }
            if (xpos - dist > 0) {
                image.setRGB(xpos - dist, ypos, color);
            }
            if (ypos + dist < image.getHeight()) {
                image.setRGB(xpos, ypos + dist, color);
            }
            if (ypos - dist > 0) {
                image.setRGB(xpos, ypos - dist, color);
            }
        }
        return image;
    }
```

3.2.16 Método setDebug

Activa o desactiva el "modo debug" del objeto.

```
    public void setDebug(boolean value) {
        this.debug = value;
    }
```

3.2.17 Método setTol

Ajusta la tolerancia del objetivo indicado por los parámetros. Emplea recursividad (llamadas al mismo método) para rellenar la lista de tolerancias en caso necesario, simplificando el código.

```
    public void setTol(int tol, int index) throws
IllegalArgumentException {
        if (index > this.tol.size()) {
            if (index > targets.size()) {
                throw new IllegalArgumentException("Index out of
bounds");
            }
            this.tol.add(0);
            this.setTol(tol, index);
        } else {
            this.tol.set(index, tol);
        }
    }
```

4 EJEMPLOS DE APLICACIÓN

"Pour ce qui est de l'avenir, il ne s'agit pas de le prévoir, mais de le rendre possible."

(No se trata sólo de prever el futuro, sino de hacerlo posible)

-Antoine de Saint-Exupéry

Con el fin de probar el funcionamiento de todos los elementos del dispositivo y el software realizados, se han creado programas sencillos en java que aprovechan algunas de las características del dispositivo diseñado.

Estas aplicaciones se han construido a partir de las mismas librerías, por lo que para aprovechar el espacio, las librerías se han mantenido en una carpeta "lib" que deberá conservarse junto a los ejecutables. Cada programa de ejemplo consta de un único método, "main", responsable de su ejecución.

La utilidad práctica de estos ejemplos es, en realidad, limitada; su única función es demostrar el funcionamiento del sistema diseñado y el uso de la librería asociada, quedando pendiente la realización de programas más avanzados que saquen provecho de la tecnología propuesta.

Algunos ejemplos y propuestas de programas más útiles que podrían desarrollarse han sido recogidos en el siguiente capítulo.

4.1. config.jar

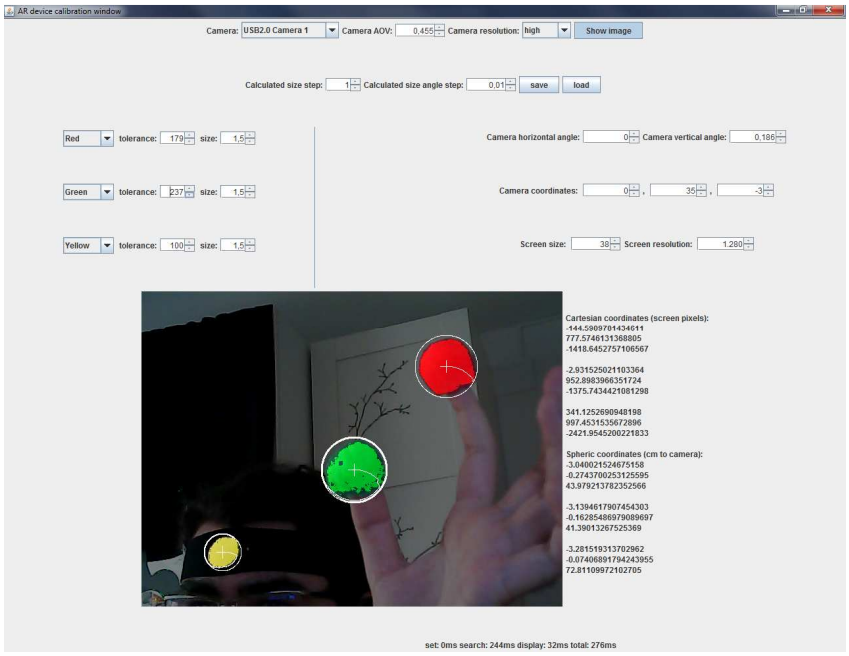

Figura 4-1: Aplicación de configuración. Elaboración propia.

Esta aplicación permite calibrar el dispositivo y generar un archivo “config.dat” con la configuración deseada, que podrá ser aprovechado por los siguientes programas.

Para ello, realiza el seguimiento de hasta 3 objetos de forma simultánea, y devuelve sus coordenadas tanto cartesianas, con origen configurable y medidas en “píxeles”, como esféricas, medidas en radianes y “cm”.

Cabe destacar que las unidades de longitud son en realidad configurables por el usuario; si se empleasen por ejemplo pulgadas para medir los tamaños de objetivos, las coordenadas de la cámara y el tamaño de pantalla, la distancia a la cámara contenida en las coordenadas esféricas estaría en pulgadas, y el programa seguiría funcionando adecuadamente.

El código de esta aplicación puede consultarse en el Anexo C.

4.1.1 Proceso de calibración

Para realizar la calibración del dispositivo, se recomienda seguir la siguiente secuencia de pasos:

1. **Selección de cámara y resolución:** En la parte superior de la ventana del programa, se selecciona la cámara de la lista desplegable y la resolución deseada. A continuación, se hace clic en el botón “show image” para ver la imagen capturada con una representación del procesado.
2. **Ajuste de los parámetros de tamaño:** Se recomienda fijarlos al menor valor posible.
3. **Selección de colores objetivo:** En la parte izquierda de la pantalla, se seleccionan los colores a detectar. Para los programas diseñados como ejemplo, se ha utilizado el primer color como dedal índice (rojo), el segundo como dedal pulgar (verde) y el tercero como bandana (amarillo), aunque el orden y los accesorios de futuras aplicaciones podría variar.
4. **Ajustes de detección:** También en la parte izquierda, se ajusta el tamaño de cada elemento, en las unidades que se desee, recomendado cm, y la tolerancia del sensor. Para ajustar la tolerancia, se recomienda mover el objeto por delante de la pantalla de forma similar a la esperada durante su uso, comprobando que se realiza correctamente el seguimiento. El objeto de la imagen debería permanecer “iluminado”, con una cruz marcando su centro, y rodeado por un círculo de píxeles blancos. Si se iluminan elementos fuera del objeto, se debe reducir la tolerancia. Se recomienda fijarla al mayor valor posible sin que esto ocurra.
5. **Ángulo de visión (AOV):** Si se conoce el ángulo de visión horizontal de la cámara, basta con introducir su valor en radianes. En caso contrario, puede emplearse como referencia uno de los objetos ya calibrados, manteniéndolo a una distancia conocida de la cámara y modificando el ángulo de visión hasta que la distancia del objeto a la cámara coincida con su tercera coordenada esférica, en las unidades empleadas anteriormente.
6. **Ángulos de cámara:** Miden la diferencia en radianes entre el objetivo de la cámara y el sistema de referencia. Si el sistema de referencia es el de una pantalla vertical, como un monitor, un ángulo de 0 corresponde a una cámara paralela a la pantalla, como una cámara de portátil, un ángulo recto positivo vertical representa el objetivo apuntando directamente hacia arriba, y un ángulo recto positivo horizontal representa el objetivo apuntando a la izquierda del usuario.
7. **Posición de cámara:** Miden la diferencia en posición entre el origen del sistema de coordenadas deseado y la cámara, en cada uno de los ejes, en las unidades empleadas. Los ejes corresponden habitualmente a los de la pantalla, siendo el eje X positivo hacia la derecha, desde el punto de vista del usuario, el Y positivo hacia abajo y el Z positivo hacia la parte posterior o el interior de la pantalla.
8. **Tamaños de la pantalla:** Tamaño en unidades empleadas y píxeles de la pantalla, medido en ambos casos en la misma dirección, por ejemplo horizontal. Se emplea en el cálculo de las coordenadas cartesianas según el sistema de referencia de la pantalla, por lo que fijar ambos a 1 devuelve la posición en las unidades empleadas en lugar de en píxeles. Algunas aplicaciones pueden utilizarlos en otros cálculos.

Una vez realizada la calibración, se recomienda pulsar el botón “save”. En la siguiente sesión, o al utilizar otra aplicación, bastará pulsar “load” para recuperar la información de calibración del archivo “config.dat”.

4.1.2 Lectura de datos

Junto a la imagen empleada para la calibración se recogen los datos de posición de cada elemento en coordenadas cartesianas (x, y, z), con el origen configurado, y esféricas (θ, φ, r), con origen en la cámara y la orientación configurada.

Bajo la imagen y la lista de datos se proporciona información sobre los tiempos de procesado, siendo "set" el tiempo empleado en lectura de los datos modificados en la interfaz de usuario, "search" el tiempo de cálculo de las coordenadas y "display" el tiempo empleado en generar la pantalla. Debe tenerse en cuenta, no obstante, que la cámara tiene un límite de capturas por segundo, y que el tiempo de espera hasta la siguiente captura se incluye en la categoría "search". Disminuir la resolución de la captura acelera el procesado, aunque no siempre aumenta las capturas por segundo de la cámara, y aumenta la sensibilidad al ruido, especialmente en la distancia r de las coordenadas esféricas y en las coordenadas cartesianas.

4.2. ARMouse.jar

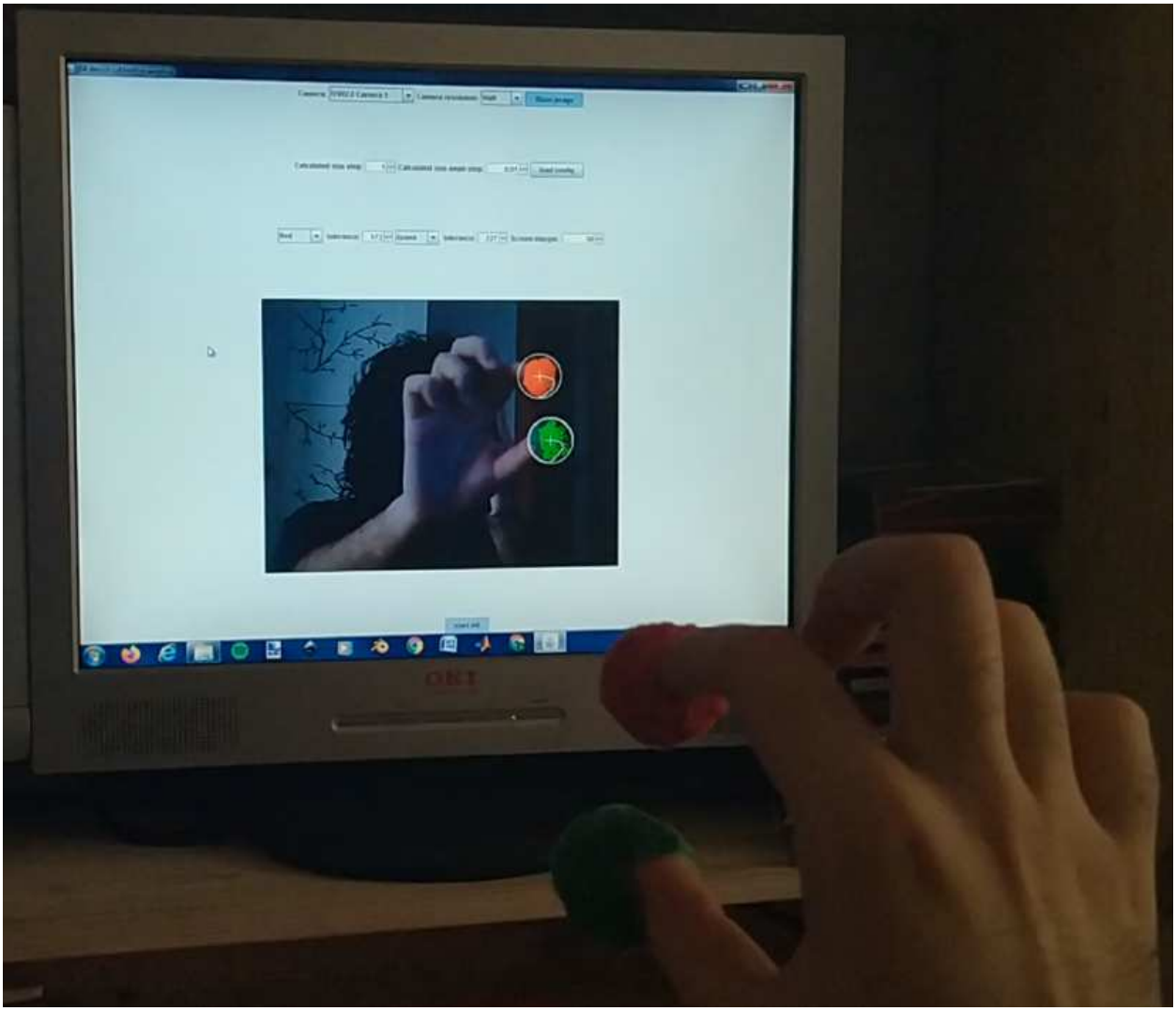

Figura 4-2: Ratón AR. Elaboración propia.

Esta aplicación implementa un control de ratón sencillo basado en la imagen 2D de la cámara. Incorpora una interfaz parecida a la de la aplicación anterior, aunque con menos elementos, y permite la carga de configuración desde un archivo config.dat, pero no la modificación del archivo.

El control se realiza a través de la posición de dos elementos, correspondientes a los dedales índice y pulgar (primer y segundo elementos de la configuración). Concretamente, el ratón se situará según la posición del pulgar, mientras que se podrán ejecutar acciones de clic y clic derecho al acercar y alejar los dedos entre sí,

respectivamente.

La interfaz contiene un campo numérico nuevo, “Screen margin”, que representa la distancia entre el borde de la imagen obtenida de la cámara y el borde de la pantalla, y un botón, “start AR”, que inicia el control del ratón al pulsarse, hasta que sea pulsado de nuevo.

Cerrar la ventana del programa en cualquier momento detendrá su ejecución y devolverá al ratón tradicional el control del cursor.

El código de esta aplicación puede consultarse en el Anexo D.

4.3. ARImage.jar

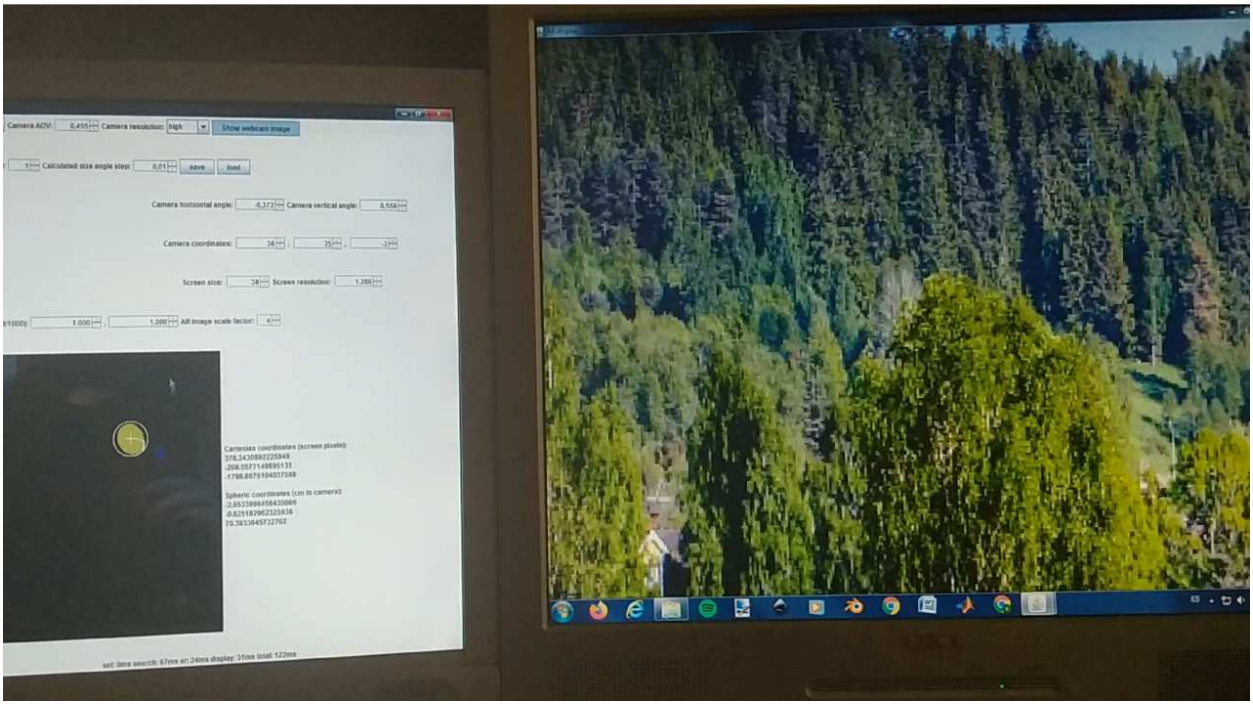

Figura 4-3: Visor de imágenes AR. Elaboración propia.

Esta aplicación aprovecha la información de la posición 3D de la cabeza del usuario para intentar “corregir” la perspectiva, creando la ilusión de que la imagen de la pantalla se encuentra a mayor distancia, imitando el paisaje lejano visto a través de una ventana. La imagen se obtiene mediante archivo, siendo válido cualquier archivo jpg guardado junto a la aplicación con el nombre “img.jpg”.

La interfaz se divide en dos pantallas, siendo una de ellas similar a la aplicación de calibración, a la que se han añadido dos controles para ajustar el tamaño de la imagen visible y un tercero para ajustar la resolución, y la otra ventana consistente en la imagen AR. Aunque es posible modificar y guardar la calibración del dispositivo desde esta pantalla, por motivos de rendimiento sólo se calculan las coordenadas del tercer objetivo. Cerrar la ventana de configuración no detiene la aplicación.

De las tres aplicaciones diseñadas, esta es la que más recursos de procesado requiere normalmente, ya que necesita analizar la posición del usuario, obteniendo coordenadas cartesianas, y generar la imagen en cada ciclo, aunque el tiempo total de procesado dependerá de la unidad de procesamiento y del tamaño de la ventana. El factor de escala de la imagen puede ayudar a minimizar el procesado, al dividir la resolución de la imagen generada.

Para generar la imagen AR, se ha seguido un proceso inverso al empleado en el cálculo de las coordenadas del usuario; para cada pixel en la ventana, se calculan los ángulos esféricos del pixel, tomando como referencia al usuario, y se asigna el color correspondiente a un pixel de la imagen según estas coordenadas. El tamaño de la imagen dependerá del tamaño de la pantalla, por lo que reducir el tamaño de esta última acelerará el cálculo.

El código de esta aplicación puede consultarse en el Anexo E.

4.4. Conclusiones experimentales extraídas de los ejemplos

Se esperaba una precisión relativamente elevada, especialmente si lograba eliminarse la mayor parte del ruido, una ausencia de derivas como las presentes en otros dispositivos VR o AR y una velocidad de procesado menor a la de dichos dispositivos.

Se ha comprobado que la precisión ha sido suficiente, aunque se ve bastante influenciada por el dispositivo de captura empleado, especialmente ante posibles errores de calibración.

En cuanto al tiempo de procesado, aunque depende del equipo empleado, se consiguen tiempos de procesado de entre 50ms (en la aplicación de configuración) y 70ms (en el visor de imágenes), llegando a triplicarse en "modo debug", empleando un procesador Intel Core i5 a 2.7GHz.

En cuanto a consumo de RAM, la aplicación de configuración utiliza en torno a 300MB, mientras que el visor de imágenes supera los 600MB de RAM. Ninguna de las aplicaciones de ejemplo utiliza recursos de la tarjeta gráfica.

Por tanto, aunque se propuso en los primeros capítulos del proyecto dividir el procesado entre un segundo dispositivo, por ejemplo una Raspberry, encargado de obtener las coordenadas de los objetivos, y el dispositivo principal, encargado del resto de la aplicación, se ha considerado que, al tratarse de una aplicación que resulta de más utilidad si funciona en tiempo real, y al no observarse una mejoría en el tiempo de procesado por tratarse habitualmente de aplicaciones secuenciales, y siendo el consumo de RAM fácilmente asumible por la mayoría de dispositivos actuales, el protocolo de comunicación sólo serviría para generar un retardo indeseado en la aplicación.

Además, la librería empleada para la obtención de imágenes desde la cámara actualmente no soporta de forma integrada el módulo de cámara de la Raspberry, soporte que se encuentra entre las funciones planeadas pero que de momento requeriría librerías adicionales. En cambio, sí se detectan cámaras USB conectadas a Raspberry.

Se ha considerado que, siendo el objetivo una demostración de funcionamiento de la librería creada, el uso de cámaras USB o integradas para PC sería suficiente por el momento, a la espera de la actualización de la librería webcam capture.

4.5. Posibles futuras aplicaciones

A partir de los resultados observados, y viendo cómo se han aplicado otras tecnologías parecidas, es posible hacer una estimación previa de cómo podría aplicarse esta tecnología en un futuro. Dada la sencillez de la librería creada y del hardware asociado, y dado que la licencia del resto de librerías empleadas lo permiten, podría favorecerse la aparición de estas y otras aplicaciones si se publica de forma abierta, por ejemplo con una licencia Creative Commons, a través de una plataforma para desarrolladores, como podría ser GitHub.

4.5.1 Aplicaciones de diseño y visualización

Una de las aplicaciones en las que más impacto podría tener esta tecnología es en el ámbito del diseño 3D.

La realidad aumentada ya se utiliza para visualización de elementos 3D a través de dispositivos de pequeño tamaño, como Smartphones, empleando a menudo patrones bidimensionales a modo de referencia sobre la que aparece el contenido virtual, y la realidad virtual está empezando a utilizarse no solo en visualización, sino también en diseño, por las facilidades que ofrece en manipulación de elementos virtuales frente al enfoque clásico basado en proyecciones 2D, especialmente en ámbitos artísticos.

El uso de la tecnología AR diseñada en este trabajo presenta ventajas importantes en este ámbito frente a ambas competidoras.

Por un lado, frente a la AR móvil, se dispone de un sistema de manipulación avanzada similar al de la VR, y, al haber sido diseñada para dispositivos de mayor tamaño, también puede beneficiarse de una mayor potencia de cálculo, lo cual otorga una mayor versatilidad, especialmente en el ámbito del diseño.

Por otro lado, frente a la VR, si bien presenta un campo de visión más limitado para el usuario, las desventajas

por su uso continuado a largo plazo son mucho menores y se encuentran mejor documentadas, al aprovechar displays clásicos cuyos efectos sobre la salud son más conocidos. Además, la ausencia de "gafas" VR facilita la consulta de documentación, referencias u otros elementos externos al entorno virtual.

En cuanto a la interacción del usuario, se observa que los mandos aquí diseñados presentan características similares a los empleados en VR, al permitir la detección de su ubicación tridimensional, aunque carecen de otros medios de interacción, como botones o palancas. Sobre este asunto se hablará con más detenimiento en el último punto de este capítulo, donde se hace un pequeño análisis sobre sus posibilidades en un futuro cercano.

Por tanto, si bien la tecnología aquí diseñada no pretende desplazar al resto de tecnologías presentes, sí que parece viable su coexistencia, aplicándose a ámbitos distintos.

4.5.2 Aplicaciones de carácter lúdico y videojuegos

Otra aplicación habitual tanto de la VR como de la AR son los videojuegos y el entretenimiento.

En este caso, la tecnología aquí diseñada se queda atrás al compararse con la portabilidad de la AR móvil o con el nivel de inmersión de la VR, por lo que parece que el público objetivo en este ámbito será mucho más reducido.

No obstante, el precio de la tecnología empleada es menor que el de los actuales sistemas de VR, y algunos videojuegos específicamente diseñados para aprovechar este sistema podrían conseguir abrirse paso en el mercado, por lo que no se descarta esta aplicación.

4.6. Conclusiones del trabajo

El objetivo del trabajo era diseñar y realizar un sistema de realidad aumentada para dispositivos de sobremesa, por lo que una vez conseguido se ha centrado en analizar las limitaciones propias del mismo, consiguiendo un dispositivo de bajo coste fácilmente replicable, incluso a nivel de usuario, que permita un desarrollo posterior integrando su tecnología con otros elementos hardware o creando nuevas aplicaciones software que aprovechen sus funciones.

Entre sus ventajas, además de las económicas, se ha obtenido un dispositivo relativamente preciso, dependiendo del hardware empleado, capaz de obtener posición absoluta en un sistema de referencia dado, inmune a deriva y con una velocidad de procesado elevada.

La principal desventaja es su sensibilidad al ruido, que depende del hardware empleado y del entorno en que se instale. Una cámara de baja resolución se ve más afectada que una de alta, ya que la variación de color de un pixel en el borde del objeto implica un cambio de distancia entre la cámara y el objeto de mayor proporción cuanto menor sea la resolución de la imagen. De igual manera, un fondo colorido o una iluminación escasa, o de un color concreto, favorecen la aparición de píxeles fuera del objeto que desplazan su "centro" y alteran la medida del tamaño, lo cual se hace más evidente empleando cámaras con menor resolución. Una tercera fuente de ruido proviene de los movimientos rápidos, que hacen que los objetos aparezcan deformados y borrosos en la imagen captada, efecto que se minimiza con una mayor velocidad de captura, normalmente asociada a una menor resolución.

En general, no obstante, se puede concluir que el bajo coste y la precisión obtenida pueden ser motivación suficiente para fomentar el desarrollo y la evolución de este sistema, contando con sus limitaciones.

4.7. El futuro de la tecnología

Aunque es difícil prever todas las posibles aplicaciones que este sistema pueda acabar teniendo, sí que pueden diferenciarse algunas vías de evolución orientadas a expandir su uso o a corregir sus deficiencias.

En este último punto, a lo largo del trabajo se ha estado empleando únicamente el sistema de seguimiento 3D

desarrollado, sin combinarlo con ninguna otra tecnología preexistente que no fuera parte intrínseca del sistema, lo que deja un importante margen de mejora simplemente añadiendo elementos que actualmente se encuentran de forma habitual en todo tipo de dispositivos.

Una mejora básica del dispositivo podría consistir en la adición de un sistema de comunicación inalámbrica y botones, sustituyendo los dedales por un mando diseñado especialmente a tal efecto. También podría añadirse algún tipo de iluminación interior a los mandos, obteniendo un sistema parecido al empleado por la consola PlayStation de Sony para sus aplicaciones VR, que realiza una triangulación basada en dos cámaras separadas una distancia fija.

También podrían añadirse sensores como giroscopios y acelerómetros, convirtiendo el mando “pasivo” en un mando “activo”, pero manteniendo el sistema diseñado para el cálculo de la posición, actualizado de forma continua en lugar de limitado a la función de sincronización como en algunos dispositivos actuales con tecnología parecida. Esto permitiría la realización de movimientos más rápidos e incluso el seguimiento del mando “fuera de cámara”.

Otra posible opción de mejora, centrada en el software en lugar del hardware, consiste en el desarrollo de sistemas de procesado de la imagen más avanzados que el empleado en la librería actual, permitiendo una mejor eliminación de ruidos, o soporte para la estimación de trayectorias durante movimientos rápidos.

También queda pendiente la realización de sistemas de calibración más cómodos para el usuario final, la complejidad de esta calibración dependerá de la aplicación a desarrollar, pero se pueden incluir algunas calibraciones predefinidas que se ajusten a determinados dispositivos comunes en los que algunos de los datos sean conocidos; por ejemplo, la mayoría de portátiles incorporan una cámara integrada, cuyos datos y posición podrían almacenarse en una base de datos de la aplicación y detectarse automáticamente.

En definitiva, el sistema aquí desarrollado dista mucho de ser un producto comercial, pero abre las puertas al desarrollo de este tipo de tecnología.

REFERENCIAS

Arora, D. (s.f.). *Understanding time complexity with simple examples*. Recuperado el 3 de septiembre de 2020, de https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/

Baum, L. (1901). *The Master Key*.

Dominio público. (11 de agosto de 2011). *Kinect*. Recuperado el 12 de agosto de 2020, de https://en.wikipedia.org/wiki/File:Xbox-360-Kinect-Standalone.png

Dominio público. (30 de agosto de 2010). *Wii remote*. Recuperado el 12 de agosto de 2020, de https://es.wikipedia.org/wiki/Wiimote#/media/Archivo:Wiimote-Safety-First.jpg

Dp Plus. (s.f.). *Aprende más acerca de la profundidad de color*. Recuperado el 3 de septiembre de 2020, de https://dpplus.es/impresion/profundidad-de-color/

Firyn, B. (s.f.). *Webcam Capture in Java*. Recuperado el 13 de agosto de 2020, de http://webcam-capture.sarxos.pl/

Frijters, J. (s.f.). *ikvm.net*. Recuperado el 7 de septiembre de 2020, de https://www.ikvm.net/

GameDev Academy. (s.f.). *Best game engines of 2020*. Recuperado el 7 de septiembre de 2020, de https://gamedevacademy.org/best-game-engines/

Hùng, L. T. (31 de 7 de 2019). *Old 16th century farms surrounded by green forest and fields (imagen libre de derechos)*. Recuperado el 12 de agosto de 2020, de https://pxhere.com/es/photo/1593914

Lee, J. C. (2007). *Johnny Chung Lee*. Recuperado el 12 de agosto de 2020, de http://johnnylee.net/projects/wii/

Microsoft. (s.f.). *Visual Basic 6.0 Documentation*. Recuperado el 7 de septiembre de 2020, de https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6.0-documentation

Shawn. (6 de diciembre de 2008). *CRJ HUD (3096245209)*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Archivo:CRJ_HUD_(3096245209).jpg

Sutherland, I. (1968). A Head Mounted Three Dimensional Display. Harvard University.

TIOBE. (6 de septiembre de 2020). *index*. Recuperado el 7 de septiembre de 2020, de https://www.tiobe.com/tiobe-index/

Wikipedia. (7 de agosto de 2020). *Augmented reality*. Recuperado el 9 de agosto de 2020, de https://en.wikipedia.org/wiki/Augmented_reality

Wikipedia. (28 de julio de 2020). *Virtual reality*. Recuperado el 9 de agosto de 2020, de https://en.wikipedia.org/wiki/Virtual_reality

ANEXO A: OBJETIVOS DE DISEÑO

A continuación se recoge una lista de objetivos a cumplir por el dispositivo diseñado. Para establecer un orden de prioridades, para que el trabajo se considere completamente terminado debe incluir, al menos, todos los objetivos que contienen "DEBE". Los objetivos que contengan "PUEDE" serán los siguientes en orden de prioridad, por lo que su no inclusión requerirá una justificación basada en resultados posteriores a la propuesta, mientras que los marcados como "PODRÍA" son sugerencias de desarrollo, características planeadas a futuro o elementos condicionales. De igual manera, "NO DEBE", "PUEDE NO" y "PODRÍA NO" representan objetivos negativos.

Cada objetivo ha sido numerado para facilitar su identificación, apareciendo en negrita. Si el objetivo ya ha sido implementado o si existen notas adicionales, se incluirán a continuación. De igual forma, objetivos relacionados entre sí o subobjetivos se incluirán de forma anidada. Los subobjetivos quedan subordinados a la prioridad del objetivo del que depende, de forma que un "DEBE" subordinado a un "PODRÍA" implica que el subobjetivo es obligatorio siempre que se implemente el objetivo principal, que es condicional.

1. Objetivos básicos del sistema

1.1 El sistema DEBE ser capaz de obtener al menos las coordenadas de posición 3D de un objeto. Implementado.

1.1.1 El sistema PUEDE obtener las coordenadas de posición de varios objetos distintos. Implementado, hasta 6 objetos simultáneamente.

1.1.2 El sistema PODRÍA obtener las coordenadas de orientación 3D de un objeto. No se ha implementado de forma nativa; puede obtenerse orientación empleando 2 marcadores por objeto, si la aplicación lo requiere.

1.2 El sistema DEBE optimizarse para un consumo de recursos reducido. Implementado.

2. Objetivos de las aplicaciones de demostración

2.1 El sistema DEBE acompañarse de al menos una aplicación de demostración. Implementadas 3.

2.1.1 El sistema PUEDE acompañarse de una aplicación de demostración que muestre las coordenadas calculadas de los objetos detectados. Implementado.

2.1.1.1 Esta aplicación PUEDE permitir la visualización del procesado de la imagen de forma gráfica. Implementado.

2.1.1.2 Esta aplicación PODRÍA emplearse para la calibración del dispositivo para otras aplicaciones. Implementado.

2.1.2 El sistema PUEDE acompañarse de una aplicación de demostración que permita el control del ordenador detectando la posición de la mano. Implementado.

2.1.2.1 La aplicación DEBE ser capaz de mover el ratón imitando movimientos de la mano del usuario. Implementado.

2.1.2.2 La aplicación PUEDE ser capaz de realizar acciones de "clic izquierdo" al detectar un gesto predeterminado del usuario. Implementado.

2.1.2.3 La aplicación PODRÍA ser capaz de realizar acciones de "clic derecho" al detectar un gesto predeterminado del usuario. Implementado.

2.1.2.4 La aplicación DEBE permitir la calibración del sistema o la carga de una calibración previa generada por otra aplicación. Implementado en ambas formas.

2.1.3 El sistema PUEDE acompañarse de una aplicación de demostración que aproveche la posición de la cabeza del usuario para generar algún tipo de ilusión en la visualización. Implementado.

2.1.3.1 La aplicación DEBE ser capaz de transformar una imagen en función de la posición de la cabeza del usuario. Implementado.

2.1.3.2 La aplicación PUEDE transformar una imagen proporcionada mediante archivo. Implementado.

2.1.3.3 La aplicación PODRÍA transformar una imagen del escritorio del PC bajo ella, sin interferir en la interacción del usuario. No implementado; se intentó el uso de un monitor virtual, pero Java no permite la creación de hardware virtual de este nivel, y otros métodos implican conocimientos avanzados de cada sistema operativo a tener en cuenta. Una opción es emplear capturas de pantalla de un monitor distinto al que ejecuta la aplicación, pero esto limita la utilidad de la aplicación. Otras opciones analizadas se escapaban del ámbito de esta demo, pasando a ser posibles aplicaciones completas.

2.1.3.3.1 La aplicación PODRÍA implementar el control de ratón AR de la aplicación anterior. No implementado al no haberse cumplido el objetivo principal asociado.

2.1.3.4 La aplicación DEBE permitir la calibración del sistema o la carga de una calibración previa generada por otra aplicación. Implementado en ambas formas.

2.1.3.5 La imagen transformada PUEDE seguir el movimiento de la cabeza del usuario mediante un desplazamiento. Implementado de forma indirecta por el siguiente objetivo.

2.1.3.6 La imagen transformada PODRÍA deformarse para corregir la perspectiva del usuario, de forma que los pixeles más alejados físicamente se hagan más grandes, creando la ilusión de que la imagen está siempre de frente y no se ve afectada por la distancia a la pantalla. Implementado.

2.1.4 El sistema PODRÍA acompañarse de una aplicación de demostración que aproveche la posición de la cabeza y la mano para interactuar con un entorno 3D. No implementado. Queda propuesta para su futura realización, dada la complejidad que podría llegar a alcanzar.

2.1.4.1 La aplicación DEBE permitir la visualización de elementos 3D.

2.1.4.2 La aplicación PODRÍA permitir la edición de elementos 3D, aprovechando la posición 3D de la mano a modo de cursor.

3. Objetivos de procesado de información

3.1 El procesado del sistema AR DEBE poder hacerse en el dispositivo principal (por ejemplo, PC). Implementado.

3.2 El procesado del sistema AR PODRÍA hacerse en un dispositivo auxiliar. No implementado de forma nativa. Se ha estimado que la disminución en los recursos de procesado empleados en el dispositivo principal no justifica el retardo inducido por el protocolo de comunicación. No obstante, es posible enviar datos desde una aplicación final a otra por cualquier vía, incluyendo comunicación entre dispositivos.

4. Objetivos económicos

4.1 El sistema DEBE tener un bajo coste económico. Implementado.

4.1.1 El sistema PUEDE aprovechar elementos de uso común sin un coste adicional. Implementado.

4.1.2 El sistema PUEDE tener partes fácilmente replicables con distintos materiales según disponibilidad. Implementado.

ANEXO B: CÓDIGO FUENTE DE LA LIBRERÍA

```
1   package desktopAR;
2
3   import com.github.sarxos.webcam.Webcam;
4   import java.awt.Color;
5   import java.awt.Dimension;
6   import java.awt.image.BufferedImage;
7   import static java.lang.Math.PI;
8   import static java.lang.Math.cos;
9   import static java.lang.Math.sin;
10  import static java.lang.Math.tan;
11  import java.util.ArrayList;
12  import java.util.Arrays;
13  import java.util.List;
14
15  /**
16   * This class implements an object that allows for colored object
17 detection.
18   * Requires the webcam-capture library by sarxos.
19   *
20   * @author Jaime Palomo Iranzo
21   */
22  public class ColorFinder {
23
24      /**
25       * Red color in ColorFinder format. Colors range from 255 (pure
26 red) to -510
27       * (pure cyan)
28       */
29      static public final float[] RED = {1, -1, -1};
30
31      /**
32       * Green color in ColorFinder format. Colors range from 255 (pure
33 green) to
34       * -510 (pure magenta)
35       */
36      static public final float[] GREEN = {-1, 1, -1};
37
38      /**
39       * Blue color in ColorFinder format. Colors range from 255 (pure
40 blue) to
41       * -510 (pure yellow)
42       */
43      static public final float[] BLUE = {-1, -1, 1};
44
45      /**
46       * Cyan color in ColorFinder format. Colors range from 255 (pure
47 cyan) to
48       * -127 (pure red)
49       */
50      static public final float[] CYAN = {-0.5f, 0.5f, 0.5f};
51
52      /**
```

```
53          * Magenta color in ColorFinder format. Colors range from 255
54    (pure magenta)
55          * to -127 (pure green)
56          */
57         static public final float[] MAGENTA = {0.5f, -0.5f, 0.5f};
58
59         /**
60          * Yellow color in ColorFinder format. Colors range from 255
61    (pure yellow)
62          * to -127 (pure blue)
63          */
64         static public final float[] YELLOW = {0.5f, 0.5f, -0.5f};
65
66         /**
67          * White color in ColorFinder format. Colors range from 230 (pure
68    white) to
69          * 0 (pure black)
70          */
71         static public final float[] WHITE = {0.3f, 0.3f, 0.3f};
72
73         /**
74          * Black color in ColorFinder format. Colors range from 0 (pure
75    black) to
76          * -230 (pure white)
77          */
78         static public final float[] BLACK = {-0.3f, -0.3f, -0.3f};
79
80         private List<float[]> targets = new ArrayList<>();
81         private List<double[]> positions = new ArrayList<>();
82         private List<double[]> positionSpheric = new ArrayList<>();
83         private List<int[]> positions2D = new ArrayList<>();
84         private List<Integer> size2D = new ArrayList<>();
85         private List<Integer> tol = new ArrayList<>();
86         private Webcam webcam;
87
88         private int width;
89         private int height;
90
91         private boolean debug = false;
92         private BufferedImage image;
93         private int[][] readArray;
94
95         /**
96          * Constructor for the ColorFinder class. Allows color based
97    image analysis
98          * for webcams. Requires the webcam-capture library by sarxos.
99          *
100         * @param webcam webcam to use
101         * @param width width of the image to obtain. Must be webcam-
102   compatible.
103         * @param height height of the image to obtain. Must be webcam-
104   compatible.
105         * @param tol base tolerance for all targets
106         * @param colors {r,g,b} vector of weights. Use with the provided
107   color
108         * constants.
```

```
109         * @throws IllegalArgumentException when using a wrong colors
110  param format.
111         */
112        public ColorFinder(Webcam webcam, int width, int height, int tol,
113  float[]... colors) throws IllegalArgumentException {
114            for (float[] color : colors) {
115                if (color.length != 3) {
116                    throw new IllegalArgumentException("color format must
117  be int array of size 3 containing {r,g,b}");
118                }
119            }
120            this.targets.addAll(Arrays.asList(colors));
121            for (float[] target : targets) {
122                this.tol.add(tol);
123                this.size2D.add(0);
124            }
125            this.webcam = webcam;
126            this.width = width;
127            this.height = height;
128            this.readArray = new int[width][height];
129        }
130
131        /**
132         * Constructor for the ColorFinder class. Allows color based
133  image analysis
134         * for webcams. Requires the webcam-capture library by sarxos.
135         *
136         * @param webcam webcam to use
137         * @param colors {r,g,b} vector of weights. Use with the provided
138  color
139         * constants.
140         * @throws IllegalArgumentException when using a wrong colors
141  param format.
142         */
143        public ColorFinder(Webcam webcam, float[]... colors) throws
144  IllegalArgumentException {
145            for (float[] color : colors) {
146                if (color.length != 3) {
147                    throw new IllegalArgumentException("color format must
148  be int array of size 3 containing {r,g,b}");
149                }
150            }
151            this.targets.addAll(Arrays.asList(colors));
152            for (float[] target : targets) {
153                this.tol.add(50);
154                this.size2D.add(0);
155            }
156            this.webcam = webcam;
157            this.width = 640;
158            this.height = 480;
159            this.readArray = new int[width][height];
160        }
161
162        /**
163         * Open the camera. Clear calculated data.
164         *
```

```
165         * @param w width of the image to obtain. Must be webcam-
166 compatible.
167         * @param h height of the image to obtain. Must be webcam-
168 compatible.
169         */
170        public void camStart(int w, int h) {
171            this.width = w;
172            this.height = h;
173            this.readArray = new int[width][height];
174            this.positions = new ArrayList<>();
175            this.positionSpheric = new ArrayList<>();
176            this.positions2D = new ArrayList<>();
177            webcam.setViewSize(new Dimension(w, h));
178            webcam.open(true);
179        }
180
181        /**
182         * Open the camera
183         *
184         */
185        public void camStart() {
186            camStart(width, height);
187        }
188
189        /**
190         * Change camera and open the new one
191         *
192         * @param webcam new webcam
193         * @param w width of the image to obtain. Must be webcam-
194 compatible.
195         * @param h height of the image to obtain. Must be webcam-
196 compatible.
197         */
198        public void camChange(Webcam webcam, int w, int h) {
199            this.webcam.close();
200            this.webcam = webcam;
201            camStart(w, h);
202        }
203
204        /**
205         * Change camera and open the new one
206         *
207         * @param webcam new webcam
208         */
209        public void camChange(Webcam webcam) {
210            camChange(webcam, width, height);
211        }
212
213        /**
214         * Update 2D average positions of the ColorFinder target colors.
215 Use
216         * getCenters2D() to get the updated values.
217         *
218         */
219        public void find2D() {
220            readArray = new int[width][height];
221            if (targets.size() > positions2D.size()) {
```

```
222                 int tsize = targets.size();
223                 int psize = positions2D.size();
224                 for (int i = 0; i < tsize - psize; i++) {
225                     positions2D.add(new int[]{0, 0});
226                 }
227             }
228             int[] count = new int[targets.size()];
229             float[] xflo = new float[targets.size()];
230             float[] yflo = new float[targets.size()];
231 
232             int argbRead;
233             int rRead;
234             int gRead;
235             int bRead;
236             Color write = new Color(0, 0, 0);
237             image = getImage();
238             for (int y = 0; y < height; y++) {
239                 for (int x = 0; x < width; x++) {
240                     argbRead = image.getRGB(x, y);
241                     rRead = (argbRead >> 16) & 0xFF;
242                     gRead = (argbRead >> 8) & 0xFF;
243                     bRead = (argbRead) & 0xFF;
244 
245                     if (debug) {
246                         write = new Color(rRead / 2, gRead / 2, bRead /
247 2);
248                     }
249                     for (int i = 0; i < targets.size(); i++) {
250 
251                         if  ((rRead  *  targets.get(i)[0]  +  gRead  *
252 targets.get(i)[1] + bRead * targets.get(i)[2] > 255 - tol.get(i))) {
253                             xflo[i]  =  (xflo[i]  *  count[i]  +  x)  /
254 (count[i] + 1);
255                             yflo[i]  =  (yflo[i]  *  count[i]  +  y)  /
256 (count[i] + 1);
257                             count[i]++;
258                             positions2D.set(i,  new  int[]{(int)  xflo[i],
259 (int) yflo[i]});
260                             if (debug) {
261                                 write       =       new       Color((int)
262 (targets.get(i)[0] * 127 + 127 + rRead) / 2, (int) (targets.get(i)[1]
263 *  127  +  127  +  gRead)  /  2,  (int)  (targets.get(i)[2]  *  127  +  127  +
264 bRead) / 2);
265                             }
266                             readArray[x][y] = i + 1;
267                         }
268                         if (debug) {
269                             image.setRGB(x, y, write.getRGB());
270                         }
271                     }
272                 }
273             }
274         }
275 
276         /**
277          * Update 2D positions and sizes.
```

```
278         *
279         * @param sizeTol size tolerance in camera pixels. Positive
280 integer.
281         * @param radTol angular tolerance for the search algorithm.
282 Smaller
283         * tolerance makes it more precise, but slower. Can't be 0.
284         * @throws IllegalArgumentException if argument conditions aren't
285 met.
286         */
287        public void find2Dsizes(int sizeTol, double radTol) throws
288 IllegalArgumentException {
289            if (sizeTol < 1) {
290                throw new IllegalArgumentException("sizeTol must be at
291 least 1");
292            }
293            if (radTol == 0) {
294                throw new IllegalArgumentException("radTol can't be 0");
295            }
296            int radius;
297            double lastAlfa = 0;
298            double alfa = 0;
299            boolean found;
300            find2D();
301            for (int i = 0; i < targets.size(); i++) {
302                found = true;
303                radius = sizeTol;
304                alfa = 0;
305                if
306 (readArray[positions2D.get(i)[0]][positions2D.get(i)[1]] == i + 1) {
307                    while (found) {
308                        found = false;
309                        while (!found && (alfa < lastAlfa + 2 * PI)) {
310                            int x = positions2D.get(i)[0] + (int) (radius
311 * cos(alfa));
312                            int y = positions2D.get(i)[1] + (int) (radius
313 * sin(alfa));
314                            if (x >= 0 && x < width && y >= 0 && y <
315 height) {
316                                if (readArray[x][y] == i + 1) {
317                                    radius = radius + sizeTol;
318                                    lastAlfa = alfa;
319                                    found = true;
320                                    size2D.set(i, radius);
321                                }
322                                if (debug) {
323                                    image.setRGB(x,                    y,
324 Color.WHITE.getRGB());
325                                }
326                            }
327                            alfa = alfa + radTol;
328                        }
329                    }
330                }
331            }
332        }
333
334        /**
```

```
335         * Update spheric positions (theta,phi,r) with camera data. Also
336   updates 2D
337         * positions and sizes.
338         *
339         *  @param  sizeTol  size  tolerance  in  camera  pixels.  Positive
340   integer
341         * @param radTol angular tolerance for the size search algorithm.
342   Smaller
343         * tolerance makes it more precise, but slower. Can't be 0
344         * @param angleOfView horizontal angle (rad) of view of the used
345   camera
346         *  @param  originTheta  horizontal  angle  (rad)  of  the  desired
347   origin in camera
348         *  spheric  coordinates,  where  (0,0,r)  is  coincident  with  the
349   leftmost pixel
350         * @param originPhi vertical angle (rad) of the desired origin in
351   camera
352         *  spheric  coordinates,  where  (0,0,r)  is  coincident  with  the
353   topmost pixel
354         * @param targetSize physical target size (radius) in the desired
355   units,
356         * used in r calculations
357         */
358        public  void  find3Dspherics(int  sizeTol,  double  radTol,  double
359   angleOfView,   double   originTheta,   double   originPhi,   double...
360   targetSize) {
361             if (targets.size() > positionSpheric.size()) {
362                 int tsize = targets.size();
363                 int psize = positionSpheric.size();
364                 for (int i = 0; i < tsize - psize; i++) {
365                     positionSpheric.add(new double[]{0, 0, 0});
366                 }
367             }
368             find2Dsizes(sizeTol, radTol);
369             for (int i = 0; i < targets.size(); i++) {
370                 positionSpheric.set(i, new double[]{(angleOfView / width)
371   *  positions2D.get(i)[0]  -  originTheta  -  PI  -  angleOfView  /  2,
372   (angleOfView  /  width)  *  positions2D.get(i)[1]  -  originPhi  -
373   angleOfView * height / (2 * width), targetSize[i] / (tan(size2D.get(i)
374   * angleOfView / width))});
375             }
376        }
377
378        /**
379         *
380         * Update spheric positions (theta,phi,r) with camera data. Also
381   updates 2D
382         * positions and sizes.
383         *
384         *  @param  sizeTol  size  tolerance  in  camera  pixels.  Positive
385   integer
386         * @param radTol angular tolerance for the size search algorithm.
387   Smaller
388         * tolerance makes it more precise, but slower. Can't be 0
389         * @param angleOfView horizontal angle (rad) of view of the used
390   camera
```

```
391         * @param originTheta horizontal angle (rad) of the desired
392   origin in camera
393         * spheric coordinates, where (0,0,r) is coincident with the
394   leftmost pixel
395         * @param originPhi vertical angle (rad) of the desired origin in
396   camera
397         * spheric coordinates, where (0,0,r) is coincident with the
398   topmost pixel
399         * @param xCamera distance between the desired origin and the
400   camera, along
401         * the x axis, positive if the camera is at the right of the
402   point, in
403         * targetSize units
404         * @param yCamera distance between the desired origin and the
405   camera, along
406         * the y axis, positive if the camera is under the point, in
407   targetSize
408         * units
409         * @param zCamera distance between the desired origin and the
410   camera, along
411         * the z axis, positive if the camera is closer to the user than
412   the point,
413         * in targetSize units
414         * @param pixelPerUnit relation between the desired pixel units
415   and the
416         *   targetSize   units.   Can   be   calculated   as   display
417   resolution/display size.
418         * @param targetSize physical target size (radius) in the desired
419   units,
420         * used in r calculations
421         */
422       public   void   find3D(int   sizeTol,   double   radTol,   double
423   angleOfView, double originTheta, double originPhi, double xCamera,
424   double  yCamera,  double  zCamera,  double  pixelPerUnit,  double...
425   targetSize) {
426           if (targets.size() > positions.size()) {
427               int tsize = targets.size();
428               int psize = positions.size();
429               for (int i = 0; i < tsize - psize; i++) {
430                   positions.add(new double[]{0, 0, 0});
431               }
432           }
433           find3Dspherics(sizeTol,  radTol,  angleOfView,  originTheta,
434   originPhi, targetSize);
435           for (int i = 0; i < targets.size(); i++) {
436               positions.set(i,  new  double[]{(positionSpheric.get(i)[2]
437   *  sin(positionSpheric.get(i)[0])  *  cos(positionSpheric.get(i)[1])  +
438   xCamera)     *     pixelPerUnit,     (positionSpheric.get(i)[2]     *
439   sin(positionSpheric.get(i)[1])    +    yCamera)    *    pixelPerUnit,
440   (positionSpheric.get(i)[2]    *    cos(positionSpheric.get(i)[0])    *
441   cos(positionSpheric.get(i)[1]) + zCamera) * pixelPerUnit});
442           }
443       }
444
445       /**
446        * Changes the target with the given index
447        *
```

```
448          * @param index target index
449          * @param target new value
450          */
451         public void setTarget(int index, float[] target) {
452             this.targets.set(index, target);
453         }
454 
455         /**
456          * Get the stored 3D coordinates of the objects in the image
457          *
458          * @return coordinates list
459          */
460         public List<double[]> getPositions() {
461             return positions;
462         }
463 
464         /**
465          * Get the stored spheric coordinates of the objects in the image
466          *
467          * @return coordinates list
468          */
469         public List<double[]> getSpheric() {
470             return positionSpheric;
471         }
472 
473         /**
474          * Get the stored sizes of the objects in the image
475          *
476          * @return sizes list
477          */
478         public List<Integer> getSizes2D() {
479             return size2D;
480         }
481 
482         /**
483          * Get the stored centers of the objects in the image.
484          *
485          * @return 2D positions array list
486          */
487         public List<int[]> getCenters2D() {
488             return positions2D;
489         }
490 
491         /**
492          * Get name of the webcam being used
493          *
494          * @return name of the webcam
495          */
496         public String getCamName() {
497             return webcam.getName();
498         }
499 
500         /**
501          * Get webcam image
502          *
503          * @return image from the webcam
```

```
504          */
505         public BufferedImage getImage() {
506             return webcam.getImage();
507         }
508 
509         /**
510          * Gets an image with a cross over the stored target centers. Use
511   after
512          * find2D if intended for current centers.
513          *
514          * @return image
515          */
516         public BufferedImage imageWithCenters() {
517             for (int[] position2D : positions2D) {
518                 drawCross(position2D[0],      position2D[1],      image,
519   Color.WHITE.getRGB());
520             }
521             return image;
522         }
523 
524         /**
525          * Returns a String containing all the 2D center coordinates and
526   sizes. Used
527          * for debuging
528          *
529          * @param coordinatesSeparator Separates x and y
530          * @param separator Separates coordinates of diferent targets
531          * @return String
532          */
533         public   String   position2DString(String   coordinatesSeparator,
534   String separator) {
535             StringBuilder sb = new StringBuilder();
536             for (int[] position2D : positions2D) {
537                 sb.append(String.valueOf(position2D[0]));
538                 sb.append(coordinatesSeparator);
539                 sb.append(String.valueOf(position2D[1]));
540                 sb.append(separator);
541             }
542             for (int size : size2D) {
543                 sb.append(size);
544                 sb.append(separator);
545             }
546             return sb.toString();
547         }
548 
549         /**
550          * Returns a String containing all the spheric coordinates. Used
551   for
552          * debuging
553          *
554          * @param coordinatesSeparator Separates x and y
555          * @param separator Separates coordinates of diferent targets
556          * @return String
557          */
558         public  String  sphericString(String  coordinatesSeparator,  String
559   separator) {
560             StringBuilder sb = new StringBuilder();
```

```
561             for (double[] position : positionSpheric) {
562                 sb.append(String.valueOf(position[0]));
563                 sb.append(coordinatesSeparator);
564                 sb.append(String.valueOf(position[1]));
565                 sb.append(coordinatesSeparator);
566                 sb.append(String.valueOf(position[2]));
567                 sb.append(separator);
568             }
569             return sb.toString();
570         }
571 
572         /**
573          * Returns a String containing all the cartesian coordinates.
574 Used for
575          * debuging
576          *
577          * @param coordinatesSeparator Separates x, y and z
578          * @param separator Separates coordinates of diferent targets
579          * @return String
580          */
581         public String cartesian3DString(String coordinatesSeparator,
582 String separator) {
583             StringBuilder sb = new StringBuilder();
584             for (double[] position : positions) {
585                 sb.append(String.valueOf(position[0]));
586                 sb.append(coordinatesSeparator);
587                 sb.append(String.valueOf(position[1]));
588                 sb.append(coordinatesSeparator);
589                 sb.append(String.valueOf(position[2]));
590                 sb.append(separator);
591             }
592             return sb.toString();
593         }
594 
595         /**
596          * Static method that draws a color cross at a given position of
597 a given
598          * image.
599          *
600          * @param xpos horizontal position of the cross center
601          * @param ypos vertical position of the cross center
602          * @param image original image
603          * @param color cross color
604          * @return image with the added cross
605          */
606         public static BufferedImage drawCross(int xpos, int ypos,
607 BufferedImage image, int color) {
608             for (int dist = 0; dist < 10; dist++) {
609                 if (xpos + dist < image.getWidth()) {
610                     image.setRGB(xpos + dist, ypos, color);
611                 }
612                 if (xpos - dist > 0) {
613                     image.setRGB(xpos - dist, ypos, color);
614                 }
615                 if (ypos + dist < image.getHeight()) {
616                     image.setRGB(xpos, ypos + dist, color);
```

```
617                 }
618                 if (ypos - dist > 0) {
619                     image.setRGB(xpos, ypos - dist, color);
620                 }
621             }
622             return image;
623         }
624 
625         /**
626          * Sets debug mode (paint analysed pixels and center crosses in
627 returned
628          * image)
629          *
630          * @param value new debug status
631          */
632         public void setDebug(boolean value) {
633             this.debug = value;
634         }
635 
636         /**
637          * Gets debug mode status (paint analysed pixels and center
638 crosses in
639          * returned image)
640          *
641          * @return true if in debug mode
642          */
643         public boolean getDebug() {
644             return debug;
645         }
646 
647         /**
648          * Set the color tolerance for the selected target
649          *
650          * @param tol new tolerance
651          * @param index target index
652          * @throws IllegalArgumentException if wrong index is inserted
653          */
654         public void setTol(int tol, int index) throws
655 IllegalArgumentException {
656             if (index > this.tol.size()) {
657                 if (index > targets.size()) {
658                     throw new IllegalArgumentException("Index out of
659 bounds");
660                 }
661                 this.tol.add(0);
662                 this.setTol(tol, index);
663             } else {
664                 this.tol.set(index, tol);
665             }
666         }
667     }
```

ANEXO C: CÓDIGO FUENTE DE LA APLICACIÓN DE CONFIGURACIÓN

```
1   package desktopARexample;
2
3   import com.github.sarxos.webcam.Webcam;
4   import desktopAR.ColorFinder;
5   import java.awt.FlowLayout;
6   import java.awt.image.BufferedImage;
7   import java.io.File;
8   import java.io.FileInputStream;
9   import java.io.FileNotFoundException;
10  import java.io.FileOutputStream;
11  import java.io.IOException;
12  import java.io.ObjectInputStream;
13  import java.io.ObjectOutputStream;
14  import static java.lang.Double.parseDouble;
15  import static java.lang.Integer.parseInt;
16  import static java.lang.Math.PI;
17  import javax.swing.BoxLayout;
18  import javax.swing.ImageIcon;
19  import javax.swing.JComboBox;
20  import javax.swing.JFrame;
21  import javax.swing.JLabel;
22  import javax.swing.JPanel;
23  import javax.swing.JSeparator;
24  import javax.swing.JSpinner;
25  import javax.swing.JToggleButton;
26  import javax.swing.SpinnerNumberModel;
27  import javax.swing.SwingConstants;
28
29  /**
30   * Example of usage of the ColorFinder class. Displays the obtained
31  image and
32   * the calculated 3D coordinates for the targets, both in cartesian
33  (custom
34   * origin) and spheric (origin in camera) forms. Allows for basic
35  calibration.
36   * Reads from up to 3 targets at the same time.
37   *
38   * @author Jaime Palomo Iranzo
39   */
40  public class BasicConfigAndViewProgram {
41
42      /**
43       * main method. Makes the class "runeable" and contains the
44  instructions for
45       * runtime.
46       *
47       * @param args
48       */
49      public static void main(String[] args) {
```

```
50
51            //Variable declaration and screen settings
52            JFrame frame = new JFrame("AR device calibration window");
53            JPanel camControls = new JPanel();
54            JPanel searchControls = new JPanel();
55            JPanel target1Controls = new JPanel();
56            JPanel target2Controls = new JPanel();
57            JPanel target3Controls = new JPanel();
58            JPanel targetControls = new JPanel();
59            JPanel origin1Controls = new JPanel();
60            JPanel origin2Controls = new JPanel();
61            JPanel origin3Controls = new JPanel();
62            JPanel originControls = new JPanel();
63            JPanel targetOriginControls = new JPanel();
64
65            JPanel display = new JPanel();
66
67            JLabel imageLabel = new JLabel();
68            JLabel coordinateLabel = new JLabel();
69            JLabel timeLabel = new JLabel();
70            JLabel fileException=new JLabel();
71
72            SpinnerNumberModel color1Model = new SpinnerNumberModel(100,
73  0, 255, 1);
74            SpinnerNumberModel color2Model = new SpinnerNumberModel(100,
75  0, 255, 1);
76            SpinnerNumberModel color3Model = new SpinnerNumberModel(100,
77  0, 255, 1);
78
79            JSpinner color1 = new JSpinner(color1Model);
80            JSpinner color2 = new JSpinner(color2Model);
81            JSpinner color3 = new JSpinner(color3Model);
82
83            SpinnerNumberModel size1Model = new SpinnerNumberModel(15, 0,
84  100, 0.1);
85            SpinnerNumberModel size2Model = new SpinnerNumberModel(15, 0,
86  100, 0.1);
87            SpinnerNumberModel size3Model = new SpinnerNumberModel(15, 0,
88  100, 0.1);
89
90            JSpinner size1 = new JSpinner(size1Model);
91            JSpinner size2 = new JSpinner(size2Model);
92            JSpinner size3 = new JSpinner(size3Model);
93
94            SpinnerNumberModel xModel = new SpinnerNumberModel(0, -1000,
95  1000, 0.1);
96            SpinnerNumberModel yModel = new SpinnerNumberModel(0, -1000,
97  1000, 0.1);
98            SpinnerNumberModel zModel = new SpinnerNumberModel(0, -1000,
99  1000, 0.1);
100
101           JSpinner x = new JSpinner(xModel);
102           JSpinner y = new JSpinner(yModel);
103           JSpinner z = new JSpinner(zModel);
104
```

```
105             SpinnerNumberModel        screenSizeModel        =        new
106     SpinnerNumberModel(0, 0, 1000, 0.1);
107             SpinnerNumberModel      screenResolutionModel      =      new
108     SpinnerNumberModel(0, 0, 10000, 1);
109
110             JSpinner screenSize = new JSpinner(screenSizeModel);
111             JSpinner           screenResolution           =           new
112     JSpinner(screenResolutionModel);
113
114             SpinnerNumberModel sizeModel = new SpinnerNumberModel(1, 0,
115     100, 1);
116             SpinnerNumberModel sizeAngleModel = new SpinnerNumberModel(PI
117     / 2, 0.01, 2 * PI, 0.01*PI);
118
119             JSpinner size = new JSpinner(sizeModel);
120             JSpinner sizeAngle = new JSpinner(sizeAngleModel);
121
122             SpinnerNumberModel thetaModel = new SpinnerNumberModel(0, -
123     PI, PI, 0.01 * PI);
124             JSpinner theta = new JSpinner(thetaModel);
125
126             SpinnerNumberModel phiModel = new SpinnerNumberModel(0, -PI,
127     PI, 0.01 * PI);
128             JSpinner phi = new JSpinner(phiModel);
129
130             SpinnerNumberModel angleModel = new SpinnerNumberModel(PI /
131     2, 0, PI, 0.01 * PI);
132             JSpinner angle = new JSpinner(angleModel);
133
134             String[]           webcamNameArray           =           new
135     String[Webcam.getWebcams().size()];
136
137             for (int i = 0; i < Webcam.getWebcams().size(); i++) {
138                 webcamNameArray[i]                                      =
139     Webcam.getWebcams().get(i).getName();
140             }
141             JComboBox selectCam = new JComboBox(webcamNameArray);
142             selectCam.setSelectedIndex(0);
143
144             String[] targetColorArray = {"Red", "Green", "Blue", "Cyan",
145     "Magenta", "Yellow"};
146             float[][]     targettedColorArray     =     {ColorFinder.RED,
147     ColorFinder.GREEN,          ColorFinder.BLUE,          ColorFinder.CYAN,
148     ColorFinder.MAGENTA, ColorFinder.YELLOW};
149             JComboBox target1 = new JComboBox(targetColorArray);
150             JComboBox target2 = new JComboBox(targetColorArray);
151             JComboBox target3 = new JComboBox(targetColorArray);
152
153             JToggleButton debug = new JToggleButton("Show image");
154
155             JComboBox  selectRes  =  new  JComboBox(new  String[]{"high",
156     "medium", "low"});
157             selectRes.setSelectedIndex(2);
158             int oldRes = 2;
159             int resX = 176;
160             int resY = 144;
161
```

```
162             JToggleButton save = new JToggleButton("save");
163             JToggleButton load = new JToggleButton("load");
164 
165             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
166             frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
167             frame.getContentPane().setLayout(new
168   BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
169             targetControls.setLayout(new        BoxLayout(targetControls,
170   BoxLayout.Y_AXIS));
171             originControls.setLayout(new        BoxLayout(originControls,
172   BoxLayout.Y_AXIS));
173             targetOriginControls.setLayout(new
174   BoxLayout(targetOriginControls, BoxLayout.X_AXIS));
175             camControls.setLayout(new FlowLayout());
176             display.setLayout(new FlowLayout());
177 
178             camControls.add(new JLabel("Camera:"));
179             camControls.add(selectCam);
180             camControls.add(new JLabel("Camera AOV:"));
181             camControls.add(angle);
182             camControls.add(new JLabel("Camera resolution:"));
183             camControls.add(selectRes);
184             camControls.add(debug);
185 
186             searchControls.add(new JLabel("Calculated size step:"));
187             searchControls.add(size);
188             searchControls.add(new    JLabel("Calculated    size    angle
189   step:"));
190             searchControls.add(sizeAngle);
191             searchControls.add(save);
192             searchControls.add(load);
193             searchControls.add(fileException);
194 
195             target1Controls.add(target1);
196             target1Controls.add(new JLabel(" tolerance:"));
197             target1Controls.add(color1);
198             target1Controls.add(new JLabel(" size:"));
199             target1Controls.add(size1);
200 
201             target2Controls.add(target2);
202             target2Controls.add(new JLabel(" tolerance:"));
203             target2Controls.add(color2);
204             target2Controls.add(new JLabel(" size:"));
205             target2Controls.add(size2);
206 
207             target3Controls.add(target3);
208             target3Controls.add(new JLabel(" tolerance:"));
209             target3Controls.add(color3);
210             target3Controls.add(new JLabel(" size:"));
211             target3Controls.add(size3);
212 
213             targetControls.add(target1Controls);
214             targetControls.add(target2Controls);
215             targetControls.add(target3Controls);
216 
217             origin1Controls.add(new JLabel("Camera horizontal angle:"));
```

```
218             origin1Controls.add(theta);
219             origin1Controls.add(new JLabel("Camera vertical angle:"));
220             origin1Controls.add(phi);
221 
222             origin2Controls.add(new JLabel("Camera coordinates: "));
223             origin2Controls.add(x);
224             origin2Controls.add(new JLabel(", "));
225             origin2Controls.add(y);
226             origin2Controls.add(new JLabel(", "));
227             origin2Controls.add(z);
228 
229             origin3Controls.add(new JLabel("Screen size: "));
230             origin3Controls.add(screenSize);
231             origin3Controls.add(new JLabel("Screen resolution: "));
232             origin3Controls.add(screenResolution);
233 
234             originControls.add(origin1Controls);
235             originControls.add(origin2Controls);
236             originControls.add(origin3Controls);
237 
238             targetOriginControls.add(targetControls);
239             targetOriginControls.add(new
240   JSeparator(SwingConstants.VERTICAL));
241             targetOriginControls.add(originControls);
242 
243             frame.getContentPane().add(camControls);
244             frame.getContentPane().add(searchControls);
245             frame.getContentPane().add(targetOriginControls);
246 
247             display.add(imageLabel);
248 
249             display.add(coordinateLabel);
250 
251             frame.getContentPane().add(display);
252 
253             frame.getContentPane().add(timeLabel);
254 
255             frame.setVisible(true);
256 
257             ColorFinder              finder              =              new
258   ColorFinder(Webcam.getWebcamByName(selectCam.getSelectedItem().toStrin
259   g()), resX, resY, 50, targettedColorArray[target1.getSelectedIndex()],
260   targettedColorArray[target2.getSelectedIndex()],
261   targettedColorArray[target3.getSelectedIndex()]);
262             finder.camStart();
263 
264             //Main loop. Runs until program exit.
265             while (true) {
266                 long startTime = System.nanoTime();
267 
268                 //Save settings
269                 if (save.isSelected()) {
270                     try {
271                         FileOutputStream  f  =  new  FileOutputStream(new
272   File("config.dat"));
273                         ObjectOutputStream o = new ObjectOutputStream(f);
274 
```

```
275                         o.writeObject(color1.getValue());
276                         o.writeObject(color2.getValue());
277                         o.writeObject(color3.getValue());
278 
279                         o.writeObject(size1.getValue());
280                         o.writeObject(size2.getValue());
281                         o.writeObject(size3.getValue());
282 
283                         o.writeObject(x.getValue());
284                         o.writeObject(y.getValue());
285                         o.writeObject(z.getValue());
286 
287                         o.writeObject(screenSize.getValue());
288                         o.writeObject(screenResolution.getValue());
289 
290                         o.writeObject(size.getValue());
291                         o.writeObject(sizeAngle.getValue());
292 
293                         o.writeObject(theta.getValue());
294                         o.writeObject(phi.getValue());
295                         o.writeObject(angle.getValue());
296 
297                         o.writeObject((int)target1.getSelectedIndex());
298                         o.writeObject((int)target2.getSelectedIndex());
299                         o.writeObject((int)target3.getSelectedIndex());
300 
301                         o.close();
302                         f.close();
303                     } catch (FileNotFoundException e) {
304                         fileException.setText("File not found");
305                     } catch (IOException e) {
306                         fileException.setText("Error         initializing
307 stream");
308                     }
309                     save.setSelected(false);
310                 }
311                 //Load settings
312                 if (load.isSelected()) {
313                     try {
314                         FileInputStream  fi  =  new  FileInputStream(new
315 File("config.dat"));
316                         ObjectInputStream oi = new ObjectInputStream(fi);
317 
318                         color1.setValue(oi.readObject());
319                         color2.setValue(oi.readObject());
320                         color3.setValue(oi.readObject());
321 
322                         size1.setValue(oi.readObject());
323                         size2.setValue(oi.readObject());
324                         size3.setValue(oi.readObject());
325 
326                         x.setValue(oi.readObject());
327                         y.setValue(oi.readObject());
328                         z.setValue(oi.readObject());
329 
330                         screenSize.setValue(oi.readObject());
```

```
                        screenResolution.setValue(oi.readObject());

                        size.setValue(oi.readObject());
                        sizeAngle.setValue(oi.readObject());

                        theta.setValue(oi.readObject());
                        phi.setValue(oi.readObject());
                        angle.setValue(oi.readObject());

                        target1.setSelectedIndex((int)oi.readObject());
                        target2.setSelectedIndex((int)oi.readObject());
                        target3.setSelectedIndex((int)oi.readObject());

                        oi.close();
                        fi.close();
                        fileException.setText(null);
                    } catch (FileNotFoundException e) {
                        fileException.setText("File not found");
                    } catch (IOException e) {
                        fileException.setText("Error         initializing
stream");
                    } catch (ClassNotFoundException ex) {
                        fileException.setText("Error: wrong or corrupted
config file");
                    }
                    load.setSelected(false);
                }
                //Read settings
                finder.setTarget(0,
targettedColorArray[target1.getSelectedIndex()]);
                finder.setTarget(1,
targettedColorArray[target2.getSelectedIndex()]);
                finder.setTarget(2,
targettedColorArray[target3.getSelectedIndex()]);

                finder.setDebug(debug.isSelected());

                if (oldRes != selectRes.getSelectedIndex()) {
                    oldRes = selectRes.getSelectedIndex();
                    switch (oldRes) {
                        case 0:
                            resX = 640;
                            resY = 480;
                            break;
                        case 1:
                            resX = 320;
                            resY = 240;
                            break;
                        default:
                            resX = 176;
                            resY = 144;
                            break;
                    }

finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
String()), resX, resY);
                }
```

```
388                 if
389   (!finder.getCamName().equals(selectCam.getSelectedItem().toString()))
390   {
391
392   finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
393   String()));
394                 }
395                 finder.setTol(parseInt(color1.getValue().toString()), 0);
396                 finder.setTol(parseInt(color2.getValue().toString()), 1);
397                 finder.setTol(parseInt(color3.getValue().toString()), 2);
398
399                 long setTime = System.nanoTime();
400
401                 //read camera
402                 finder.find3D((int)
403   parseDouble(size.getValue().toString()),
404   parseDouble(sizeAngle.getValue().toString()),
405   parseDouble(angle.getValue().toString()),
406   parseDouble(theta.getValue().toString()),
407   parseDouble(phi.getValue().toString()),
408   parseDouble(x.getValue().toString()),
409   parseDouble(y.getValue().toString()),
410   parseDouble(z.getValue().toString()),
411   parseDouble(screenResolution.getValue().toString())                    /
412   parseDouble(screenSize.getValue().toString()),
413   parseDouble(size1.getValue().toString()),
414   parseDouble(size2.getValue().toString()),
415   parseDouble(size3.getValue().toString()));
416
417                 long searchTime = System.nanoTime();
418
419                 //show results
420                 if (debug.isSelected()) {
421                     imageLabel.setIcon(new
422   ImageIcon(finder.imageWithCenters().getScaledInstance(640,         480,
423   BufferedImage.SCALE_FAST)));
424                 } else {
425                     imageLabel.setIcon(null);
426                 }
427                 coordinateLabel.setText("<html>   Cartesian   coordinates
428   (screen pixels):<br>" + finder.cartesian3DString("<br>", "<br><br>") +
429   "Spheric      coordinates      (cm      to      camera):<br>"      +
430   finder.sphericString("<br>", "<br><br>") + "</html>");
431
432                 long endTime = System.nanoTime();
433                 timeLabel.setText(String.format("set:   %s   search:   %s
434   display:    %s    total:    %s",    toString(setTime    -    startTime),
435   toString(searchTime   -   setTime),   toString(endTime   -   searchTime),
436   toString(endTime - startTime)));
437             }
438         }
439
440         /**
441          * Generates a string containing time data. Used for debugging
442   and test
443          * purposes.
```

```
444          *
445          * @param nanoSecs
446          * @return
447          */
448         private static String toString(long nanoSecs) {
449             int minutes = (int) (nanoSecs / 60000000000.0);
450             int  seconds = (int) (nanoSecs / 1000000000.0)  - (minutes *
451 60);
452             int millisecs = (int) (((nanoSecs / 1000000000.0) - (seconds
453 + minutes * 60)) * 1000);
454 
455             if (minutes == 0 && seconds == 0) {
456                 return millisecs + "ms";
457             } else if (minutes == 0 && millisecs == 0) {
458                 return seconds + "s";
459             } else if (seconds == 0 && millisecs == 0) {
460                 return minutes + "min";
461             } else if (minutes == 0) {
462                 return seconds + "s " + millisecs + "ms";
463             } else if (seconds == 0) {
464                 return minutes + "min " + millisecs + "ms";
465             } else if (millisecs == 0) {
466                 return minutes + "min " + seconds + "s";
467             }
468 
469             return minutes + "min " + seconds + "s " + millisecs + "ms";
470         }
471 
472     }
```

ANEXO D: CÓDIGO FUENTE DE LA APLICACIÓN DE RATÓN AR

```
1   package desktopARexample;
2
3   import com.github.sarxos.webcam.Webcam;
4   import desktopAR.ColorFinder;
5   import java.awt.AWTException;
6   import java.awt.Dimension;
7   import java.awt.FlowLayout;
8   import java.awt.Robot;
9   import java.awt.Toolkit;
10  import java.awt.event.InputEvent;
11  import java.awt.image.BufferedImage;
12  import java.io.File;
13  import java.io.FileInputStream;
14  import java.io.FileNotFoundException;
15  import java.io.IOException;
16  import java.io.ObjectInputStream;
17  import static java.lang.Double.parseDouble;
18  import static java.lang.Integer.parseInt;
19  import static java.lang.Math.PI;
20  import javax.swing.BoxLayout;
21  import javax.swing.ImageIcon;
22  import javax.swing.JComboBox;
23  import javax.swing.JFrame;
24  import javax.swing.JLabel;
25  import javax.swing.JOptionPane;
26  import javax.swing.JPanel;
27  import javax.swing.JSpinner;
28  import javax.swing.JToggleButton;
29  import javax.swing.SpinnerNumberModel;
30
31  /**
32   * Example of usage of the ColorFinder class. Control the mouse using
33  two
34   * target colors, with x and y of mouse corresponding to those of the
35  second
36   * target, main click corresponding to moving the target closer to
37  each other
38   * and secondary click corresponding to moving it away from each
39  other.
40   *
41   * @author Jaime Palomo Iranzo
42   */
43  public class ARMouse {
44
45      /**
46       * main method. Makes the class "runeable" and contains the
47  instructions for
48       * runtime.
49       *
```

```
50          * @param args
51          */
52         public static void main(String[] args) {
53 
54             try {
55                 //Variable declaration and screen settings
56                 Robot bot = new Robot();
57 
58                 double mouseX;
59                 double mouseY;
60                 boolean mouse1press = false;
61                 boolean mouse2press = false;
62 
63                 JFrame frame = new JFrame("AR device calibration
64 window");
65                 JPanel camControls = new JPanel();
66                 JPanel searchControls = new JPanel();
67                 JPanel target1Controls = new JPanel();
68                 JPanel target2Controls = new JPanel();
69                 JPanel targetControls = new JPanel();
70                 JPanel originControls = new JPanel();
71                 JPanel targetOriginControls = new JPanel();
72 
73                 JPanel display = new JPanel();
74 
75                 JLabel imageLabel = new JLabel();
76                 JLabel fileException = new JLabel();
77 
78                 SpinnerNumberModel color1Model = new
79 SpinnerNumberModel(100, 0, 255, 1);
80                 SpinnerNumberModel color2Model = new
81 SpinnerNumberModel(100, 0, 255, 1);
82 
83                 JSpinner color1 = new JSpinner(color1Model);
84                 JSpinner color2 = new JSpinner(color2Model);
85 
86                 SpinnerNumberModel screenMarginModel = new
87 SpinnerNumberModel(50, 0, 1000, 0.1);
88 
89                 JSpinner screenMargin = new JSpinner(screenMarginModel);
90 
91                 SpinnerNumberModel sizeModel = new SpinnerNumberModel(1,
92 0, 100, 1);
93                 SpinnerNumberModel sizeAngleModel = new
94 SpinnerNumberModel(PI / 2, 0.01, 2 * PI, 0.01 * PI);
95 
96                 JSpinner size = new JSpinner(sizeModel);
97                 JSpinner sizeAngle = new JSpinner(sizeAngleModel);
98 
99                 String[] webcamNameArray = new
100 String[Webcam.getWebcams().size()];
101 
102                 for (int i = 0; i < Webcam.getWebcams().size(); i++) {
103                     webcamNameArray[i] =
104 Webcam.getWebcams().get(i).getName();
105                 }
```

```
106                 JComboBox selectCam = new JComboBox(webcamNameArray);
107                 selectCam.setSelectedIndex(0);
108 
109                 String[]  targetColorArray  =  {"Red",  "Green",  "Blue",
110 "Cyan", "Magenta", "Yellow"};
111                 float[][]    targettedColorArray    =    {ColorFinder.RED,
112 ColorFinder.GREEN,          ColorFinder.BLUE,          ColorFinder.CYAN,
113 ColorFinder.MAGENTA, ColorFinder.YELLOW};
114                 JComboBox target1 = new JComboBox(targetColorArray);
115                 JComboBox target2 = new JComboBox(targetColorArray);
116 
117                 JToggleButton debug = new JToggleButton("Show image");
118 
119                 JComboBox selectRes = new JComboBox(new String[]{"high",
120 "medium", "low"});
121                 selectRes.setSelectedIndex(2);
122                 int oldRes = 2;
123                 int resX = 176;
124                 int resY = 144;
125 
126                 JToggleButton start = new JToggleButton("start AR");
127                 JToggleButton load = new JToggleButton("load config");
128 
129                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
130                 frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
131                 frame.getContentPane().setLayout(new
132 BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
133                 targetControls.setLayout(new    BoxLayout(targetControls,
134 BoxLayout.Y_AXIS));
135                 originControls.setLayout(new FlowLayout());
136                 targetOriginControls.setLayout(new FlowLayout());
137                 camControls.setLayout(new FlowLayout());
138                 display.setLayout(new FlowLayout());
139 
140                 camControls.add(new JLabel("Camera:"));
141                 camControls.add(selectCam);
142                 camControls.add(new JLabel("Camera resolution:"));
143                 camControls.add(selectRes);
144                 camControls.add(debug);
145 
146                 searchControls.add(new JLabel("Calculated size step:"));
147                 searchControls.add(size);
148                 searchControls.add(new   JLabel("Calculated   size   angle
149 step:"));
150                 searchControls.add(sizeAngle);
151 
152                 searchControls.add(load);
153                 searchControls.add(fileException);
154 
155                 targetOriginControls.add(target1);
156                 targetOriginControls.add(new JLabel(" tolerance:"));
157                 targetOriginControls.add(color1);
158 
159                 targetOriginControls.add(target2);
160                 targetOriginControls.add(new JLabel(" tolerance:"));
161                 targetOriginControls.add(color2);
162 
```

```
163                 targetControls.add(target1Controls);
164                 targetControls.add(target2Controls);
165 
166                 targetOriginControls.add(new JLabel("Screen margin: "));
167                 targetOriginControls.add(screenMargin);
168 
169                 frame.getContentPane().add(camControls);
170                 frame.getContentPane().add(searchControls);
171                 frame.getContentPane().add(targetOriginControls);
172 
173                 display.add(imageLabel);
174 
175                 frame.getContentPane().add(display);
176 
177                 frame.getContentPane().add(start);
178 
179                 frame.setVisible(true);
180 
181                 ColorFinder            finder            =            new
182   ColorFinder(Webcam.getWebcamByName(selectCam.getSelectedItem().toStrin
183   g()), resX, resY, 50, targettedColorArray[target1.getSelectedIndex()],
184   targettedColorArray[target2.getSelectedIndex()]);
185                 finder.camStart();
186 
187                 //load config at start
188                 try {
189                     FileInputStream   fi   =   new   FileInputStream(new
190   File("config.dat"));
191                     ObjectInputStream oi = new ObjectInputStream(fi);
192 
193                     color1.setValue(oi.readObject());
194                     color2.setValue(oi.readObject());
195                     oi.readObject();
196 
197                     oi.readObject();
198                     oi.readObject();
199                     oi.readObject();
200 
201                     oi.readObject();
202                     oi.readObject();
203                     oi.readObject();
204 
205                     oi.readObject();
206                     oi.readObject();
207 
208                     size.setValue(oi.readObject());
209                     sizeAngle.setValue(oi.readObject());
210 
211                     oi.readObject();
212                     oi.readObject();
213                     oi.readObject();
214 
215                     target1.setSelectedIndex((int) oi.readObject());
216                     target2.setSelectedIndex((int) oi.readObject());
217                     oi.readObject();
218 
```

```
                    oi.close();
                    fi.close();
                    fileException.setText(null);
                } catch (FileNotFoundException e) {
                    fileException.setText("File not found");
                } catch (IOException e) {
                    fileException.setText("Error initializing stream");
                } catch (ClassNotFoundException ex) {
                    fileException.setText("Error:   wrong   or   corrupted
config file");
                }

                //Main loop. Runs until program exit.
                while (true) {

                    //Load settings
                    if (load.isSelected()) {
                        try {
                            FileInputStream fi = new FileInputStream(new
File("config.dat"));
                            ObjectInputStream       oi       =       new
ObjectInputStream(fi);

                            color1.setValue(oi.readObject());
                            color2.setValue(oi.readObject());
                            oi.readObject();

                            oi.readObject();
                            oi.readObject();
                            oi.readObject();

                            oi.readObject();
                            oi.readObject();
                            oi.readObject();

                            oi.readObject();
                            oi.readObject();

                            size.setValue(oi.readObject());
                            sizeAngle.setValue(oi.readObject());

                            oi.readObject();
                            oi.readObject();
                            oi.readObject();

                            target1.setSelectedIndex((int)
oi.readObject());
                            target2.setSelectedIndex((int)
oi.readObject());
                            oi.readObject();

                            oi.close();
                            fi.close();
                            fileException.setText(null);
                        } catch (FileNotFoundException e) {
                            fileException.setText("File not found");
                        } catch (IOException e) {
```

```
                            fileException.setText("Error     initializing
stream");
                        } catch (ClassNotFoundException ex) {
                            fileException.setText("Error:     wrong     or
corrupted config file");
                        }
                        load.setSelected(false);
                    }
                    //Read settings
                    finder.setTarget(0,
targettedColorArray[target1.getSelectedIndex()]);
                    finder.setTarget(1,
targettedColorArray[target2.getSelectedIndex()]);

                    finder.setDebug(debug.isSelected());

                    if (oldRes != selectRes.getSelectedIndex()) {
                        oldRes = selectRes.getSelectedIndex();
                        switch (oldRes) {
                            case 0:
                                resX = 640;
                                resY = 480;
                                break;
                            case 1:
                                resX = 320;
                                resY = 240;
                                break;
                            default:
                                resX = 176;
                                resY = 144;
                                break;
                        }

finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
String()), resX, resY);
                    }
                    if
(!finder.getCamName().equals(selectCam.getSelectedItem().toString()))
{

finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
String()));
                    }
                    finder.setTol(parseInt(color1.getValue().toString()),
0);
                    finder.setTol(parseInt(color2.getValue().toString()),
1);

                    //read camera
                    Dimension                screenSize                =
Toolkit.getDefaultToolkit().getScreenSize();
                    double                    margin                    =
parseDouble(screenMargin.getValue().toString());
                    double width = screenSize.getWidth();
                    double height = screenSize.getHeight();

```

```
                    finder.find2Dsizes((int)
parseDouble(size.getValue().toString()),
parseDouble(sizeAngle.getValue().toString()));

                    double coord11 = finder.getCenters2D().get(0)[0];
                    double coord12 = finder.getCenters2D().get(0)[1];
                    double coord21 = finder.getCenters2D().get(1)[0];
                    double coord22 = finder.getCenters2D().get(1)[1];

                    mouseX = width + margin - (coord21 - margin) * width
/ (resX - 2 * margin);
                    mouseY = (coord22 - margin) * height / (resY - 2 *
margin) - margin;
                    double dist = (coord11 - coord21) * (coord11 -
coord21) + (coord12 - coord22) * (coord12 - coord22);
                    double sqrsize = finder.getSizes2D().get(0) *
finder.getSizes2D().get(0);

                    if (start.isSelected()) {
                        bot.mouseMove((int) mouseX, (int) mouseY);
                        if (sqrsize * 4 > dist && mouse1press == false) {
                            bot.mousePress(InputEvent.BUTTON1_DOWN_MASK);
                            mouse1press = true;
                        }
                        if (sqrsize * 4 < dist && mouse1press == true) {

bot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
                            mouse1press = false;
                        }
                        if (sqrsize * 36 < dist && mouse2press == false)
{
                            bot.mousePress(InputEvent.BUTTON3_DOWN_MASK);
                            mouse2press = true;
                        }
                        if (sqrsize * 36 > dist && mouse2press == true) {

bot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK);
                            mouse2press = false;
                        }
                    }
                    //show results
                    if (debug.isSelected()) {
                        imageLabel.setIcon(new
ImageIcon(finder.imageWithCenters().getScaledInstance(640,        480,
BufferedImage.SCALE_FAST)));
                    } else {
                        imageLabel.setIcon(null);
                    }
                }
            } catch (AWTException ex) {
                JOptionPane.showMessageDialog(null,  "Error:  could  not
open the application");
            }
        }
    }
```

ANEXO E: CÓDIGO FUENTE DE LA APLICACIÓN DE VISIÓN DE IMÁGENES

```
1   package desktopARexample;
2
3   import com.github.sarxos.webcam.Webcam;
4   import desktopAR.ColorFinder;
5   import java.awt.Color;
6   import java.awt.FlowLayout;
7   import java.awt.Point;
8   import java.awt.image.BufferedImage;
9   import java.io.File;
10  import java.io.FileInputStream;
11  import java.io.FileNotFoundException;
12  import java.io.FileOutputStream;
13  import java.io.IOException;
14  import java.io.ObjectInputStream;
15  import java.io.ObjectOutputStream;
16  import static java.lang.Double.parseDouble;
17  import static java.lang.Integer.parseInt;
18  import static java.lang.Math.PI;
19  import static java.lang.Math.atan;
20  import javax.imageio.ImageIO;
21  import javax.swing.BoxLayout;
22  import javax.swing.ImageIcon;
23  import javax.swing.JComboBox;
24  import javax.swing.JFrame;
25  import javax.swing.JLabel;
26  import javax.swing.JOptionPane;
27  import javax.swing.JPanel;
28  import javax.swing.JSeparator;
29  import javax.swing.JSpinner;
30  import javax.swing.JToggleButton;
31  import javax.swing.SpinnerNumberModel;
32  import javax.swing.SwingConstants;
33
34  /**
35   * Example of usage of the ColorFinder class. Displays an image while
36  tracking
37   *  the user's head, attempting to correct the perspective by
38  deforming the
39   * image. Includes a separate config window as well. Only the third
40  target is
41   * used. You can close the config window after it's set.
42   *
43   * @author Jaime Palomo Iranzo
44   */
45  public class ARImage {
46
47      /**
48       * main method. Makes the class "runeable" and contains the
49  instructions for
```

```
50          * runtime.
51          *
52          * @param args
53          */
54         public static void main(String[] args) {
55
56             //Variable declaration and screen settings
57             JFrame frame = new JFrame("AR device calibration window");
58             JFrame imageFrame = new JFrame("AR display");
59             JPanel camControls = new JPanel();
60             JPanel searchControls = new JPanel();
61             JPanel target1Controls = new JPanel();
62             JPanel target2Controls = new JPanel();
63             JPanel target3Controls = new JPanel();
64             JPanel targetControls = new JPanel();
65             JPanel origin1Controls = new JPanel();
66             JPanel origin2Controls = new JPanel();
67             JPanel origin3Controls = new JPanel();
68             JPanel originControls = new JPanel();
69             JPanel targetOriginControls = new JPanel();
70             JPanel imageControls = new JPanel();
71
72             JPanel display = new JPanel();
73
74             BufferedImage ar;
75             BufferedImage img = null;
76             try {
77                 img = ImageIO.read(new File("img.jpg"));
78             } catch (IOException e) {
79                 JOptionPane.showMessageDialog(null,  "Error:  could  not
80   find img.jpg in program folder");
81                 java.lang.System.exit(0);
82             }
83
84             JLabel imageLabel = new JLabel();
85             JLabel arLabel = new JLabel();
86             JLabel coordinateLabel = new JLabel();
87             JLabel timeLabel = new JLabel();
88             JLabel fileException = new JLabel();
89
90             SpinnerNumberModel       angularWidthModel       =       new
91   SpinnerNumberModel(3141, 0, 1000 * PI, 1);
92             SpinnerNumberModel       angularHeightModel       =       new
93   SpinnerNumberModel(3141, 0, 1000 * PI, 1);
94             SpinnerNumberModel  arScaleModel  =  new  SpinnerNumberModel(4,
95   1, 10, 1);
96
97             JSpinner arScale = new JSpinner(arScaleModel);
98             JSpinner angularWidth = new JSpinner(angularWidthModel);
99             JSpinner angularHeight = new JSpinner(angularHeightModel);
100
101            SpinnerNumberModel color1Model = new SpinnerNumberModel(100,
102  0, 255, 1);
103            SpinnerNumberModel color2Model = new SpinnerNumberModel(100,
104  0, 255, 1);
```

```
105             SpinnerNumberModel color3Model = new SpinnerNumberModel(100,
106     0, 255, 1);
107
108             JSpinner color1 = new JSpinner(color1Model);
109             JSpinner color2 = new JSpinner(color2Model);
110             JSpinner color3 = new JSpinner(color3Model);
111
112             SpinnerNumberModel size1Model = new SpinnerNumberModel(15, 0,
113     100, 0.1);
114             SpinnerNumberModel size2Model = new SpinnerNumberModel(15, 0,
115     100, 0.1);
116             SpinnerNumberModel size3Model = new SpinnerNumberModel(15, 0,
117     100, 0.1);
118
119             JSpinner size1 = new JSpinner(size1Model);
120             JSpinner size2 = new JSpinner(size2Model);
121             JSpinner size3 = new JSpinner(size3Model);
122
123             SpinnerNumberModel xModel = new SpinnerNumberModel(0, -1000,
124     1000, 0.1);
125             SpinnerNumberModel yModel = new SpinnerNumberModel(0, -1000,
126     1000, 0.1);
127             SpinnerNumberModel zModel = new SpinnerNumberModel(0, -1000,
128     1000, 0.1);
129
130             JSpinner x = new JSpinner(xModel);
131             JSpinner y = new JSpinner(yModel);
132             JSpinner z = new JSpinner(zModel);
133
134             SpinnerNumberModel        screenSizeModel        =        new
135     SpinnerNumberModel(0, 0, 1000, 0.1);
136             SpinnerNumberModel      screenResolutionModel      =      new
137     SpinnerNumberModel(0, 0, 10000, 1);
138
139             JSpinner screenSize = new JSpinner(screenSizeModel);
140             JSpinner           screenResolution           =           new
141     JSpinner(screenResolutionModel);
142
143             SpinnerNumberModel sizeModel = new SpinnerNumberModel(1, 0,
144     100, 1);
145             SpinnerNumberModel sizeAngleModel = new SpinnerNumberModel(PI
146     / 2, 0.01, 2 * PI, 0.01 * PI);
147
148             JSpinner size = new JSpinner(sizeModel);
149             JSpinner sizeAngle = new JSpinner(sizeAngleModel);
150
151             SpinnerNumberModel thetaModel = new SpinnerNumberModel(0, -
152     PI, PI, 0.01 * PI);
153             JSpinner theta = new JSpinner(thetaModel);
154
155             SpinnerNumberModel phiModel = new SpinnerNumberModel(0, -PI,
156     PI, 0.01 * PI);
157             JSpinner phi = new JSpinner(phiModel);
158
159             SpinnerNumberModel angleModel = new SpinnerNumberModel(PI /
160     2, 0, PI, 0.01 * PI);
161             JSpinner angle = new JSpinner(angleModel);
```

```
162
163             String[]           webcamNameArray           =           new
164     String[Webcam.getWebcams().size()];
165
166             for (int i = 0; i < Webcam.getWebcams().size(); i++) {
167                 webcamNameArray[i]                                      =
168     Webcam.getWebcams().get(i).getName();
169             }
170             JComboBox selectCam = new JComboBox(webcamNameArray);
171             selectCam.setSelectedIndex(0);
172
173             String[] targetColorArray = {"Red", "Green", "Blue", "Cyan",
174     "Magenta", "Yellow"};
175             float[][]     targettedColorArray     =     {ColorFinder.RED,
176     ColorFinder.GREEN,          ColorFinder.BLUE,          ColorFinder.CYAN,
177     ColorFinder.MAGENTA, ColorFinder.YELLOW};
178             JComboBox target1 = new JComboBox(targetColorArray);
179             JComboBox target2 = new JComboBox(targetColorArray);
180             JComboBox target3 = new JComboBox(targetColorArray);
181
182             JToggleButton debug = new JToggleButton("Show webcam image");
183
184             JComboBox  selectRes  =  new  JComboBox(new  String[]{"high",
185     "medium", "low"});
186             selectRes.setSelectedIndex(2);
187             int oldRes = 2;
188             int resX = 176;
189             int resY = 144;
190
191             JToggleButton save = new JToggleButton("save");
192             JToggleButton load = new JToggleButton("load");
193
194             imageFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
195             imageFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
196             frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
197             frame.getContentPane().setLayout(new
198     BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
199             imageFrame.getContentPane().setLayout(new
200     BoxLayout(imageFrame.getContentPane(), BoxLayout.Y_AXIS));
201             targetControls.setLayout(new        BoxLayout(targetControls,
202     BoxLayout.Y_AXIS));
203             originControls.setLayout(new        BoxLayout(originControls,
204     BoxLayout.Y_AXIS));
205             targetOriginControls.setLayout(new
206     BoxLayout(targetOriginControls, BoxLayout.X_AXIS));
207             camControls.setLayout(new FlowLayout());
208             display.setLayout(new FlowLayout());
209
210             camControls.add(new JLabel("Camera:"));
211             camControls.add(selectCam);
212             camControls.add(new JLabel("Camera AOV:"));
213             camControls.add(angle);
214             camControls.add(new JLabel("Camera resolution:"));
215             camControls.add(selectRes);
216             camControls.add(debug);
217
```

```
218             searchControls.add(new JLabel("Calculated size step:"));
219             searchControls.add(size);
220             searchControls.add(new    JLabel("Calculated    size    angle
221   step:"));
222             searchControls.add(sizeAngle);
223             searchControls.add(save);
224             searchControls.add(load);
225             searchControls.add(fileException);
226
227             target1Controls.add(target1);
228             target1Controls.add(new JLabel(" tolerance:"));
229             target1Controls.add(color1);
230             target1Controls.add(new JLabel(" size:"));
231             target1Controls.add(size1);
232
233             target2Controls.add(target2);
234             target2Controls.add(new JLabel(" tolerance:"));
235             target2Controls.add(color2);
236             target2Controls.add(new JLabel(" size:"));
237             target2Controls.add(size2);
238
239             target3Controls.add(target3);
240             target3Controls.add(new JLabel(" tolerance:"));
241             target3Controls.add(color3);
242             target3Controls.add(new JLabel(" size:"));
243             target3Controls.add(size3);
244
245             targetControls.add(target1Controls);
246             targetControls.add(target2Controls);
247             targetControls.add(target3Controls);
248
249             origin1Controls.add(new JLabel("Camera horizontal angle:"));
250             origin1Controls.add(theta);
251             origin1Controls.add(new JLabel("Camera vertical angle:"));
252             origin1Controls.add(phi);
253
254             origin2Controls.add(new JLabel("Camera coordinates: "));
255             origin2Controls.add(x);
256             origin2Controls.add(new JLabel(", "));
257             origin2Controls.add(y);
258             origin2Controls.add(new JLabel(", "));
259             origin2Controls.add(z);
260
261             origin3Controls.add(new JLabel("Screen size: "));
262             origin3Controls.add(screenSize);
263             origin3Controls.add(new JLabel("Screen resolution: "));
264             origin3Controls.add(screenResolution);
265
266             originControls.add(origin1Controls);
267             originControls.add(origin2Controls);
268             originControls.add(origin3Controls);
269
270             targetOriginControls.add(targetControls);
271             targetOriginControls.add(new
272   JSeparator(SwingConstants.VERTICAL));
273             targetOriginControls.add(originControls);
274
```

```
275             imageControls.add(new   JLabel("AR   image   angular   sizes
276   (rad/1000):"));
277             imageControls.add(angularWidth);
278             imageControls.add(new JLabel(","));
279             imageControls.add(angularHeight);
280             imageControls.add(new JLabel("AR image scale factor:"));
281             imageControls.add(arScale);
282
283             frame.getContentPane().add(camControls);
284             frame.getContentPane().add(searchControls);
285             frame.getContentPane().add(targetOriginControls);
286
287             imageFrame.getContentPane().add(arLabel);
288             frame.getContentPane().add(imageControls);
289
290             display.add(imageLabel);
291
292             display.add(coordinateLabel);
293
294             frame.getContentPane().add(display);
295
296             frame.getContentPane().add(timeLabel);
297
298             imageFrame.setVisible(true);
299             frame.setVisible(true);
300
301             ColorFinder              finder              =              new
302   ColorFinder(Webcam.getWebcamByName(selectCam.getSelectedItem().toStrin
303   g()),                  resX,                  resY,                  50,
304   targettedColorArray[target3.getSelectedIndex()]);
305             finder.camStart();
306
307
308             //Main loop. Runs until program exit.
309             while (true) {
310                 long startTime = System.nanoTime();
311
312                 //Save settings
313                 if (save.isSelected()) {
314                     try {
315                         FileOutputStream  f  =  new  FileOutputStream(new
316   File("config.dat"));
317                         ObjectOutputStream o = new ObjectOutputStream(f);
318
319                         o.writeObject(color1.getValue());
320                         o.writeObject(color2.getValue());
321                         o.writeObject(color3.getValue());
322
323                         o.writeObject(size1.getValue());
324                         o.writeObject(size2.getValue());
325                         o.writeObject(size3.getValue());
326
327                         o.writeObject(x.getValue());
328                         o.writeObject(y.getValue());
329                         o.writeObject(z.getValue());
330
```

```
331                         o.writeObject(screenSize.getValue());
332                         o.writeObject(screenResolution.getValue());
333 
334                         o.writeObject(size.getValue());
335                         o.writeObject(sizeAngle.getValue());
336 
337                         o.writeObject(theta.getValue());
338                         o.writeObject(phi.getValue());
339                         o.writeObject(angle.getValue());
340 
341                         o.writeObject((int) target1.getSelectedIndex());
342                         o.writeObject((int) target2.getSelectedIndex());
343                         o.writeObject((int) target3.getSelectedIndex());
344 
345                         o.close();
346                         f.close();
347                     } catch (FileNotFoundException e) {
348                         fileException.setText("File not found");
349                     } catch (IOException e) {
350                         fileException.setText("Error         initializing
351 stream");
352                     }
353                     save.setSelected(false);
354                 }
355                 //Load settings
356                 if (load.isSelected()) {
357                     try {
358                         FileInputStream  fi  =  new  FileInputStream(new
359 File("config.dat"));
360                         ObjectInputStream oi = new ObjectInputStream(fi);
361 
362                         color1.setValue(oi.readObject());
363                         color2.setValue(oi.readObject());
364                         color3.setValue(oi.readObject());
365 
366                         size1.setValue(oi.readObject());
367                         size2.setValue(oi.readObject());
368                         size3.setValue(oi.readObject());
369 
370                         x.setValue(oi.readObject());
371                         y.setValue(oi.readObject());
372                         z.setValue(oi.readObject());
373 
374                         screenSize.setValue(oi.readObject());
375                         screenResolution.setValue(oi.readObject());
376 
377                         size.setValue(oi.readObject());
378                         sizeAngle.setValue(oi.readObject());
379 
380                         theta.setValue(oi.readObject());
381                         phi.setValue(oi.readObject());
382                         angle.setValue(oi.readObject());
383 
384                         target1.setSelectedIndex((int) oi.readObject());
385                         target2.setSelectedIndex((int) oi.readObject());
386                         target3.setSelectedIndex((int) oi.readObject());
387 
```

```
                    oi.close();
                    fi.close();
                    fileException.setText(null);
                } catch (FileNotFoundException e) {
                    fileException.setText("File not found");
                } catch (IOException e) {
                    fileException.setText("Error         initializing
stream");
                } catch (ClassNotFoundException ex) {
                    fileException.setText("Error: wrong or corrupted
config file");
                }
                load.setSelected(false);
            }
            //Read settings
            finder.setTarget(0,
targettedColorArray[target3.getSelectedIndex()]);

            finder.setDebug(debug.isSelected());

            if (oldRes != selectRes.getSelectedIndex()) {
                oldRes = selectRes.getSelectedIndex();
                switch (oldRes) {
                    case 0:
                        resX = 640;
                        resY = 480;
                        break;
                    case 1:
                        resX = 320;
                        resY = 240;
                        break;
                    default:
                        resX = 176;
                        resY = 144;
                        break;
                }

finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
String()), resX, resY);
            }
            if
(!finder.getCamName().equals(selectCam.getSelectedItem().toString()))
{

finder.camChange(Webcam.getWebcamByName(selectCam.getSelectedItem().to
String()));
            }
            finder.setTol(parseInt(color3.getValue().toString()), 0);

            long setTime = System.nanoTime();

            //read camera
            finder.find3D((int)
parseDouble(size.getValue().toString()),
parseDouble(sizeAngle.getValue().toString()),
parseDouble(angle.getValue().toString()),
```

```
parseDouble(theta.getValue().toString()),
parseDouble(phi.getValue().toString()),
parseDouble(x.getValue().toString()),
parseDouble(y.getValue().toString()),
parseDouble(z.getValue().toString()),
parseDouble(screenResolution.getValue().toString())                    /
parseDouble(screenSize.getValue().toString()),
parseDouble(size3.getValue().toString()));

              long searchTime = System.nanoTime();
              //update ar image
              double                        awidth                          =
parseDouble(angularWidth.getValue().toString()) / 1000;
              double                       aheight                          =
parseDouble(angularHeight.getValue().toString()) / 1000;
              Point origin = arLabel.getLocationOnScreen();
              int                                                  aRScale=
(int)parseDouble(arScale.getValue().toString());
              ar                            =                          new
BufferedImage(imageFrame.getBounds().width/aRScale,
imageFrame.getBounds().height/aRScale, BufferedImage.TYPE_INT_ARGB);
              for (int yscreen = origin.y; yscreen < ar.getHeight() +
origin.y; yscreen++) {
                  int   yar   =   (int)   ((atan(-(yscreen*aRScale   -
finder.getPositions().get(0)[1]) / finder.getPositions().get(0)[2]) +
aheight / 2) * frame.getBounds().height / aheight);
                  for (int xscreen = origin.x; xscreen < ar.getWidth()
+ origin.x; xscreen++) {
                      int  xar  =  (int)  ((atan(-(xscreen*aRScale  -
finder.getPositions().get(0)[0]) / finder.getPositions().get(0)[2]) +
awidth / 2) * frame.getBounds().width / awidth);
                      if (xar >= 0 && xar < img.getWidth() && yar >= 0
&& yar < img.getHeight()) {
                          ar.setRGB(xscreen   -   origin.x,   yscreen   -
origin.y, img.getRGB(xar, yar));
                      } else {
                          ar.setRGB(xscreen   -   origin.x,   yscreen   -
origin.y, Color.BLACK.getRGB());
                      }
                  }
              }

              arLabel.setIcon(new
ImageIcon(ar.getScaledInstance(imageFrame.getBounds().width,
imageFrame.getBounds().height, BufferedImage.SCALE_FAST)));

              long arTime = System.nanoTime();

              //show results
              if (debug.isSelected()) {
                  imageLabel.setIcon(new
ImageIcon(finder.imageWithCenters().getScaledInstance(640,         480,
BufferedImage.SCALE_FAST)));
              } else {
                  imageLabel.setIcon(null);
              }
```

```
            coordinateLabel.setText("<html>  Cartesian  coordinates
(screen pixels):<br>" + finder.cartesian3DString("<br>", "<br><br>") +
"Spheric      coordinates      (cm      to      camera):<br>"      +
finder.sphericString("<br>", "<br><br>") + "</html>");

            long endTime = System.nanoTime();
            timeLabel.setText(String.format("set: %s search: %s ar:
%s  display:  %s  total:  %s",  toString(setTime  -  startTime),
toString(searchTime  -  setTime),  toString(arTime  -  searchTime),
toString(endTime - arTime), toString(endTime - startTime)));
        }
    }

    /**
     * Generates a string containing time data. Used for debugging
and test
     * purposes.
     *
     * @param nanoSecs
     * @return
     */
    private static String toString(long nanoSecs) {
        int minutes = (int) (nanoSecs / 60000000000.0);
        int seconds = (int) (nanoSecs / 1000000000.0) - (minutes *
60);
        int millisecs = (int) (((nanoSecs / 1000000000.0) - (seconds
+ minutes * 60)) * 1000);

        if (minutes == 0 && seconds == 0) {
            return millisecs + "ms";
        } else if (minutes == 0 && millisecs == 0) {
            return seconds + "s";
        } else if (seconds == 0 && millisecs == 0) {
            return minutes + "min";
        } else if (minutes == 0) {
            return seconds + "s " + millisecs + "ms";
        } else if (seconds == 0) {
            return minutes + "min " + millisecs + "ms";
        } else if (millisecs == 0) {
            return minutes + "min " + seconds + "s";
        }

        return minutes + "min " + seconds + "s " + millisecs + "ms";
    }

}
```