DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

PROYECTO FIN DE GRADO

DETECCIÓN Y EXTRACCIÓN DE CARACTERÍSTICAS DE LA MANO HUMANA MEDIANTE TECNOLOGÍA PRIMESENSE

Autor: Roberto Montero Pérez

Tutor: Alejandro Lumbier Álvarez

LEGANÉS

SEPTIEMBRE 2012

"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica"

Aristóteles, (384 AC - 322 AC).

AGRADECIMIENTOS

A mi familia, mis padres Antonio y María José, y mi hermano Enrique, que me han apoyado y aconsejado durante toda la carrera.

A mis amigos, Adrián e Ismael, que siempre escuchan todas las cosas raras que les cuento sin rechistar.

A mis compañeros, en especial a Borja, Erik, Víctor y Santiago, por compartir conmigo esta aventura.

Y también a mi tutor Alejandro, por confiar en mí y ayudarme a mejorar cada día.

A mi abuela Josefa, que fue como una segunda madre para mí

Índice general

Lista de Figuras

Resumen

Uno de los campos más importantes de la robótica actualmente es el de la percepción artificial del entorno, entendiendo como tal la adquisición de los datos visuales por medio de cámaras y su posterior tratamiento. Es lógico que siendo la vista uno de los sentidos más importantes para el ser humano, también lo sea para los sistemas que pretendemos tomen decisiones a nuestro nivel.

Sin embargo, aspectos que para nosotros pueden parecer triviales, como la diferenciación entre objetos, no lo son en absoluto a la hora de procesarse digitalmente por medio de un ordenador. Los problemas que nos encontramos son, entre otros, la identificación de objetos, extracción de sus características y seguimiento de los mismos.

De esta forma, el objetivo principal de este proyecto es identificar la mano humana, extraer sus características, principalmente la posición de los dedos y la palma, y hacer un seguimiento de los mismos en un espacio 3D. Para ello utilizaremos una cámara Kinect de Microsoft, equipada con tecnología Primesense.

El objetivo último del proyecto es dotar al robot Manfred, de la Universidad Carlos III de Madrid, de la capacidad de analizar la manera en la que se manipulan diversos objetos, de forma que posteriormente será capaz de llevarlo a cabo su mano robótica.

Palabras clave:

robótica, percepción, mano, detección de objetos, Kinect, Manfred.

Abstract

Nowadays, one of the most important fields of robotics is artificial perception of the environment. This means the acquisition of visual data by cameras and their subsequent treatment. It makes sense that being the vision one of the most important senses for humans, it is also for systems that we aim to make decisions at our level.

However, aspects that may seem trivial to us, like object detection, are not at all when digitally processed by a computer. The problems we encounter include the identification of objects, feature extraction, and tracking.

Thus, the main objective of this project is to identify the human hand, extract features, mainly the position of the fingers and palm, and keep track of them in 3D space. We will use a Microsoft Kinect camera, equipped with PrimeSense technology.

The ultimate goal of the project is to provide the robot Manfred, University Carlos III of Madrid, the ability to analyse how various objects are manipulated so that later will be able to carry out with it's robotic hand.

Keywords:

robotics, perception, hand, object detection, Kinect, Manfred.

Capítulo 1

Introducción

Este capítulo, que sirve como introducción al proyecto, tratará de explicar brevemente el significado y la trayectoria de la robótica, y dentro de ésta, de la visión artificial o por computador.

1.1. Sobre la robótica

La robótica se puede definir como la rama tecnológica encargada del diseño y construcción de robots, así como la programación y control de los mismos. Por ello tiene carácter multidisciplinar, y en ella participan la mecánica, electrónica, informática o ingeniería de control entre otras.

El nombre de robótica deriva de la palabra *'robot'*, utilizada por primera vez en 1921 en la obra **R.U.R (Rossum's Universal Robots)**, del escritor checo **Karel Capek**. En esta obra, *'robot'* designaba a unas máquinas con forma humana que realizaban diversos trabajos de manera autónoma. A su vez, *'robot'* deriva del termino eslavo *'robota'*, que significa "trabajos forzados".

Así pues, el fin último de los robots, tal y como se entiende hoy en día, es el de mejorar nuestra calidad de vida y hacérnosla más fácil, ya sea ejerciendo tareas industriales, domésticas, de ayuda en salvamento o quirúrgicas por nombrar algunas.

Sin embargo, aunque la robótica es un campo relativamente actual y novedoso, la ambición del ser humano por crear mecanismos autónomos es muy antigua. Uno de los autómatas mas antiguos de los que se tiene constancia data del siglo III a.C. Se trataba de una figura humana de tamaño natural que se movía y cantaba, según un texto del chino *Lie Zi*.

Dado que la idea de los robots es que realicen tareas de manera lo más parecida a como lo haría una persona, resulta indispensable dotarles de nuestras mismas capacidades de percepción del entorno, y en muchos casos, de una morfología similar. Así, durante estos últimos años, los robots han evolucionado su forma y su control para parecerse más a nosotros y poder ejercer tareas de más alto nivel de manera autónoma. Por eso no es de extrañar que la visión del robot del futuro sea la del robot humanoide.

En este sentido nos centraremos en cómo los robots captan la información visual del entorno, en particular la cromática y la referente a la posición 3D de los objetos, y cómo podemos procesarla para lograr nuestros objetivos. Siendo algo vital para el desarrollo de la inteligencia artificial y el aprendizaje autónomo.

1.2. Visión artificial

Una de las principales características que tienen los robots modernos es la capacidad de recibir información del entorno por medio de sensores. Esto les permite adecuar su comportamiento a las necesidades de cada situación o incluso aprender determinadas acciones por sí mismos.

La visión artificial es la encargada de procesar y analizar la información visual que se obtiene a través de las cámaras. Algunas de las aplicaciones típicas de la visión artificial o por computador relacionadas con la robótica son:

- Reconocimiento de objetos, personas, símbolos, etc. y estimación del posicionamiento de los mismos.

- Reconocimiento facial y distinción entre diferentes rostros.
- Seguimiento de objetos y análisis de su movimiento en cadenas de frames.
- Mapeado del entorno y localización de la posición del robot en relación al entorno.

Con la llegada de los computadores las expectativas en cuanto a resolver problemas por medio de cámaras digitales eran muy altas. Se empezó a hablar de inteligencia artificial y de cibernética, que busca definir los fundamentos del control automático de una máquina bajo un punto de vista inspirado en la biología. Así, la percepción empieza a tener un papel importante en la inteligencia artificial.

Durante los años 50 y 60 se empieza a trabajar en la visión artificial y reina un gran optimismo, ya que la tarea de ver en el ser humano parece algo fácil e intuitivo a priori. Además se tiene una gran confianza en el poder de los ordenadores. Aparecen los primeros trabajos en procesamiento automático de imágenes (Moravec, Sussman, Roberts...), aplicaciones industriales y de control, y trabajos de morfología matemática (Matheron, Serra, Haralick).

Sin embargo, durante los 70 y 80, y a pesar de algunos éxitos, los avances obtenidos en este campo no estaban a la altura de las expectativas. Algunos de los problemas que tenían eran que la imagen es una proyección bidimensional de una escena tridimensional, lo que genera múltiples soluciones, además de los recursos de computación limitados, robustez, desconocimiento del proceso de visión humano, etc... Además se empieza a entender que lo que resulta fácil para los humanos (reconocimiento facial y de objetos, navegación, etc.), es decir, tareas de alto nivel, no lo es para los ordenadores. Por el contrario, a los computadores les resulta más sencillo las tareas de bajo nivel (capacidad de procesamiento matemático). Por ello durante los 80 se enfoca la visión por computador desde una perspectiva más realista y se simplifican los problemas.

Con respecto a la metodología, se pueden considerar dos enfoques principales a la hora de resolver problemas en visión artificial: Bottom-Up y Top-Down

- El enfoque Bottom-Up parte de la imagen hasta llegar al conocimiento que precisamos. Se suele utilizar en aplicaciones industriales.

- Por el contrario, el enfoque Top-Down parte del conocimiento recopilado con anterioridad e intenta asociarlo a la imagen. Este sería el enfoque utilizado en tareas de reconocimiento de patrones u objetos (template matching) por medio de una base de datos.

Actualmente se suele utilizar una mezcla de los dos, con soluciones que utilizan aspectos de ambos métodos.

Hoy en día, y gracias a la mejora en la capacidad de los ordenadores o la introducción de las cámaras 3D, la visión artificial está a la orden del día. De esta manera está presente en multitud de aplicaciones, desde tratamiento de imágenes biomédicas, guiado de robots industriales o identificación biométrica hasta la inteligencia artificial de los robots del futuro.

Capítulo 2

Objetivos

En este capítulo nos centraremos en explicar los objetivos del proyecto. El fin del trabajo es programar una aplicación que segmente una mano humana e identifique en ella la posición de la palma y la punta de los dedos. Posteriormente se aplicará un seguimiento a esos puntos. Los datos se recopilan mediante una cámara 3D Kinect de Microsoft.

Este trabajo, junto con otros de reconocimiento de objetos o cinemática de manos robóticas, se enmarca dentro de un proyecto mayor que pretende enseñar a un robot a manipular objetos de manera eficiente. El aprendizaje se realiza grabando múltiples escenas de personas manipulando esos mismos objetos para posteriormente calcular los puntos de agarre óptimos que el robot intentará utilizar después de manera autónoma. El robot en cuestión es MANFRED (MANipulator FRiEnDly robot), de la Universidad Carlos III de Madrid, y está equipado con un brazo de fibra de carbono y una pinza en su extremo que será reemplazada en un futuro por una mano robótica.

MANFRED nació en el Departamento de Sistemas y Automática en 2001. La primera parte de su desarrollo se centró en el diseño mecánico del robot, con especial énfasis en su brazo. Este hecho le hizo convertirse en un robot pionero en España, ya que fue el primero que tuvo un brazo construido en fibra de carbono, con un peso aproximado de sólo 18 Kg.

A partir de 2003 se empezó a trabajar en su software en diferentes ámbitos relacionados con su inteligencia artificial como procesos de aprendizaje del entorno y localización. Esto ha per-

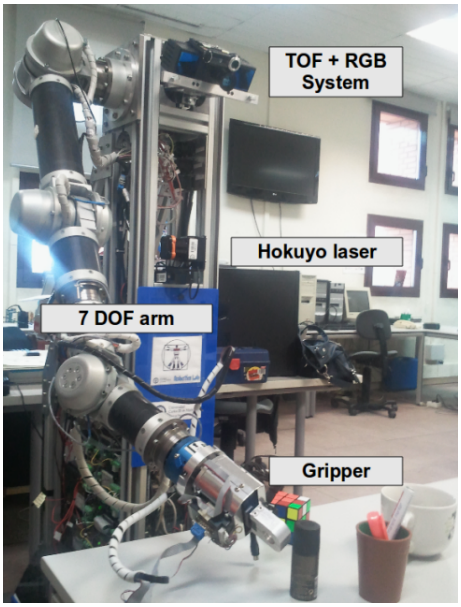

Figura 2.1: Especificaciones de MANFRED.

mitido que actualmente MANFRED tenga todas las características propias de un robot móvil, siendo capaz de moverse de manera autónoma evitando obstáculos y calculando trayectorias. También se ha conseguido una buena coordinación entre la base y el brazo manipulador de forma que es capaz de agarrar y mover objetos. Actualmente se trabaja con el robot MANFRED-2, una evolución del primero más compacta, como se puede apreciar en la figura 2.3.

Volviendo a nuestro proyecto, el objetivo del mismo se conseguirá siguiendo los siguientes pasos:

1. Obtención de los datos provenientes de la cámara Kinect.
2. Localización de la mano en la secuencia de vídeo capturada.
3. Caracterización de los puntos de interés (palma y dedos).
4. Seguimiento de los puntos calculados anteriormente utilizando un filtro de Kalman.
5. Representación visual de la escena y los datos en 3D.

Estos objetivos se conseguirán utilizando las bibliotecas de OpenCV, OpenGL y OpenFrameWorks. El lenguaje de programación es C++ y el sistema operativo Linux Ubuntu 10.10.

Figura 2.2: El robot MANFRED.

Figura 2.3: El robot MANFRED-2.

Capítulo 3

Estado del arte

En este capítulo haremos un repaso por las distintas soluciones que existen actualmente para el problema de captar la mano humana y sus características. En segundo lugar se nombrarán los proyectos que han servido de inspiración y las alternativas que se han tenido en cuenta a la hora de hacer este trabajo.

3.1. Adquisición de datos

De cara a tratar el problema de la localización de la mano humana uno de los aspectos a tener en cuenta es qué tipo de información tenemos a la hora de aplicar los algoritmos. Así en este sentido se han utilizado diversos métodos de captura de imágenes. Algunas soluciones utilizan cámaras convencionales para captar las imágenes como en [3] y [22], por lo que se utiliza información en 2D para la segmentación. Otro acercamiento se basa en la obtención de datos en 3D mediante un par de cámaras estereoscópicas, usadas en [2] y [21], o incluso 8 cámaras [26]. También se utilizan cámaras TOF [16] o de tipo PrimeSense como la que utilizamos en este proyecto en [27].

3.2. Detección

Una vez definida la adquisición de los datos, el primer paso a tener en cuenta en el sistema es la detección de la mano y su segmentación. De esta forma se centra el procesado en la región de la mano y se pueden aplicar los algoritmos necesarios. Existen una gran cantidad de métodos que utilizan diversas características de la imagen o la combinación de varias. Estas características son entre otras color, forma, movimiento y modelos anatómicos de la mano.

3.2.1. Color

La segmentación por color depende en gran medida del espacio de color elegido para trabajar. Preferiblemente se suelen utilizar espacios de color donde la luminancia y crominancia están separadas, de manera que se descarta el modelo RGB y se trabaja con HSV [30], YCrCb [8] o YUV [1].

Una vez que se ha elegido el espacio de color apropiado, las técnicas para definir las regiones a segmentar pueden ser simples como la utilizada por Chai y Ngan [8], donde se definen unos límites a los valores de color que puede tomar el píxel y se binariza la imagen dependiendo de si el píxel se encuentra dentro de los límites o no. Otra técnica utilizada por Saxe y Foulds [30] define la probabilidad de los píxeles de pertenecer a la región dentro de una distribución que puede "aprenderse" mediante métodos off-line u on-line. Argyros y Lourakis [1] utilizan la histéresis para clasificar los píxeles de forma que si un píxel tiene una probabilidad baja de pertenecer a la región, se clasifica como tal si a su alrededor existen píxeles con una alta probabilidad.

Uno de los problemas de la identificación por color aparece cuando hay objetos en la escena con un color semejante al de la piel. Para evitar ese problema se utilizan técnicas de extracción de fondo como las utilizadas por Gavrila y Davis [14] para cámaras fijas. Otro método se aplica en [5] para cámaras móviles donde Blake, North e Isard utilizan una corrección dinámica del fondo. En el caso de que la cara del usuario aparezca también en la imagen se requiere de un procesado extra para localizar la mano debido a la similitud de sus colores. Zhou y Hoang [36] utilizan las formas de los objetos detectados mediante extracción de fondo para determinar si

son humanos o no en una escena, lo que se puede aplicar a la forma de la cara o las manos.

3.2.2. Forma

Otra característica utilizada para identificar la mano es su forma. En general, aplicar una detección de contornos a la imagen nos proporciona los contornos de la mano, pero también de múltiples objetos de la imagen, de forma que es necesario un procesamiento posterior. Por este motivo se suele acompañar la extracción de contorno con la segmentación por color y la extracción de fondo, de forma que sólo obtengamos el contorno de la mano, que es el que nos interesa.

Utsumi y Ohya [33] segmentan la mano del usuario y se obtiene su contorno asumiendo un fondo uniforme en la escena y aplicando extracción de bordes en tiempo real. Downton y Drouet [12] extraen los dedos y el brazo mediante un *clustering* de grupos de bordes paralelos. Con un objetivo más general, Gavrila y Davis [13] evalúan hipótesis de modelos 3D de la mano extrayendo de esos modelos una imagen de sus bordes y comparándola con los bordes obtenidos anteriormente.

Otro tipo de acercamiento es de los descriptores topológicos. Belongie, Malik y Puzicha [4] utilizan como descriptor del borde de la mano el histograma de las coordenadas relativas polares de los puntos del borde. La detección se basa en la suposición de que los puntos correspondientes de dos bordes tienen un descriptor similar.

También se utilizan métodos morfológicos para detectar características de la mano tales como las puntas de los dedos. Argyros y Lourakis [2] utilizan la curvatura del borde de la mano para identificar las puntas de los dedos de forma similar a como se hace en este proyecto. Otra técnica utilizada para la obtención de las puntas de los dedos es la de *template matching*, muy utilizada en reconocimiento de objetos o símbolos. Esta técnica la utilizan Crowley, Berard y Coutaz en [9], donde las plantillas son imágenes de puntas de dedos; también Rehg y Kanade en [29] donde son dedos completos; o Davis y Shah en [11], donde las plantillas son modelos 3D cilíndricos. Sin embargo, uno de los problemas del *template matching* es que la identificación depende de la orientación y el tamaño del objeto, que tiene que coincidir con la plantilla. Este

problema lo tienen en cuenta Crowley, Berard y Coutaz [9], donde la plantilla se va actualizando continuamente.

3.2.3. Movimiento

Algunas soluciones utilizan el movimiento para definir la mano partiendo de la suposición de que es el único objeto de la escena que se mueve. Esta suposición la utilizan Cui y Weng en [10]. Por otro lado, Martin, Devin y Crowley [23] distinguen las manos con la información de color en frames adyacentes. Se parte de la suposición de que el cambio en la luminancia en dos frames seguidos es próxima a 0 para objetos del fondo de la escena. De esta forma, eligiendo y manteniendo unos umbrales adecuados se separan los objetos en movimiento del fondo.

3.2.4. Modelos de la mano

En este tipo de soluciones se utilizan modelos 3D de la mano humana con suficientes grados de libertad para definir los puntos que se necesiten. Wu, Lin y Huang [35] utilizan puntos y líneas características en un modelo cinemático de la mano y se calculan los ángulos que forman las articulaciones. Posteriormente la postura de la mano se define comparando estas características en el modelo 3D y en la imagen.

Otro tipo de modelos son los encontrados en ([32], [18], [15], [14], [20]). Stenger, Mendonca y Cippola [32] utilizan un modelo de 27 grados de libertad (6 para la posición y orientación general y 21 para las articulaciones). Lin, Wu y Huang [18] utilizan un modelo donde cada dedo se representa con 3 planos conectados. Goncalves, di Bernardo, Ursella y Perona [15] utilizan un modelo 3D que representa el brazo y tiene 7 parámetros. Gavrila y Davis [14] utilizan un modelo para todo el cuerpo de 22 grados de libertad, con 4 parámetros para cada brazo. Finalmente, MacCormick e Isard [20] utilizan el modelo más simple, definiendo la mano como un objeto rígido con 3 partes, el pulgar, el índice y la palma.

3.3. Tracking

El seguimiento frame a frame de la mano y sus características es el segundo paso a seguir a la hora de obtener la posición de la mano en movimiento. Este seguimiento se utiliza para definir las trayectorias de la mano de cara a definir gestos o se puede utilizar para optimizar los métodos basados en modelos. Además, puede proporcionar información sobre puntos o características que no han sido identificadas en algún frame, por medio de predicciones.

Uno de los métodos de *tracking* se basa en el *template matching*. Este método es el que utilizan O'Hagan y Zelinsky en [25], donde se aplica *template matching* basada en correlación, y una vez se detecta la mano, se utiliza ese entorno para buscar la mano en el siguiente frame. Otro método se basa en el tracking de blobs, y es el que se suele utilizar si se utiliza segmentación por color. Argyros y Lourakis [1] utilizan este método basado en blobs para aplicar el tracking.

Bradski [6] utiliza una versión del algoritmo *mean shift algorithm* para seguir los blobs pertenecientes a la mano humana, segmentados por medio de color. Para mejorar la robustez de este método se hace el seguimiento del centroide del blob y se adapta la distribución de color de manera continua.

En cuanto a seguimiento óptimo y cálculo de predicciones se suele utilizar el filtro de Kalman. Breig y Kohler [7] utilizan el filtro de Kalman para realizar estimaciones sobre la orientación de la mano para localizar el punto del espacio al que apunta al extender el dedo índice. Utsumi y Ohya [34] utilizan un método en el que las manos se siguen con varias cámaras aplicando el filtro de Kalman en cada imagen.

Una alternativa al filtro de Kalman son los filtros de partículas, pero su principal problema al aplicarse a la mano humana es la cantidad de partículas que hacen falta debido al número de grados de libertad de la mano, lo que hace que aumente sus requerimientos de computación. Por ejemplo, en [17] Isard y Blake utilizan el *CONDENSATION algorithm* en tiempo real y se utiliza un modelo de mano reducido. Otro tipo de filtro de partículas es el filtro de Monte Carlo, utilizado por Pérez, Hue, Vermaak y Gangnet en [28].

De todas las soluciones que se han nombrado en este capítulo, han sido dos las que más se han tenido en cuenta a la hora de desarrollar el trabajo. Una es el trabajo de A. A. Argyros en [27], sin embargo, y dados los altos requerimientos necesarios para ejecutar el algoritmo, un acercamiento al problema de este trabajo mediante esta solución se hizo impracticable. La otra solución es la aplicación de Stefan Stegmueller, Candescent NUI [31], de la que se ha utilizado por ejemplo un método similar para calcular la palma de la mano.

Capítulo 4

Conceptos Teóricos

En este capítulo explicaremos los conceptos teóricos de algunos de los aspectos del trabajo. Empezaremos hablando de la visión humana y artificial, de las cámaras digitales y 3D, y terminaremos explicando algunos conceptos relacionados con el tratamiento de la imagen y los algoritmos utilizados.

4.1. Visión biológica

La vista es la habilidad de recibir e interpretar la información del entorno por medio de los rayos de luz. Es uno de los sentidos más importantes del ser humano y por los que recibe una gran cantidad de información. Los sensores biológicos encargados de recibir esa información son los ojos, que la transmiten al cerebro para su procesado.

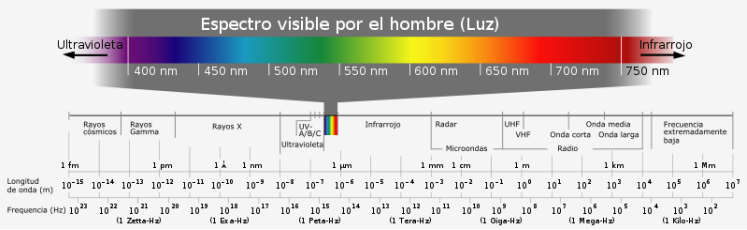

Figura 4.1: Espectro electromagnético y luz visible en detalle.

Los ojos humanos tienen un tamaño de unos $25mm$ de diámetro. Son sensibles a la ra-

diación electromagnética en un espectro concreto (luz visible), con una longitud de onda desde los $390nm$ hasta los $750nm$ aproximadamente, como se puede observar en la figura 4.1. Las partes de las que se compone, siguiendo el recorrido de la luz en el ojo son las siguientes:

- **Córnea**: Elemento transparente que recubre el globo ocular y protege el iris y el cristalino.
- **Iris**: Membrana donde está situada la pupila.
- **Pupila**: Actúa a modo de diafragma adecuándose a la cantidad de luz del entorno, permitiendo que se reciba más o menos luz.
- **Cristalino**: Su función es la de permitir enfocar los objetos, lo cual se consigue modificando las propiedades físicas del mismo (en concreto, la curvatura).
- **Musculo ciliar**: Es el encargado de modificar la forma del cristalino.
- **Humor acuoso**: Es un medio gelatinoso cuya función es el transporte de los elementos necesarios para nutrir a la córnea y el cristalino.
- **Humor vítreo**: Similar al humor acuoso, pero algo más denso y que es el encargado de dar su forma esférica al globo ocular.
- **Retina**: Se encuentra en la zona más interna del globo ocular y en ella se sitúan millones de células fotosensibles (los conos y bastones) encargadas de captar toda la luz que penetra en el ojo y transformarla en impulsos eléctricos.
  - **Conos**: Estas células son muy sensibles al color. Hay en torno a 7 millones, se encuentran localizadas en la fóvea y tienen su propia terminación nerviosa que las conecta directamente al cerebro. Existen 3 tipos distintos (S, M y L) cada una de ellas especializadas en captar una longitud de onda concreta, correspondientes al rojo, verde y azul respectivamente.
  - **Bastones**: Estas células son sensibles a la intensidad lumínica. Su número es muy superior a los conos, situándose entre los 70 y los 140 millones, que se distribuyen por toda la retina.

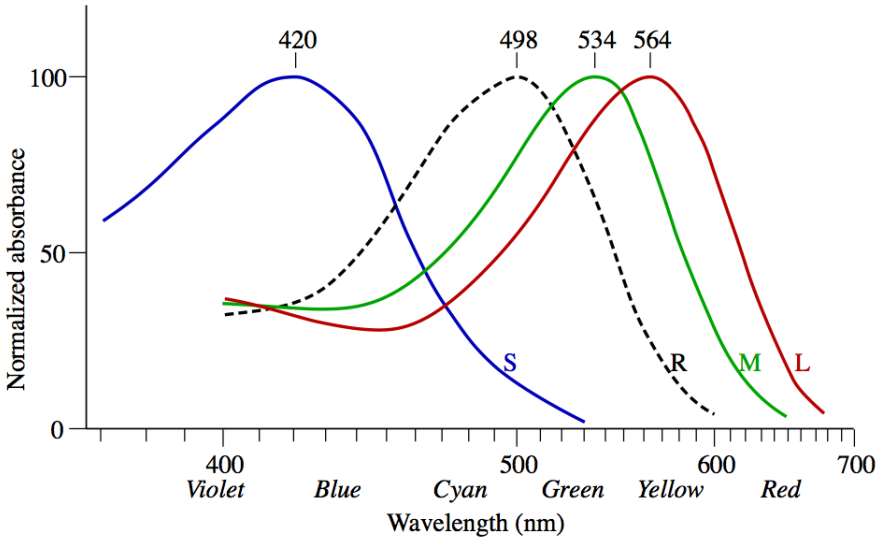

Figura 4.2: Longitudes de onda absorbidas por los bastones (curva R) y los conos S, M y L.

- **Fóvea central**: Situada en la retina, es el punto donde la agudeza visual es máxima ya que es aquí donde se enfocan los rayos de luz.
- **Nervio óptico**: A través de él se transmiten las señales eléctricas generadas en la retina al córtex cerebral.

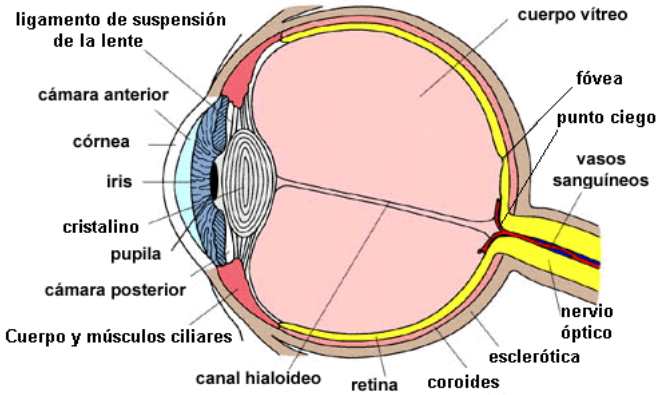

Figura 4.3: Ojo humano y las partes de las que se compone.

Como hemos explicado, el ojo tiene su propio sistema de óptica para enfocar la imagen (cristalino), además de ser capaz de modificar la cantidad de iluminación que recibe (pupila).

Una vez la luz llega a la retina, se recibe con dos tipos de sensores, uno especializado en captar la intensidad (bastones) y otro el color (conos). Estos cuatro elementos del ojo humano son los que se toman como modelo a la hora de fabricar las cámaras digitales.

4.2. Cámaras digitales

Las cámaras, de foto o de vídeo, tienen una estructura similar a la que hemos visto anteriormente. Capturan la luz por medio de una serie de lentes de diferentes tamaños y formas. Estas lentes son modificables al igual que el cristalino, y pueden adecuar la cantidad de iluminación que reciben como lo hace la pupila.

Posteriormente la luz se hace pasar por unos cristales polarizados que separan la luz en 3 componentes pertenecientes a las longitudes de onda del rojo, verde y azul. De esta manera se imita el funcionamiento de los conos a la hora de procesar el color. Hay dos formas principales de separar la luz:

- Filtro de Bayer: De bajo costo y más común, crea una matriz con filtros de color de forma que hay píxeles dedicados a cada color (rojo, verde y azul). De esta forma sólo se utiliza un sensor, y para cada color se rellenan los píxeles que faltan mediante interpolación cromática.

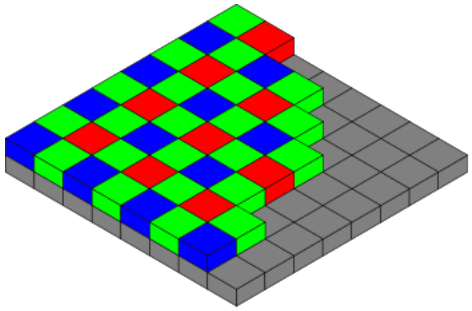

Figura 4.4: Filtro de bayer. Se puede observar la matriz de color resultante.

- Prisma dicroico: Se hace pasar la luz por un prisma con una serie de cristales que filtran la luz en las componentes rojo, verde y azul, y se envían a tres sensores diferentes. Mediante

este método no es necesaria la interpolación cromática, pero en cambio se necesita la colocación de tres sensores de imagen de las mismas características.

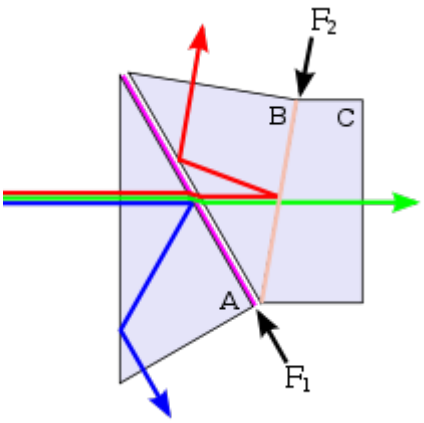

Figura 4.5: Prisma dicroico. Cada componente tiene una dirección diferente, donde llega a su sensor correspondiente.

Una vez la información de color llega a los sensores, estos la transforman en impulsos eléctricos que posteriormente son procesados para formar la imagen. Hay varios tipos de sensores de imagen en las cámaras digitales (CCD, SuperCCD, CMOS, Foveon X3...), aunque los más utilizados a día de hoy son CMOS y CCD.

- CCD: Los sensores CCD (Charge-Coupled Device) se basan en el efecto fotoeléctrico para transformar la luz en señales eléctricas, de forma que el número de electrones producido es proporcional a la cantidad de luz recibida. Esta información se almacena en forma de carga eléctrica en un condensador, de manera que mientras se lee la información del CCD se debe pasar por un chip externo, un ADC (Analog Digital Converter) para manejar datos digitales posteriormente. En cuanto a las ventajas con respecto a los CMOS podemos señalar su mayor rango dinámico y respuesta uniforme. En cambio, sufren de *blooming* cuando una celda se satura y empieza a saturar a las de su alrededor. Este efecto puede observarse en la figura 4.6.

Figura 4.6: Fenómeno de *blooming* con CCD's. Se observa que la saturación se produce en vertical debido a la manera en la que se recoge la información en el chip.

- CMOS: Los sensores CMOS (Complementary Metal-Oxide-Semiconductor) también se basan en el efecto fotoeléctrico, pero incluyen en cada celda su propio ADC, por lo que la digitalización del píxel se hace independientemente en cada celda. Por ello se le conoce también como APS (Active Píxel Sensor). Algunas de las ventajas de un CMOS con respecto al CCD son su menor consumo energético (mayor autonomía de la cámara), mayor velocidad de adquisición y precio más bajo. Sin embargo, aunque no sufren de *blooming*, si lo hacen de *rolling shutter*, debido a que no se procesa una imagen completa, sino que según se van leyendo las celdas, la imagen se va actualizando, lo que provoca errores en imágenes en movimiento. Este hecho se puede observar en la figura 4.7.

Figura 4.7: *Rolling shutter* en imágenes en movimiento. Se puede apreciar cómo las aspas del helicóptero aparecen deformadas.

4.3. Visión biológica vs visión artificial

Hemos visto que la forma en la que las cámaras reciben y procesan la información visual del entorno se parece mucho a como lo hace el ojo humano. Sin embargo, aún hay numerosas diferencias entre lo que las cámaras digitales pueden conseguir y lo que nuestra visión nos permite.

Algunos de los aspectos en los que las cámaras digitales superan al ojo humano son:

- Óptica: Las cámaras pueden equiparse con multitud de ópticas diferentes dependiendo de la aplicación en la que se utilicen. De esta manera pueden modificar fácilmente su campo de visión, zoom o profundidad de campo.
- Mejor resolución: Los sensores incorporados en las cámaras cada vez tienen más resolución. Además la resolución de la imagen es la misma en todo su conjunto. Esto no ocurre en la visión humana, donde la resolución de la fóvea es mucho mayor que la del resto de la imagen (visión periférica).
- Mayor rango espectral: Mientras el ojo humano sólo funciona en el rango de luz visible (ver figura 4.1) las cámaras digitales pueden adaptarse a múltiples rangos de funcionamiento. Algunos ejemplos son las cámaras de infrarrojos (IR), térmicas (basadas en infrarrojos), ultravioleta (UV) o multiespectrales, que funcionan en diferentes rangos del espectro.
- Mayor velocidad de adquisición: Las cámaras digitales de vídeo tienen una velocidad que puede llegar a los millones de fps (frames o imágenes por segundo) en las cámaras denominadas superlentas. La velocidad de captura de imágenes humana se puede aproximar a unos 30 fps, aunque en la visión biológica no se procesan frames.

En este sentido, las cámaras digitales superan al ojo humano en la adquisición de los datos. Pero a la hora de procesar las imágenes obtenidas, aun no se está a la altura del procesamiento que hace el cerebro de las mismas, aunque existen formas de "engañar" al cerebro en las llamadas ilusiones ópticas como la de la figura 4.8. En general el cerebro interpreta las imágenes obteniendo fácilmente la segmentación y localización de objetos, seguimiento, reconocimiento

de rostros, etc...

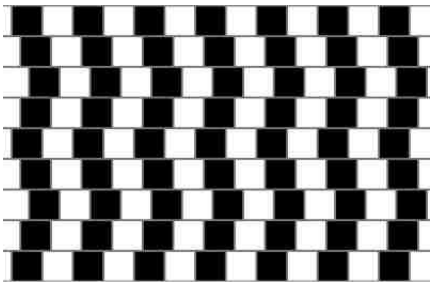

Figura 4.8: Ilusión óptica. Las líneas son paralelas aunque instintivamente no nos lo parezca.

Un aspecto del que no hemos hablado aun es el de la percepción de profundidad. El hecho de tener dos ojos separados nos proporciona lo que se llama visión estereoscópica, lo que nos permite percibir la profundidad de los objetos del entorno. Una de los acercamientos a cómo procesa las imágenes el cerebro consiste en obtener, además de los datos de color, los datos de profundidad de la imagen. Esto se ha conseguido mediante el uso de dos o más cámaras convencionales para obtener visión estereoscópica, o tecnologías más modernas que nos permiten calcular la distancia de los píxeles a la cámara de manera más directa, como la TOF (Time Of Flight) o PrimeSensor.

4.4. Cámaras 3D

Aunque existen varios métodos para calcular la profundidad en imágenes como interferometría, LIDAR, o microondas, mostradas en la figura 4.9, nos centraremos en la triangulación pasiva mediante par estereoscópico, TOF y PrimeSensor.

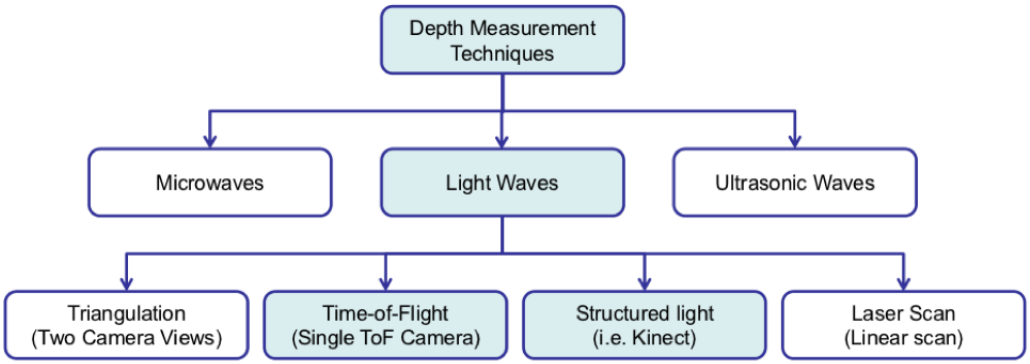

Figura 4.9: Técnicas de obtención de mapas de profundidad.

4.4.1. Visión estereoscópica

Una sistema estereoscópico consta de una cámara con dos lentes o dos cámaras separadas una distancia concreta. Esto permite capturar dos imágenes a la vez con una ligera traslación una respecto a la otra. De esta forma se puede calcular la distancia a un objeto concreto identificando y relacionando los píxeles de las dos imágenes y utilizando la diferencia de los mismos entre las imágenes. Estas cámaras suelen necesitar una calibración inicial, como se observa en [24].

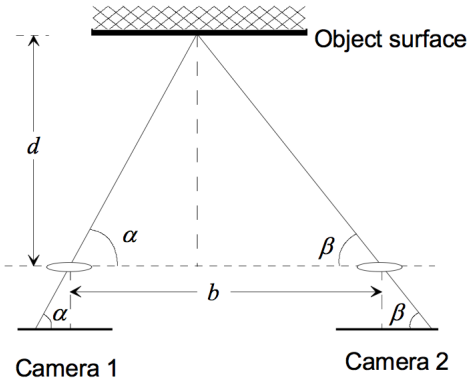

Figura 4.10: Obtención de la profundidad mediante triangulación.

Mediante mapas de disparidad se pueden encontrar las posiciones de los puntos de la escena en las dos imágenes, y posteriormente se aplica una triangulación para calcular la distancia al punto como se puede ver en la figura 4.10. La distancia se calcula mediante la ecuación 4.1:

$$d = \frac{b}{\frac{1}{\tan\alpha} + \frac{1}{\tan\beta}} \tag{4.1}$$

Los ángulos α y β se calculan a partir de la posición del píxel en las dos imágenes y la distancia focal de las cámaras.

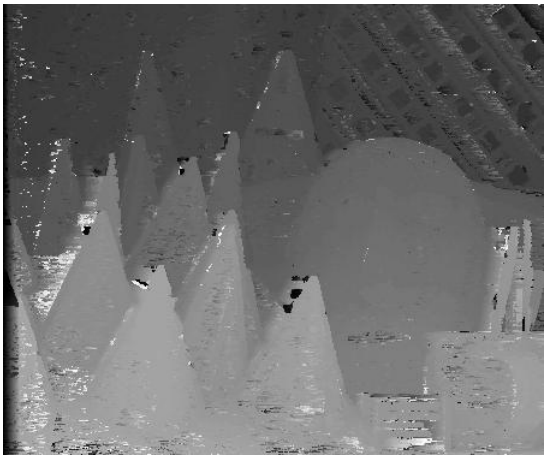

Figura 4.11: Ejemplo de mapa de profundidad obtenido mediante par estereoscópico.

4.4.2. TOF (Time Of Flight)

Otro método para obtener imágenes 3D es el de las cámaras TOF. Mientras que con el método estereoscópico necesitamos dos cámaras, mediante TOF sólo necesitamos un único sensor, un PMD (Photonic Mixer Device). Las cámaras TOF constan de un emisor de infrarrojos y un receptor PMD. El emisor emite un haz de luz IR en un momento determinado y cuando el receptor lo capta se calcula la distancia con el tiempo que ha tardado en volver. El error en la localización en posición de un punto de un objeto puede llegar a ser relativamente pequeño, del orden de milímetros, dependiendo del dispositivo TOF que utilicemos.

Gracias a la tecnología TOF nos ahorramos el procesamiento de las imágenes estéreo por

separado, de forma que obtenemos un mapa de profundidad con menos coste computacional. En la figura 4.12 se puede observar una comparación entre la imagen obtenida por una cámara estéreo[1] y una TOF[2].

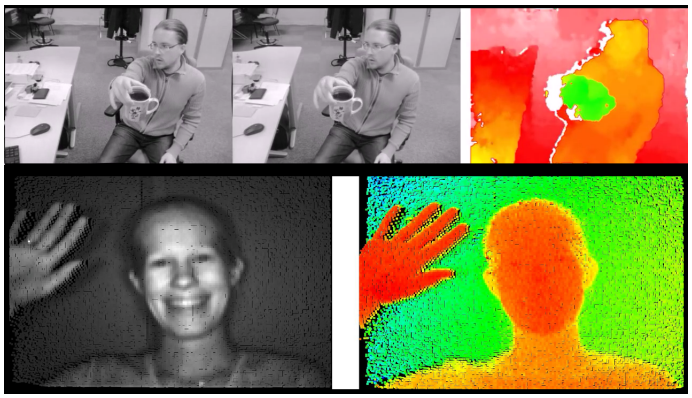

Figura 4.12: Comparativa de procesado entre visión estereoscópica arriba y tecnología TOF debajo.

Como lo normal es que se tomen medidas consecutivas con la cámara, se utiliza el desfase de las señales de entrada y salida para calcular la profundidad. La fuente de luz emite una señal modulada en fase, y el sensor PMD además de capturar la intensidad de la señal reflejada también es capaz de detectar su fase.

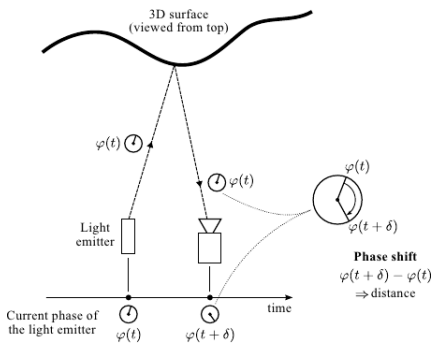

Figura 4.13: Cálculo de profundidad en cámaras TOF

[1] www.youtube.com/watch?v=HlyFiH2ZF9Y
[2] www.mariofrank.net/3dcameras.html

La señal de modulación se sincroniza entre la fuente de luz y el receptor, y posteriormente se calcula la distancia mediante la comparación de la fase de modulación, en el momento de la recepción, y la fase de la señal recibida. Este proceso aparece esquematizado en la figura 4.13. Aunque existen otros métodos para calcular profundidad con cámaras TOF, este es el que utilizan cámaras como las PMDTec mostrada en la figura 4.14 o SwissRanger.

Figura 4.14: Cámara TOF modelo CamCube3.0 de PMDTec.

Una visión detallada del funcionamiento de las cámaras TOF que utilizan PMD puede encontrarse en [19].

Algunas de las ventajas que presenta un sistema TOF son su velocidad, que les permite manejar hasta 100 fps, o su resolución, más alta que con el sistema estereoscópico.

Aun así, aunque las cámaras TOF son capaces de generar mapas de profundidad en tiempo real, aun tienen el inconveniente de ser demasiado caras. Esto es debido a que los tiempos que se manejan en los cálculos son muy pequeños, y hacen falta componentes electrónicos muy precisos y por ende, caros.

Sin embargo, con la llegada de la tecnología PrimeSense, se han podido fabricar cámaras 3D a precios mas asequibles, haciendo que adquirirlas sea más fácil para la gente y fomentando la investigación en el campo de la visión artificial con ellas.

4.4.3. PrimeSense

Con la tecnología PrimeSense se obtienen resultados similares a los de las cámaras TOF, pero la manera que tienen de calcular el mapa de profundidad es completamente distinto. Al igual que las TOF, constan de un emisor y un receptor. Sin embargo, el emisor proyecta una matriz de puntos IR con un patrón pseudoaleatorio (fig.4.15), propio de cada cámara y conocido por ella, y la distancia se calcula por medio de la deformación de ese patrón en la imagen capturada por el sensor.

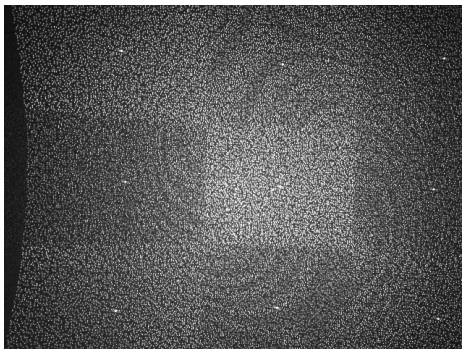

Figura 4.15: Imagen del sensor IR de una cámara PrimeSense donde se observa la matriz de puntos proyectada.

De esta manera, y tras una calibración inicial, sólo se necesita comparar la imagen del sensor con el patrón conocido, por lo que no es necesario métodos de correlación ni temporizadores electrónicos de alta precisión. Posteriormente se calculan las distancias por triangulación de forma similar a como se ha explicado antes.

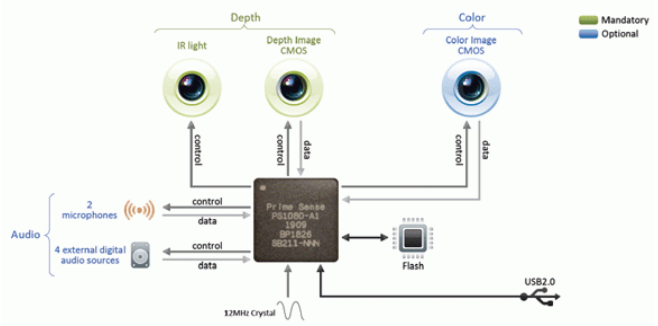

Figura 4.16: Esquema de funcionamiento de un dispositivo Primesense.

Otro aspecto a tener en cuenta es que el tamaño de los puntos depende de la distancia de los mismos al sensor, por lo que se pueden definir tres zonas de funcionamiento (fig.4.17):

1. Región 1: Esta primera zona comprende entre los $0{,}8m$ hasta $1{,}2m$ de distancia al sensor. En esta zona la precisión de las medidas es más alta, además de que se proyectan más puntos en el objeto, por lo que obtenemos más información de su forma y textura.

2. Región 2: La segunda zona abarca desde los $1{,}2m$ a los $2m$. Se aprecia una pérdida de resolución, aunque los datos siguen siendo aceptables. El error de precisión se encuentra en torno a 1cm en esta zona.

3. Región 3: La tercera región comienza a los $2m$ y llega hasta aproximadamente los $6m$. La pérdida de resolución es significativa y aunque aun se distinguen formas y superficies las medidas no son del todo fiables.

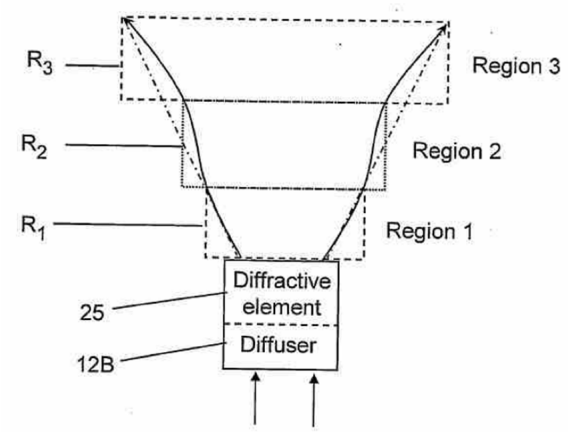

Figura 4.17: Zonas de funcionamiento de una cámara PrimeSense.

Uno de los problemas que tienen estas cámaras es que generan unas "sombras" en la imagen de profundidad debido a la distancia entre el emisor IR y la cámara IR. Debido a ello hay zonas de la escena que la cámara IR capta, pero debido a una oclusión con un objeto, la matriz de puntos del emisor no se proyecta en esa zona. Por lo tanto se pierde la información de la malla en ese área y aparece como píxeles negros (sin datos de profundidad) en la imagen que proporciona la cámara. El proceso que crea estas sombras puede verse gráficamente en la figura 4.18.

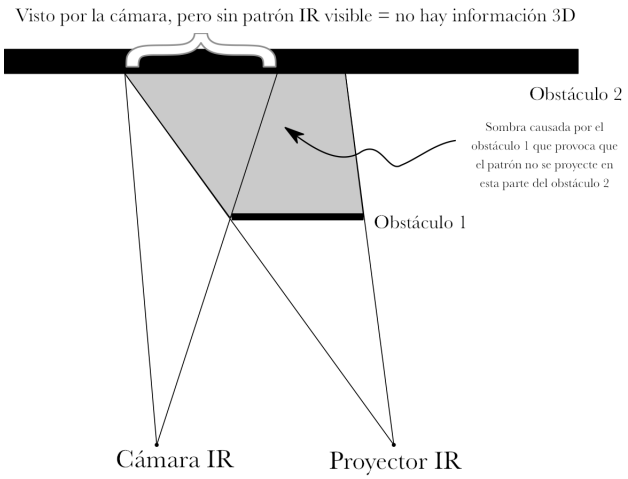

Figura 4.18: Explicación de las sombras en la imagen de profundidad de Kinect.

Y en la figura 4.19 puede verse una imagen de profundidad donde se aprecian estas sombras. También hay que mencionar que debido al alcance de la cámara, unos $6m$, también aparecen negros los píxeles pertenecientes a zonas que están demasiado lejos para ser captadas.

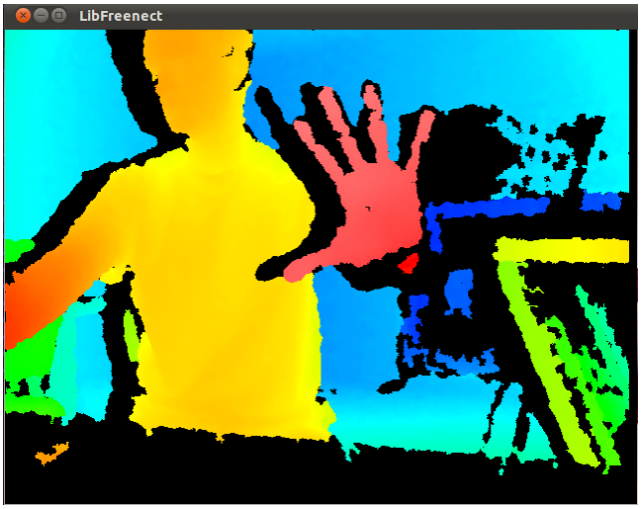

Figura 4.19: Imagen de profundidad de Kinect donde se observan los píxeles negros creados tanto por sombras como por distancia excesiva a la cámara.

4.5. Procesamiento de imagen

Una vez obtenidas las imágenes, ya sean de color en 2D, de profundidad o ambas, el siguiente paso es procesarlas para obtener la información necesaria para nuestra aplicación. Para entender la manera de aplicar los algoritmos a las imágenes primero debemos preguntarnos qué es una imagen y que características tiene.

Una imagen digital no es más que una matriz de $n \times m$ en la que cada elemento (píxel), contiene información relativa a intensidad, color o profundidad generalmente. Los valores del píxel se suelen tomar como variables de 8 bits, por lo que normalmente su valor numérico oscila entre 0 y 255, aunque se pueden utilizar otros sistemas de numeración, por ejemplo de 0 a 1 o con cualquier otro tamaño en memoria (16 bits, 32 bits...). En la ecuación 4.2 se representa el modelo matemático de una imagen.

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & \cdots & f(0,n-1) \\ f(1,0) & f(1,1) & \cdots & f(1,n-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(m-1,0) & f(m-1,1) & \cdots & f(m-1,n-1) \end{pmatrix} \tag{4.2}$$

Al definir una imagen de esta manera reducimos el tratamiento de la misma a operaciones matemáticas tanto matriciales o con máscaras, como con los píxeles por separado.

En la siguiente sección hablaremos sobre la representación del color en las imágenes. A continuación haremos un repaso por los algoritmos de preprocesamiento básicos que se suelen utilizar para adecuar la imagen para el tratamiento posterior. Finalmente explicaremos el fundamento teórico de los algoritmos utilizados en el proyecto.

4.5.1. Espacios de color

Como hemos mencionado antes, las cámaras captan la información de color mediante tres canales independientes que representan la intensidad de los colores rojo, verde y azul (espacio RGB). Sin embargo esta no es la única forma de representar estos datos, y a veces se hace nece-

sario cambiar de espacio de color para facilitar la aplicación de algunos algoritmos o de cara a utilizarlos en hardware específico.

A continuación se muestra un resumen de algunos espacios de color en función de sus dimensiones:

- 1 dimensión: Escala de grises
- 2 dimensiones: Subespacio RGB
- 3 dimensiones: RGB, CMY, HSV, YUV, LUV, CIE XYZ, NCS, Lab
- 4 dimensiones: CMYK

Explicaremos brevemente tres de los espacios de color más importantes para visión artificial: escala de grises, RGB y HSV.

4.5.1.1. Escala de grises

Esta manera de representar la información sólo utiliza un canal de datos, por lo que a veces no se suele incluir entre los espacios de color, al no tener información cromática como tal.

Figura 4.20: Imagen a color a la izquierda y su transformación en escala de grises, también conocida como *blanco y negro*.

Por lo general se muestra la intensidad de los píxeles de la imagen, por lo que para transformar una imagen RGB en escala de grises basta con normalizar la suma de las tres componentes como se muestra en la ecuación 4.3.

$$x_p = \frac{R_p + G_p + B_p}{3} \tag{4.3}$$

Siendo x_p el valor en escala de grises del píxel p, y (R_p, G_p, B_p) la terna que representa el color en espacio RGB de ese píxel.

A veces si se quiere ser más exacto en cuanto a representar la luminancia de la escena en escala de grises, se suele aplicar una expansión gamma (eq. 4.4) a las componentes RGB.

$$(R_{out}, G_{out}, B_{out}) = (AR_{in}^{\gamma}, AG_{in}^{\gamma}, AB_{in}^{\gamma}) \quad \text{con A=1 y } \gamma > 1 \tag{4.4}$$

Hay que tener en cuenta también que al pasar una imagen de color a escala de grises se pierde una cantidad de información significativa. Sin embargo, cuando la información a representar depende de una sola variable este espacio de color es más adecuado. En nuestro caso, el mapa de profundidad con el que trabajamos se puede representar en escala de grises, ya que sólo depende de la distancia de los objetos a la cámara y no aporta ninguna información más.

4.5.1.2. RGB

Este es el espacio de color más utilizado en multitud de aplicaciones y en el cual las cámaras digitales captan los colores imitando el funcionamiento del ojo humano. Se basa en el principio de la síntesis aditiva, por el cual la mayoría de los colores pueden representarse mediante suma de 3 componentes: roja, verde y azul. De ahí viene el nombre, RGB (Red Green Blue).

Se representa mediante un modelo cartesiano contenido en un cubo que por comodidad tiene de lado 1 unidad. De este modo el color queda representado mediante un vector de la forma (R, G, B) como se puede observar en la figura 4.21.

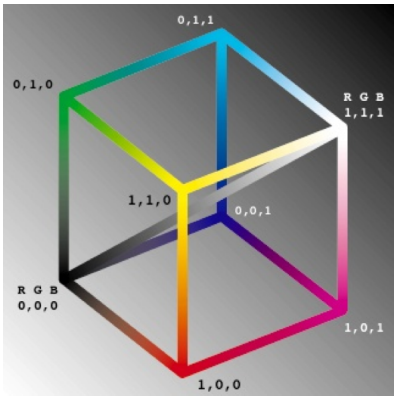

Figura 4.21: Espacio de color RGB en su representación cartesiana.

Los vértices del cubo corresponden con los colores rojo $(1, 0, 0)$, verde $(0, 1, 0)$, azul $(0, 0, 1)$, cian $(0, 1, 1)$, amarillo $(1, 1, 0)$, magenta $(1, 0, 1)$, blanco $(1, 1, 1)$ y negro $(0, 0, 0)$.

Algunas de las desventajas del modelo RGB radican en que la distribución de los colores no es uniforme e intuitiva y se hace difícil la asociación de los mismos. Además no distingue entre cromaticidad e intensidad, por lo que la variación de luminosidad en una imagen provoca variaciones en el vector RGB difíciles de cuantificar y manejar.

Por ello se adoptó un nuevo espacio de color mas apropiado para tareas de procesamiento y reconocimiento de color, el HSV.

4.5.1.3. HSV

El espacio de color HSV se basa en las coordenadas polares, donde el ángulo representa la cromaticidad o el tono, la distancia al centro representa la saturación del mismo y la altura la cantidad de intensidad luminosa. El modelo queda reflejado en la figura 4.22.

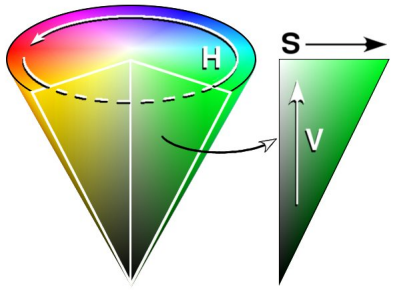

Figura 4.22: Representación tridimensional del espacio HSV.

Las ecuaciones para pasar del modelo RGB al HSV son las siguientes:

$$H = \begin{cases} \text{no definido} & \text{si } \max(R,G,B) = \min(R,G,B) \\ 60^o \times \frac{G-B}{\max(R,G,B)-\min(R,G,B)} + 0^o & \text{si } \max(R,G,B) = R \text{ y } G \geq B \\ 60^o \times \frac{G-B}{\max(R,G,B)-\min(R,G,B)} + 360^o & \text{si } \max(R,G,B) = R \text{ y } G < B \\ 60^o \times \frac{B-R}{\max(R,G,B)-\min(R,G,B)} + 120^o & \text{si } \max(R,G,B) = G \\ 60^o \times \frac{R-G}{\max(R,G,B)-\min(R,G,B)} + 240^o & \text{si } \max(R,G,B) = B \end{cases} \tag{4.5}$$

$$S = \begin{cases} 0 & \text{si } \max(R,G,B) = 0 \\ 1 - \frac{\min(R,G,B)}{\max(R,G,B)} & \text{en los demás casos} \end{cases} \tag{4.6}$$

$$V = \max(R,G,B) \tag{4.7}$$

Como se puede observar la transformación no es lineal, por lo que su ejecución es una de las desventajas de HSV. Además, cuando el color tiene poca saturación, es decir, se acerca al eje del cono, el modelo se vuelve muy inestable, llegando al límite cuando la saturación es nula ($\max(R,G,B) = \min(R,G,B)$) no pudiendo definir el tono (H) como se puede ver en la ecuación 4.5. Esto ocurre porque cualquier color, con una saturación nula o muy pequeña, se

convierte en gris independientemente de su tono.

Dejando esto de lado el espacio de color HSV proporciona en la mayoría de los casos una fuente de información estable del color y la iluminación de la imagen, permitiendo la identificación y seguimiento por color más fácilmente que con el espacio RGB.

4.5.2. Preprocesamiento de la imagen

Una de los primeros tratamientos que se suele aplicar a la imagen es el del preprocesamiento. Durante esta etapa se acondiciona la imagen y se filtran posibles ruidos o información no útil para nuestro algoritmo. En cambio, se puede resaltar la información que a nosotros nos sea útil como el color o la profundidad.

Otra de las operaciones que se le aplican a la imagen es el acondicionamiento de los datos observando el histograma de la imagen. De esta manera se pueden aplicar técnicas como el realce o manipulación de brillo o contraste, ecualización del histograma o realce en el dominio frecuencial.

Ya que la cantidad de algoritmos diferentes aplicados en visión artificial son demasiado numerosos como para explicarlos todos aquí, daremos un repaso por los algoritmos de preprocesamiento más importantes y básicos, mientras que los algoritmos utilizados propiamente en el trabajo serán explicados en la siguiente sección.

4.5.2.1. Realce frecuencial

En cuanto a los filtros que podemos aplicar a la imagen podemos separarlos entre paso-bajo, para eliminar ruido y suavizar la imagen, y paso-alto, para realce de bordes.

En cuanto a los filtros paso-bajo lo más común es utilizarlos para eliminar ruido. Los más básicos son los siguientes:

- Filtro de media: Al aplicar este filtro el valor del píxel se actualiza con la media de él

mismo con su entorno de vecindad (4 u 8).

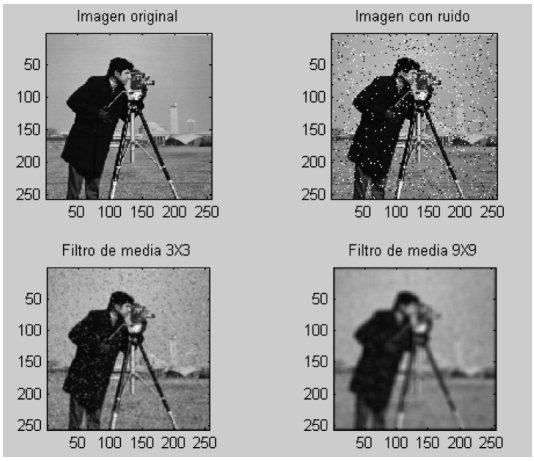

Figura 4.23: Ejemplo de aplicación de un filtro de media con diferentes tamaños de máscara.

- Filtro de mediana: Este filtro es el más adecuado para eliminar el ruido impulsional y se calcula con la mediana del píxel y sus vecinos.

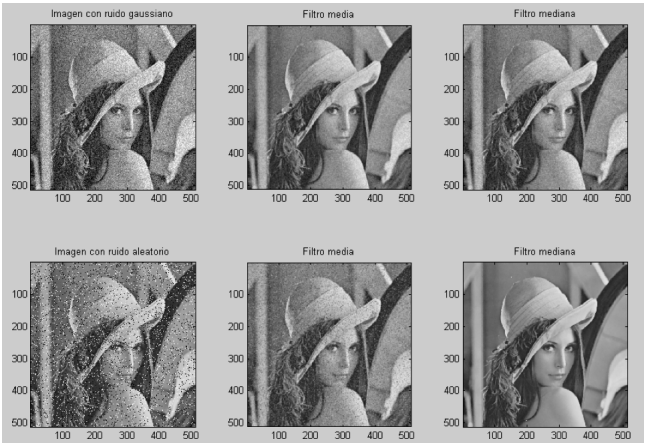

Figura 4.24: Ejemplo de aplicación de un filtro de mediana y comparación con uno de media.

- Filtro gaussiano: Otro tipo de filtro en el que se aplica el operador gaussiano al entorno de vecindad del píxel.

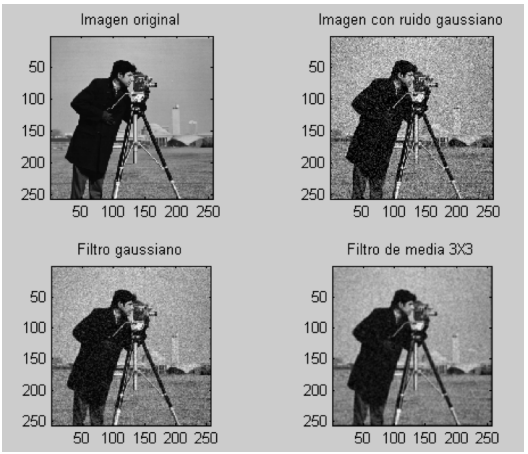

Figura 4.25: Ejemplo de aplicación de un filtro de gauss y comparación con uno de media.

Con respecto al realce de bordes se aplica una operación lineal a la imagen con un filtro paso alto. Esta operación está expresada en la ecuación 4.8.

$$I_{salida} = A \cdot I - F_{\text{paso-bajo}}(I) = (A-1) \cdot I + F_{\text{paso-alto}}(I) \tag{4.8}$$

I es la imagen, F el filtro (paso-alto o paso-bajo), y A es un escalar. Cuando $A > 1$ se obtiene un realce de las altas frecuencias, mientras que si $A = 1$ el resultado es el propio filtro paso-alto. Si $A < 1$ se obtiene el negativo de la imagen. El resultado de aplicar este realce puede verse en la figura 4.26.

Figura 4.26: Realce de las componentes de alta frecuencia de la imagen utilizando un filtro laplaciano.

4.5.2.2. Modificación de brillo y contraste

A veces nos encontramos con imágenes que tienen una falta o un exceso de brillo dependiendo de las condiciones de iluminación de la escena. Para ello se puede aplicar una operación muy sencilla de cara a obtener un brillo medio en la imagen. El brillo μ se define tal como aparece en la ecuación 4.9, y es igual al valor medio de todos los píxeles de la imagen.

$$\mu = \frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} f(x, y) \tag{4.9}$$

Siendo M el número de columnas de la imagen y N el número de filas.

Si la imagen es demasiado oscura se puede aplicar simplemente una suma a los píxeles. Si la imagen es demasiado luminosa se aplica una resta. En la ecuación 4.10 se muestra el proceso a seguir. Se ha supuesto el rango de los datos de 0 a 1. El valor de $N > 0$ implica aumentar el brillo, mientras que un valor $N < 0$ implica disminuirlo.

$$f_{procesada}(x, y) = \begin{cases} 1 & \text{si } (f(x, y) + N) > 1 \\ 0 & \text{si } (f(x, y) + N) < 0 \\ f(x, y) + N & \text{en los demás casos} \end{cases} \tag{4.10}$$

Estas operaciones se aplican a todos los canales de la imagen. Si es en escala de grises se aplicará solo al único canal, pero si es en color (RGB, HSV, etc...) la operación se aplica a cada canal independiente. Un ejemplo de esta operación se refleja en la figura 4.27

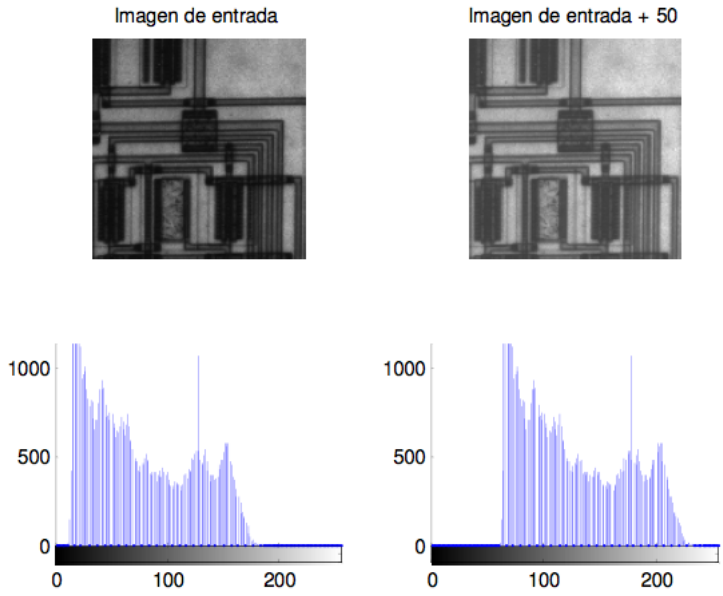

Figura 4.27: Realce del brillo en una imagen oscura con sus histogramas asociados.

El contraste en cambio indica la dispersión de los valores de los píxeles de una imagen. Cuando los valores de los píxeles no abarcan todo el rango dinámico que la imagen puede ofrecer, se aplica una corrección a los valores de acuerdo a la ecuación 4.11.

$$f_{procesada}(x,y) = \frac{\text{MAX}}{\text{máx}\, f(x,y)} f(x,y) \tag{4.11}$$

Siendo $\text{máx}\, f(x,y)$ el valor más alto de los píxeles de la imagen, y MAX el valor máximo que un píxel puede tomar, normalmente 255.

Un ejemplo de adecuación de rango dinámico se encuentra en la figura 4.28.

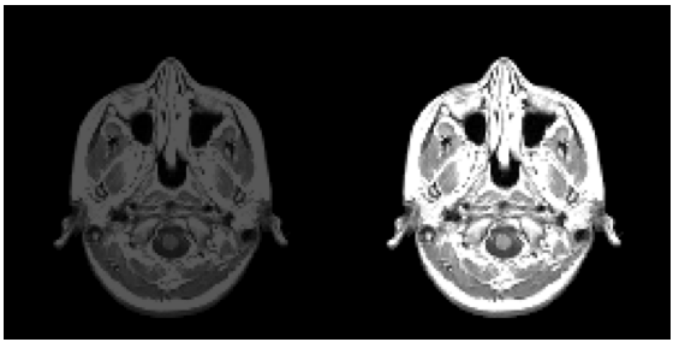

Figura 4.28: Aumento del contraste en una imagen médica.

4.5.2.3. Ecualización de histograma

Este método se utiliza para maximizar la explotación de la información. El proceso se basa en realizar una conversión en el histograma de forma que la probabilidad de encontrar cualquier nivel de gris en la imagen sea la misma. La ecuación aplicada a cada píxel para calcular su nuevo valor es la 4.12.

$$f_{procesada}(x,y) = round\left(\frac{cdf(f(x,y) - cdf_{min})}{(M \times N) - cdf_{min}} \times (L-1)\right) \tag{4.12}$$

Siendo $round$ la función de redondeo, $M \times N$ el tamaño de la imagen, y $cdf(x)$ la función de distribución acumulada para el valor x.

De esta forma se obtiene una imagen de salida como la de la figura 4.29.

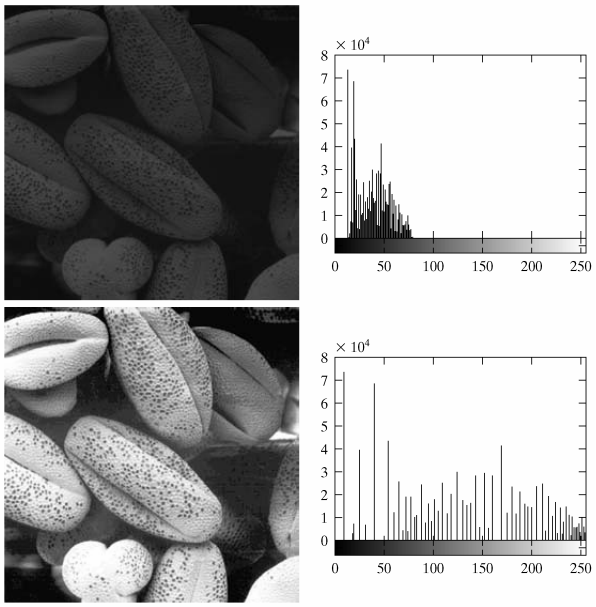

Figura 4.29: Ecualización de histograma.

Se puede observar que el histograma no queda totalmente plano, debido a que estamos trabajando con distribuciones discretas, por lo que los datos sufren cambios al redondear con respecto a si fuesen continuos.

4.6. Algoritmos utilizados

En esta sección se explicarán los aspectos teóricos de los algoritmos aplicados en el proyecto, mientras que los detalles de su aplicación se encuentran en el capítulo 6.

4.6.1. Umbralización

La umbralización es una técnica básica de segmentación que convierte una imagen en escala de grises o en color en una imagen binaria, de forma que se separa la información relevante de lo demás. La ecuación básica que se aplica en una umbralización es la 4.13. Esta ecuación generaliza el proceso de umbralización donde A y B son los valores extremos que tendrá la imagen binarizada y U es el valor umbral que se aplica.

$$f_{umb}(x,y) = \begin{cases} A & \text{si } f(x,y) \leq U \\ B & \text{si } f(x,y) > U \end{cases} \tag{4.13}$$

El parámetro U definido a priori es el más importante a la hora de obtener un buen resultado en la segmentación y existen varios métodos para elegirlo correctamente, como el método de *Otsu* basado en la forma del histograma, o el *k-mean clustering*, que es un algoritmo iterativo que elige un umbral automáticamente, aunque depende del valor inicial del mismo.

Cuando la imagen es a color además se puede aplicar una umbralización diferente a cada canal y posteriormente mediante operadores lógicos combinar los resultados para obtener la imagen binarizada. A esto se le llama umbralización multibanda.

4.6.2. Extracción de fondo

El método de extracción de fondo utilizado en este proyecto se basa en la media móvil. La ecuación que se aplica a la imagen para actualizar el fondo de la escena es la 4.14.

$$B_k = (1-\alpha)B_{k-1} + \alpha I_k \tag{4.14}$$

Siendo B_k y B_{k-1} las imágenes del fondo en el frame actual y el anterior respectivamente, I_k es la imagen obtenida por la cámara en el frame actual, y α la tasa de actualización. Aplicando esta ecuación a cada uno de los píxeles de la escena, estos van almacenando la media de todos los valores obtenidos hasta ese momento, dándole un peso a cada dato dependiendo del valor de α.

Posteriormente la extracción de fondo propiamente dicha se calcula comparando la imagen almacenada del fondo con la imagen actual, obteniendo lo que se conoce como foreground. Se

pueden tener en cuenta diversos aspectos a la hora de aplicar la actualización al fondo, como la aplicación de una máscara basada en cálculos anteriores del foreground. De esta manera se puede determinar si para ese píxel se actualiza el fondo o no, evitando "contaminar" la imagen almacenada del fondo con elementos que a priori no pertenecen al mismo.

Otros métodos de extracción de fondo asocian los valores de los píxeles a distribuciones gaussianas de forma que se calculan las probabilidades de que el píxel pertenezca al fondo o no, evitando en cierta medida el ruido de la imagen. Además los píxeles se pueden asociar a más de una distribución, por lo que se solucionan problemas en imágenes en las que el fondo se mueve ligeramente, como en los movimientos de las ramas de los árboles.

En cualquier caso, la tasa de aprendizaje se debe elegir correctamente si se quiere hacer robusta la extracción de fondo ante cambios en la iluminación de la escena si se están usando datos de color para el algoritmo.

4.6.3. Erosión-Dilatación

Una vez obtenida una imagen binaria en la que se resalta el objeto que vamos a tratar, lo siguiente es mejorar y definir el objeto de cara a su extracción posterior de características. Además, erosionar y dilatar la imagen elimina el ruido que pudiese tener, si éste se manifiesta en puntos los suficientemente pequeños.

Un ejemplo gráfico de una erosión y una dilatación con un elemento estructural no regular se encuentra en la figura 4.30.

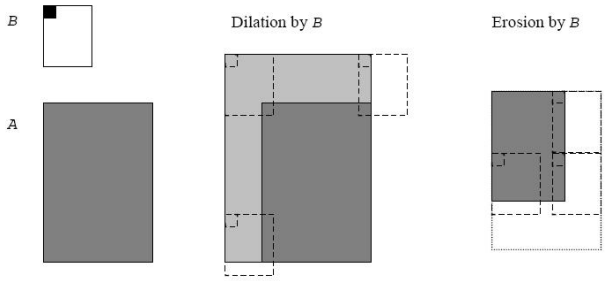

Figura 4.30: Proceso de dilatación y erosión de A con el elemento estructural B.

Los procesos de dilatación y erosión se definen de la siguiente manera:

- Dilatación: Esto significa que el píxel se encuentra en la imagen resultante si la intersección entre A y B es no nula.

$$A \oplus B = \{t : B_t \cap A \neq 0\}$$

- Erosión: Esto significa que el píxel se encuentra en la imagen resultante si B está contenido o es igual a A.

$$A \ominus B = \{t : B_t \subseteq A \neq 0\}$$

Cuando los procesos de dilatación y erosión se utilizan uno después de otro se habla de procesos de opening y closing:

- Opening: Erosión seguida de dilatación. Este proceso redondea los objetos y hace desaparecer los bordes afilados.

$$A \circ B = A \oplus (A \ominus B)$$

- Closing: Dilatación seguida de erosión. Este proceso hace desaparecer pequeños agujeros en los objetos e irregularidades del borde del mismo.

$$A \bullet B = A \ominus (A \oplus B)$$

4.6.4. ROI

Una ROI (Region Of Interest), es una zona, normalmente rectangular, de la imagen y es la que se utiliza para procesarse. De esta forma se elimina del procesado las partes de la imagen donde tenemos conocimiento de que no hay ninguna característica importante, mejorando la velocidad del procesado y disminuyendo su coste computacional.

La ROI se puede definir mediante figuras geométricas (rectángulos, círculos, elipses, etc.), de forma que para comprobar si un punto de la imagen está dentro de la ROI se comprueba si se encuentra dentro de la figura elegida utilizando sus parámetros. Otra forma de definir la ROI es mediante una máscara binaria. Se utiliza una imagen de las mismas dimensiones que la principal, y se ponen a 1 los píxeles pertenecientes a la ROI en esa imagen. De esta forma sólo hay que comprobar el píxel en la imagen auxiliar para saber si nos encontramos dentro de la ROI en la imagen principal.

4.6.5. Convex Hull

El Convex Hull o cerco convexo de un conjunto de puntos se define como el conjunto convexo más pequeño que contiene a todos los puntos. El símil gráfico para un convex hull en 2D sería el del polígono creado al poner una goma elástica alrededor de todos los puntos.

Existen varios algoritmos encargados de calcular el Convex Hull en conjuntos 2D entre los que destacan:

- Brute Force $O(n^4)$
- Gift Wrapping $O(nh)$
- Graham Scan $O(n \log n)$
- Jarvis March $O(nh)$
- Quickhull $O(nh)$
- Divide and Conquer $O(n \log n)$

- Monotone Chain $O(n \log n)$
- Incremental $O(n \log n)$
- Marriage before Conquest $O(n \log n)$

4.6.6. Curvatura de bordes

Para caracterizar las puntas de los dedos lo que haremos será realizar un estudio de la curvatura del borde de la mano. Ya que los dedos son estructuras estrechas, la punta de los mismos tendrá una curvatura lo suficientemente alta con respecto a los demás puntos del borde de la mano. Por eso mismo se calcula la curvatura de los puntos pertenecientes al borde de la mano según la ecuación 4.17.

Definimos b_n como el enésimo punto del borde, $\vec{A}$ y $\vec{B}$ son vectores calculados según la ecuación 4.15 y c_n es el valor de curvatura para b_n. La curvatura se calcula en el punto b_n con respecto a dos puntos situados l posiciones por delante y por detrás de b_n en el borde: b_{n-l} y b_{n+l}.

$$\vec{A} = \overrightarrow{b_{n-l}, b_n} \quad \vec{B} = \overrightarrow{b_{n+l}, b_n} \tag{4.15}$$

Una vez calculados los vectores, calculamos el seno y coseno del ángulo que forman. El coseno se utilizará para calcular el valor de curvatura en ese punto y el seno para saber si el ángulo es cóncavo o convexo.

$$\begin{aligned} \cos\alpha &= \frac{A_x \cdot B_x + A_y \cdot B_y}{||\vec{A}|| \cdot ||\vec{B}||} \\ \sin\alpha &= \frac{A_x \cdot B_y + A_y \cdot B_x}{||\vec{A}|| \cdot ||\vec{B}||} \end{aligned} \tag{4.16}$$

Y el valor de curvatura es:

$$c_n = \begin{cases} \frac{1}{2}(1+\cos\alpha) & \text{si } \sin\alpha \geq 0 \\ -\frac{1}{2}(1+\cos\alpha) & \text{si } \sin\alpha < 0 \end{cases} \tag{4.17}$$

4.6.7. Filtro de Kalman

El filtro de Kalman, también conocido como LQE (Linear Cuadratic Estimation) es un observador de estado óptimo capaz de predecir el estado de un sistema sometido a ruido gaussiano. Este algoritmo desarrollado por Rudolf E. Kalman en los 60 usa una serie de datos obtenidos temporalmente y que contienen ruido y otras inexactitudes para calcular el estado que tendrá el sistema de forma más precisa que si solo se utilizase una sola medida.

El algoritmo funciona en dos etapas. En la primera se utilizan los datos recopilados para generar una estimación del estado. En la segunda, y capturando el nuevo dato, se procede a una corrección del filtro, obteniendo una medida más exacta que la obtenida en ese momento.

Las ecuaciones que definen el sistema son la 4.18 y 4.19.

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k \tag{4.18}$$

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{4.19}$$

Donde:

- $\mathbf{x}_k$ y $\mathbf{x}_{k-1}$: Son los estados del sistema en los instantes k y $k-1$ respectivamente. En nuestro sistema están definidos por la posición, velocidad y aceleración del punto.

- $\mathbf{F}_k$: Es la matriz de transición de estado en el instante k. Está definida de antemano por el modelo del sistema a filtrar. También se denomina $\mathbf{A}_k$ a veces. En nuestro caso define la cinemática de los puntos (posición, velocidad y aceleración).
- $\mathbf{B}_k$: Es la Matriz de control en el instante k. Define las relaciones entre la entrada de control y el estado. En nuestro caso no se utiliza, ya que no hay entrada de control.
- $\mathbf{u}_k$: Es la entrada de control en el instante k. Inexistente en nuestro sistema.
- $\mathbf{w}_k$: Ruido gaussiano aplicado al estado. Tiene la estructura de una distribución normal multivariable de la forma $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$, donde $\mathbf{Q}_k$ es la covarianza.
- $\mathbf{z}_k$ y $\mathbf{z}_{k-1}$: Es el vector con las variables medidas u observaciones del sistema en los instantes k y $k-1$ respectivamente.
- $\mathbf{H}_k$: Es la matriz que define el modelo de las observaciones del sistema, que relaciona el estado y la medida, en el instante k.
- $\mathbf{v}_k$: Al igual que $\mathbf{w}_k$, se trata de ruido gaussiano aplicado a la hora de tomar las medidas en el instante k. Tiene la forma $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$, donde $\mathbf{R}_k$ es la covarianza.

Y las ecuaciones que definen el filtro de Kalman son 4.20 para la etapa de predicción y 4.21 para la de corrección.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \end{aligned} \tag{4.20}$$

$$\begin{aligned} \tilde{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \end{aligned} \tag{4.21}$$

Capítulo 5

Herramientas usadas

En este capítulo se especifican las características de las herramientas usadas en el proyecto. Está dividido en herramientas hardware y software.

5.1. Hardware

5.1.1. Kinect

Kinect es una cámara 3D de Microsoft con tecnología PrimeSense que salió al mercado en noviembre de 2010. Apareció como periférico de su consola Xbox360, pero debido a su relativo bajo precio con respecto a otras cámaras 3D y gracias a la ingeniería inversa con la que se pudo acceder a los datos generados por la cámara, se popularizó en los círculos de la investigación en visión artificial.

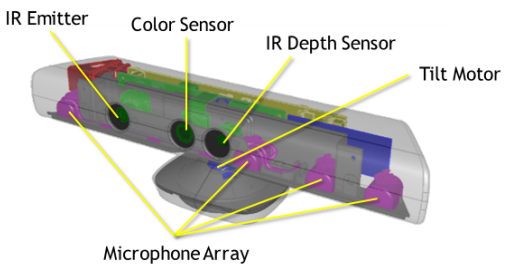

Figura 5.1: Cámara Kinect y partes de las que se compone.

Las características de Kinect son las siguientes:

- Cámara RGB con resolución de 640x480 a $30Hz$.
- Camara IR para calcular profundidad (11 bits) con resolución de 640x480 a $30Hz$.
- Rango de la cámara de profundidad: $0{,}5m$ - $6m$.
- Ángulo de visión: 57º horizontal, 43º vertical.
- Base motorizada en vertical de 27º a -27º.
- Array de 4 micrófonos a $16KHz$ y 24 bits.
- Led multicolor (rojo, verde, naranja) en el frontal de la cámara.
- Error en la resolución espacial: $3mm$ a $2m$ de distancia.
- Error en la resolución de profundidad: $1cm$ a $2m$ de distancia.

Además de Kinect existen en el mercado otras cámaras 3D con tecnología Primesense como XTionPro de ASUS o PSDK Reference. Estas cámaras son totalmente compatibles con el software utilizado en el proyecto, a excepción de que no poseen ni la base motorizada, ni el LED ni el array de micrófonos de Kinect.

5.1.2. MacBook Pro

Para el proyecto usaremos un portátil MacBook Pro modelo MacBookPro7.1 con las siguientes características:

- Nombre del procesador: Intel Core 2 Duo.
- Velocidad del procesador: 2.4 GHz.
- Caché de nivel 2: 3 MB.
- Memoria RAM: 4 GB, 2 bancos a 1067 MHz tipo DDR3.
- Tarjeta gráfica NVidia MCP89 AHCI.
- Disco duro TOSHIBA de 250 GB.
- Dos puertos USB 2.0.

5.2. Software

5.2.1. Ubuntu Linux 10.10

El sistema operativo elegido ha sido Ubuntu en su versión 10.10 *Maverick Meerkat*. Durante el desarrollo del trabajo no se ha actualizado el sistema operativo para asegurarnos de la funcionalidad de todas las herramientas. Los motivos por los que elegir este sistema operativo están bien explicados por el propio equipo de Ubuntu:

Ubuntu es un completo sistema operativo libre creado alrededor del núcleo Linux. La comunidad de Ubuntu gira alrededor de las ideas expresadas en la Filosofía Ubuntu: que el software debe estar disponible de forma gratuita, que las herramientas de software deben poder ser utilizadas por la gente en su idioma local, y que la gente debe tener la libertad de personalizar y alterar su software de la manera que necesiten. Por esos motivos:

- *Ubuntu siempre será gratuito y no tiene costes adicionales en la «enterprise edition»; hacemos accesible nuestro mejor trabajo a cualquiera en los mismos términos de gratuidad.*
- *Ubuntu usa lo mejor en infraestructura de traducciones y accesibilidad que la comunidad del software libre es capaz de ofrecer, para hacer que Ubuntu sea utilizable por el mayor número de personas posible.*
- *Ubuntu se publica se manera regular y predecible; se publica una nueva versión cada seis meses. Puede usar la versión estable actual o ayudar a mejorar la versión actualmente en desarrollo. Cada versión está soportada al menos durante 18 meses.*
- *Ubuntu está totalmente comprometido con los principios del desarrollo de software de código abierto; animamos a la gente a utilizar software de código abierto, a mejorarlo y a compartirlo.*

Para poder trabajar en Ubuntu con el portátil MacBook Pro, fue necesario la instalación de una herramienta para particionar el disco y correr Linux mediante una máquina virtual. El programa dedicado a ello es Parallels Desktop, en su versión 7. Debido al uso de una máquina virtual, el rendimiento obtenido no es del 100 % con respecto a las características del portátil funcionando en OSX.

5.2.2. Code::Blocks

Code::Blocks es el entorno de desarrollo integrado (IDE, Integrated Development Enviroment) utilizado para programar en Ubuntu en este trabajo. Code::Blocks es un IDE libre y multiplataforma que soporta diferentes compiladores entre otros GCC, MSVC o Borland C++. Está basado en la plataforma de interfaces gráficas *WxWidgets* y ostenta la licencia pública general de GNU.

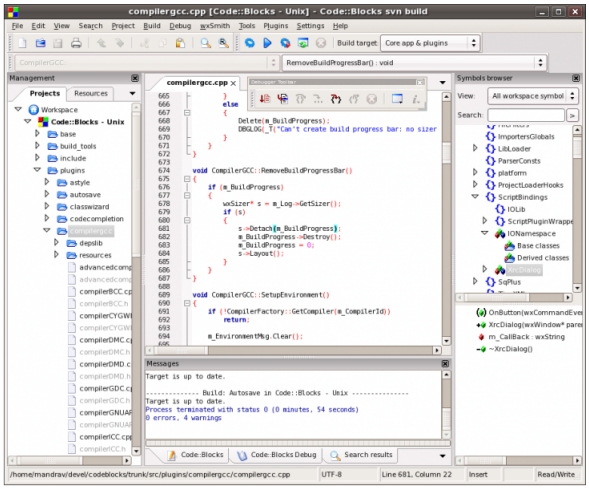

Figura 5.2: Interfaz gráfica de Code::Blocks

Code::Blocks está orientado a la programación en C y C++, y está disponible para Linux, Windows y Mac OS X. Además ha sido portado a FreeBSD. Por lo demás funciona como casi cualquier otro entorno de desarrollo y su interfaz es bastante intuitiva, como se puede observar en la figura 5.2.

5.2.3. Drivers de Kinect

Para los drivers a utilizar de cara a interactuar con la Kinect se barajó la posibilidad de utilizar OpenNI o Libfreenect.

5.2.3.1. OpenNI

En cuanto a OpenNI, se compone de una serie de API's que pretenden facilitar la creación de interfaces naturales. OpenNI está bajo Licencia Pública General GNU, y utiliza dispositivos de interacción natural para su funcionamiento, entre ellos Kinect. El objetivo de OpenNI es crear una API estándar que permita la comunicación entre sensores de vídeo y audio, y una serie de middleware creado para tareas de más alto nivel.

Los datos en bruto que es capaz de manejar OpenNI son los siguientes:

- Datos de color, sensor RGB.
- Datos de profundidad.
- Cámara de IR.
- Datos de audio, de un micrófono o array de micrófonos.

El middleware es lo que diferencia OpenNI de Libfreenect. Entre los diferentes módulos de middleware de OpenNI, los más importantes de cara al proyecto son el que proporciona la posición y orientación de las articulaciones del cuerpo mediante una calibración, y otro capaz de obtener un punto aproximado perteneciente a la mano al realizar el gesto de saludo.

Sin embargo, teniendo en cuenta el carácter académico del trabajo, se decidió utilizar Libfreenect por ser más básico y no utilizar middleware ya creado. De esta forma se resolverían los problemas que fueran surgiendo de manera personal de cara a una mejor comprensión de los algoritmos utilizados.

5.2.3.2. Libfreenect

Libfreenect es un software perteneciente al grupo OpenKinect, y que sirve como como driver a la hora de obtener la información de la cámara Kinect principalmente, aunque también funciona con las otras cámaras Primesensor, XTionPro de ASUS y PSDK Reference.

OpenKinect es una comunidad abierta de personas interesadas en hacer uso del hardware de Kinect en ordenadores y otros dispositivos. Trabajan en bibliotecas gratuitas de código abierto que permitan el uso de Kinect en Windows, Mac y Linux. El código perteneciente a OpenKinect tiene licencia Apache20 o GPL2.

En cuanto a las funcionalidades de Libfreenect, básicamente son las mismas que las de OpenNI a la hora de obtener datos. Libfreenect es capaz de manejar los píxeles de la cámara RGB como de la cámara infrarroja, y también obtener los datos de profundidad de 11 bits. Además de eso, también tiene métodos para calcular distancias reales en metros o para obtener el sonido de los micrófonos, cambiar el LED de colores o activar el motor instalado en la base de la Kinect.

En la figura 5.3 se puede observar que la única diferencia entre los datos que recopila OpenNI y Libfreenect es el tratamiento a la hora de mostrarlos por pantalla. Mientras que OpenNI los muestra mediante un sólo color, Libfreenect asigna diferentes colores a diferentes profundidades. Sin embargo este hecho es meramente estético, ya que los datos en sí son los mismos.

Figura 5.3: Imágenes obtenidas mediante OpenNI arriba y Libfreenect abajo.

5.2.4. OpenCV

OpenCV (Open source Computer Vision library), es un conjunto de bibliotecas para su uso en aplicaciones de visión artificial o por computador. Fue desarrollada por Intel y ahora la lleva Willow Garage y Itseez. Su código es libre y gratuito bajo la licencia BSD. Además, es multiplataforma y su lenguaje originario es C, por lo que existen versiones de OpenCV para Java, Python, Ruby, etc... Posteriormente OpenCV se ha actualizado a C++, lo que ha resuelto algunos problemas y ha mejorado la biblioteca en general.

Entre algunas de las aplicaciones para las que está diseñado OpenCV destacan las siguientes:

- Filtrado.
- Calibración.
- Segmentación.
- Seguimiento.
- Efectos visuales.

Se ha utilizado OpenCV ya que, aunque los datos a tratar están en 3D, la mayoría del procesado se hace sobre la imagen de profundidad o color binarizada, que está en escala de grises, por lo que se puede tratar como una imagen 2D.

5.2.5. OpenFrameworks

OpenFrameworks es un kit de herramientas con código abierto en C++ diseñado para ayudar al proceso creativo, proporcionando un marco sencillo e intuitivo para la experimentación. El conjunto de herramientas incorpora bibliotecas de uso común como:

- OpenGL, GLEW, GLUT, libtess2 y cairo para gráficos.
- rtAudio, PortAudio o FMod y KissFFT para entrada/salida de audio y procesamiento.
- FreeType para las fuentes.

- FreeImage para guardar/cargar imágenes.
- Quicktime y videoInput para reproducción y captura de vídeo.

El código está escrito para ser lo mas compatible posible, y en la actualidad soporta cinco sistemas operativos (Windows, OSX, Linux, iOS, Android) y cuatro IDEs (XCode, Code::Blocks, Visual Studio y Eclipse). La API está diseñada para ser compacta y fácil de entender.

OpenFrameworks está sostenido por una comunidad de personas que dedican su tiempo gratuitamente para su desarrollo. La licencia de OpenFrameworks es la MIT License, mientras que las bibliotecas que incorpora tienen sus propias licencias.

Se ha elegido utilizar OpenFrameworks por su estructura de aplicación similar a OpenGL (con funciones principales setup, update, draw, callbacks para teclado y ratón...) y por ser totalmente compatible con Code::Blocks. Además integra herramientas útiles de manejo de imágenes y de texturas a la hora de mostrar por pantalla y se le pueden aplicar añadidos (addons), entre los que está el de OpenCV (ofxOpenCV).

Capítulo 6

Experimentación

Este capítulo está dedicado a discutir las soluciones que hemos implementado en el proyecto.

6.1. Estructura de la aplicación

Al utilizar OpenFrameworks, la estructura de la aplicación va a tener un archivo `main.cpp` principal, donde se va a lanzar la aplicación del tipo `testApp` y se define la ventana donde va a trabajar. Una vez hecho eso, todo el programa se divide en cinco estructuras básicas localizadas en `testApp.cpp` donde iremos implementando las funcionalidades. Estas funciones principales son muy parecidas a las que encontramos en el entorno de OpenGL y son las siguientes:

1. `setup()`: Función que se ejecuta una sola vez al unicio del programa. Sirve para configurar todas las herramientas que vayamos a utilizar, además de reservar memoria para imágenes, establecer la conexión con la Kinect y demás.

2. `update()`: Esta función se ejecuta al principio de cada ciclo del programa. Junto con `draw`, son las funciones que se ejecutan en bucle una y otra vez hasta que la aplicación termina. En esta función aplicamos todo el procesado a las imágenes y los algoritmos necesarios.

3. `draw()`: La otra función que se ejecuta en cada ciclo de programa, después de `update`.

En esta función se le dice al programa qué mostrar por pantalla y donde.

4. "Callbacks" para teclado y ratón: Son funciones que se ejecutan cuando ocurre algún evento, como pulsar una tecla o pulsar/mover el ratón. Algunas de ellas son: `keyPressed(key)` para pulsación de teclado, `mousePressed(x, y, button)` y `mouseReleased(x, y, button)` para pulsaciones de ratón y `mouseDragged(x, y, button)` para su movimiento. Donde `key` es la tecla pulsada, (x, y) la posición del ratón y `button` el botón pulsado del ratón (derecho, izquierdo, central...).

5. `exit()`: Esta función se ejecuta al finalizar la aplicación. En ella cerramos la conexión con la cámara y liberamos el espacio utilizado.

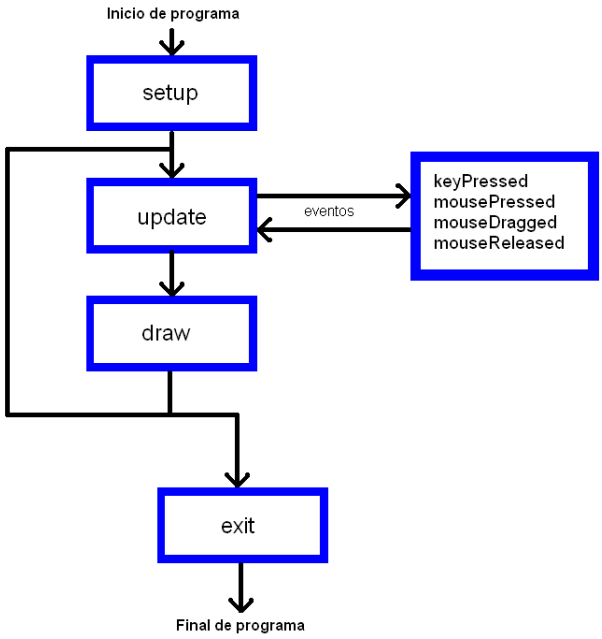

Figura 6.1: Diagrama de flujo de una aplicación en OpenFrameworks.

Una vez definidas las funciones principales de la aplicación se puede empezar a programar qué queremos que se haga en cada momento. Lo primero será abrir una conexión con Kinect y obtener/mostrar las imágenes recibidas.

6.2. Obtención de las imágenes

Para manejar la kinect mediante OpenFrameworks se utilizará un addon del mismo llamado ofxKinect. Este añadido está basado en Libfreenect y es capaz de manejar la información de la cámara mediante métodos entre los que podemos destacar:

- `init`: Esta función inicializa la Kinect de forma que se puede elegir si activar o desactivar la cámara rgb para aumentar la velocidad en caso de que no la necesitemos. Además puede cambiar la imagen rgb por la de la cámara de infrarrojos.
- `setRegistration`: Permite definir si se quiere calibrar las cámaras rgb y de profundidad entre sí para que concuerden sus píxels.
- `close`: Cierra la conexión con la Kinect y deja de tomar imágenes.
- `update`: Actualiza la imagen capturada por Kinect. Esta imagen puede ser nueva o no. Esto se comprueba con la siguiente función.
- `isFrameNew`: Permite saber si la imagen se ha actualizado con un nuevo frame o no. De esta forma sólo se ejecuta el procesamiento cuando hay un nuevo frame.
- Funciones "get" para distancias: Estas funciones nos devuelven los datos de distancia de diversas formas (en bruto 11bits, con respecto al centro de la imagen, en mm o en escala de grises).
- Funciones para manejar el motor de la base (definir u obtener el ángulo) y el LED (cambiar de colores o apagar).
- `getPixels` y `getDepthPixels`: Sirven para obtener el array de píxeles de las imágenes rgb y de profundidad respectivamente.
- `draw` y `drawDepth`: Estas funciones son las encargadas de dibujar las texturas de las imágenes rgb y de profundidad respectivamente por pantalla.

Gracias a estas funciones y algunas más que no hemos nombrado podemos comunicarnos con la Kinect para obtener sus imágenes rgb, IR o de profundidad; distancias de múltiples tipos, manejar su motor y su LED y mostrar por pantalla las imágenes obtenidas.

El siguiente paso es aplicar los algoritmos para obtener la mano y discriminar el fondo y posteriormente obtener características.

6.3. Segmentación

Para segmentar la mano debemos saber las características que va a tener nuestra escena. Ya que el objetivo es enseñar al robot MANFRED, la escena va a ser un entorno controlado en el que la cámara enfoca a una mesa con objetos y una persona los manipula. Por lo tanto, la persona no va a aparecer entera en la imagen, solo su brazo, por lo que las soluciones basadas en extraer el esqueleto no se van a utilizar.

En su lugar se obtuvo la segmentación de la mano mediante tres métodos, hasta quedarse con el último como definitivo.

Posteriormente se eliminó ruido y se mejoró la segmentación aplicando los algoritmos de erosión y dilatación a la imagen obtenida.

6.3.1. Umbralización por profundidad

Mediante este primer método lo que se hace es utilizar la imagen de profundidad, que está en escala de grises, y umbralizarla con dos límites T_1 y T_2, uno por debajo y otro por encima. Esta operación se muestra en la ecuación 6.1.

$$f_{umb}(x,y) = \begin{cases} 255 & \text{si } T_1 \leq f(x,y) < T_2 \\ 0 & \text{en los demás casos} \end{cases} \tag{6.1}$$

Siendo T_1 y T_2 los límites inferior y superior respectivamente, $f_{umb}(x,y)$ la imagen binarizada tras aplicar la umbralización, y $f(x,y)$ la imagen antes de procesarse.

El sentido de esta umbralización es el de quedarnos solamente con los píxeles que queden entre los dos valores. Como los valores representan la distancia a la cámara, lo que estamos haciendo es discriminar todos los puntos que no se encuentren dentro de unos límites de distancia definidos por el usuario.

En este sentido, la primera versión del algoritmo realizaba una umbralización estática con los valores definidos por el usuario. Estos valores se podían modificar en tiempo de ejecución con el teclado. La segunda versión implementada realizaba la umbralización de manera dinámica sin necesidad de definir los límites.

6.3.1.1. Umbralización estática

En esta primera versión se definen manualmente los límites T_1 y T_2 y se aplica la ecuación 6.1 a la imagen. Un esquema gráfico del proceso puede verse en la figura 6.2.

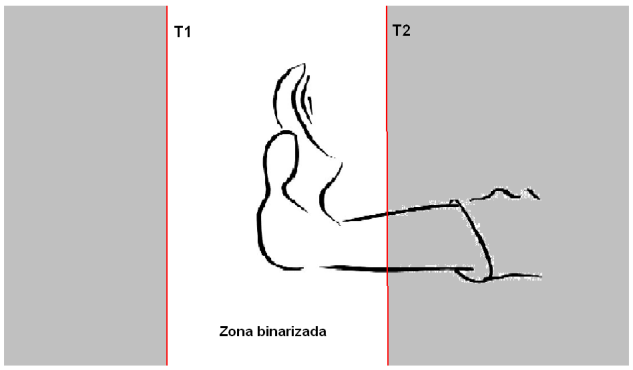

Figura 6.2: Esquema de la umbralización estática.

En las figuras 6.3 y 6.4 se muestra el resultado de aplicar este método con dos valores diferentes para cada límite.

Figura 6.3: Proceso de umbralización estática con $T_1 = 220$ y $T_2 = 255$.

Figura 6.4: Proceso de umbralización estática con $T_1 = 190$ y $T_2 = 255$.

6.3.1.2. Umbralización dinámica

Para la umbralización dinámica lo que se pretendía conseguir era no tener que definir los límites de manera manual de forma que estos cambiasen dinámicamente para adecuarse al movimiento de la mano. La solución consistió en definir los límites en cada frame de la forma expresada en la ecuación 6.2.

$$T_1 = \text{mín}\, f(x, y) \quad T_2 = T_1 + N \tag{6.2}$$

De esta forma, se conseguía que la sección de profundidad que se obtenía empezase en la parte mas cercana de la mano y terminase al final de ésta. Para que siempre se segmentase la mano entera se utilizó un valor de N igual a la distancia de la mano desde la muñeca hasta la punta del dedo corazón, de forma que si la mano aparecía en horizontal extendida hacia la cámara los límites T_1 y T_2 abarcaran toda ella. Un esquema del proceso se encuentra en la figura 6.5.

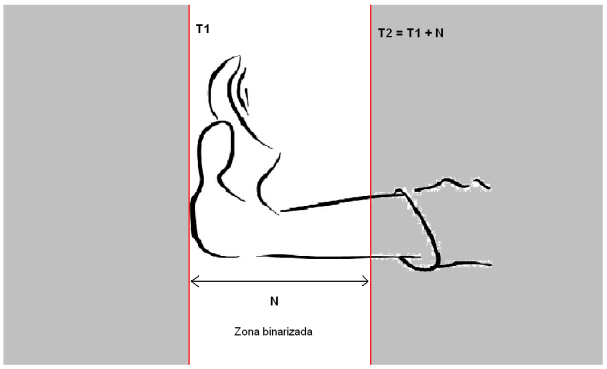

Figura 6.5: Esquema de la umbralización dinámica.

En la figura 6.6 se puede observar el resultado de aplicar este método. El valor de N en este caso es 15.

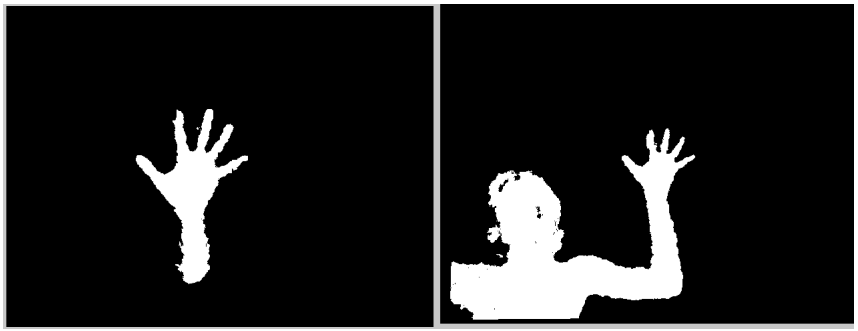

Figura 6.6: Proceso de umbralización dinámica. A la izquierda con la mano extendida y a la derecha con la mano a la misma profundidad que el resto del cuerpo.

Uno de los problemas de la umbralización es que si existía un objeto, cualquiera que fuese, más cerca de la cámara que la mano, el objeto era segmentado en su lugar. En la figura 6.7 se observa este hecho. Para evitar que se segmentasen objetos que pertenecen al fondo (background) de la escena se cambió la umbralización por una extracción de fondo.

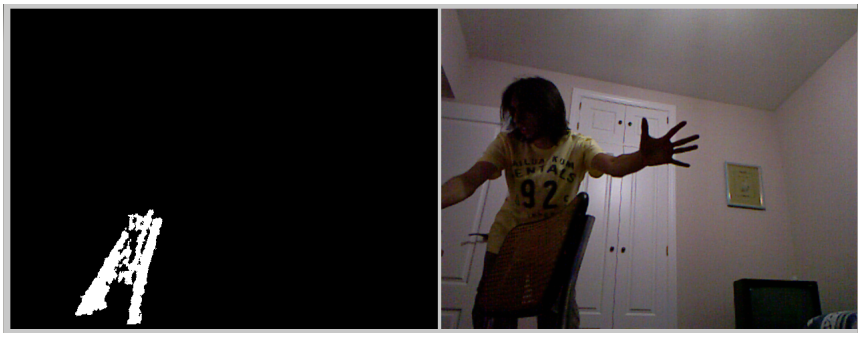

Figura 6.7: Umbralización incorrecta de un objeto en lugar de la mano. En este caso, la silla está más cerca de la cámara que la mano.

6.3.2. Extracción de fondo

Esta forma de segmentar utiliza una sucesión de imágenes para crear una imagen de fondo o background según la ecuación 4.14, donde el valor de α escogido es de $0{,}01$.

Una vez calculado el fondo o background en el frame actual, se pasa a calcular el foreground, es decir, aquellos objetos que no pertenecen al fondo comparando la imagen I_k con el fondo en ese frame, B_k calculado anteriormente, aplicando la ecuación 6.3.

$$F(x,y) = \begin{cases} 255 & \text{si } |I_k(x,y) - B_k(x,y)| \geq T \\ 0 & \text{en los demás casos} \end{cases} \tag{6.3}$$

De esta forma en la imagen resultante, $F(x,y)$, aparecen binarizados los elementos que no pertenecen al fondo. En nuestro caso este elemento será la mano, como se puede observar en la

imagen 6.8.

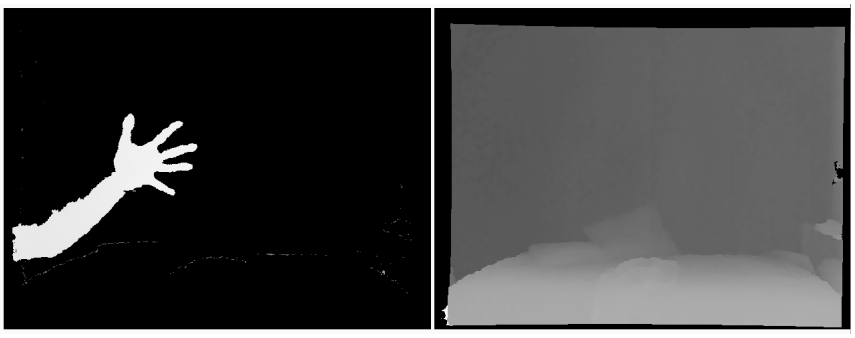

Figura 6.8: Extracción de fondo. Foreground a la izquierda y background a la derecha.

También hay que tener en cuenta que, a la hora de calcular el fondo, los píxeles sin datos de profundidad aparecen en negro, esto es, con un valor de 0. Este hecho hace que en ese píxel, la media calculada para el fondo baje rápidamente y se convierta en 0 también. Para evitarnos este problema añadimos al algoritmo una comprobación del píxel (x, y) que estamos calculando, tanto del fondo calculado anteriormente como de la imagen actual que recibimos de la cámara. De este modo tenemos cuatro posibles escenarios:

1. $B_{k-1}(x, y) \neq 0$ y $I_k(x, y) \neq 0$: En este caso se ejecuta la ecuación 4.14 de manera normal.
2. $B_{k-1}(x, y) \neq 0$ y $I_k(x, y) = 0$: En este caso hemos perdido información del fondo que antes sí teníamos. Para que la media calculada no baje debido al 0, se deja el background como estaba ($B_k(x, y) = B_{k-1}(x, y)$).
3. $B_{k-1}(x, y) = 0$ y $I_k(x, y) \neq 0$: En este caso tenemos información de un píxel que antes no teníamos, por lo que copiamos directamente esa información al nuevo background ($B_k(x, y) = I_k(x, y)$).
4. $B_{k-1}(x, y) = 0$ y $I_k(x, y) = 0$: En este caso no se ejecuta nada y se deja $B_k(x, y) = B_{k-1}(x, y) = 0$.

En cuanto al procesado de los objetos (foreground), tenemos que tener las mismas consideraciones que con el background. Aquí se vuelven a dar los mismos cuatro casos:

1. $B_k(x, y) \neq 0$ y $I_k(x, y) \neq 0$: Se ejecuta la ecuación 6.3 sobre el píxel de manera normal.
2. $B_k(x, y) \neq 0$ y $I_k(x, y) = 0$: No se ejecuta nada, ya que no existe $I_k(x, y)$ para comparar.
3. $B_k(x, y) = 0$ y $I_k(x, y) \neq 0$: Se muestra ese píxel como foreground.
4. $B_k(x, y) = 0$ y $I_k(x, y) = 0$: No se ejecuta nada, ya que no existe ni $I_k(x, y)$ ni $B_k(x, y)$ para comparar.

Gracias a estas consideraciones, el fondo de la escena se va actualizando a medida que la aplicación está activa y el cálculo del foreground es satisfactorio. El cálculo del fondo se inicia en el primer frame, o pulsando una tecla si hemos movido la cámara de lugar y queremos capturar el fondo de nuevo. En ambas situaciones se aplica la ecuación 6.4.

$$B_k = I_k \tag{6.4}$$

Sin embargo, existe aún un último problema. En el caso de que existan los píxeles tanto de $B_{k-1}(x, y)$ como de $I_k(x, y)$ (caso 1), el nuevo fondo ($B_k(x, y)$) se actualiza independientemente de que el píxel pertenezca al foreground o no.

Un primer acercamiento al problema fue utilizar una extracción de fondo dinámica, de forma que si existía algún objeto en escena, el valor de la tasa de aprendizaje α pasaba de 0.01 a 0.0001. De esta forma, el fondo no cambiaba de manera sustancial en presencia de objetos o foreground. Sin embargo, aunque de manera sutil, los píxeles del foreground se seguían utilizando para actualizar el fondo, por lo que se utilizó otra solución.

Para solucionar este problema, se calcula el foreground con respecto al fondo del frame anterior ($B_{k-1}(x, y)$), y posteriormente se añade una condición a la actualización del nuevo fondo. Si el píxel pertenece al foreground ($F(x, y) \neq 0$) se deja el píxel sin actualizar. Si no pertenece, se

ejecutan las instrucciones explicadas anteriormente. De esta forma se soluciona el problema sin necesidad de comprobar B_k dos veces, y además los píxeles pertenecientes al fondo se siguen actualizando aun en presencia de objetos.

6.3.3. Erosión-Dilatación

Los algoritmos de erosión y dilatación utilizados se encuentran en OpenCV, e implementan estas transformaciones morfológicas de forma que se puede aplicar a una imagen utilizando cualquier tipo de elemento estructural y con el numero de iteraciones deseadas.

En nuestro caso hemos aplicado a la imagen un opening de 3 erosiones seguidas de 3 dilataciones. De esta forma se elimina gran cantidad de puntos clasificados como ruido. Posteriormente se ha aplicado un closing de 2 dilataciones seguido de 2 erosiones. En los dos casos se ha utilizado un elemento estructural de forma circular. El resultado puede observarse en la figura 6.9.

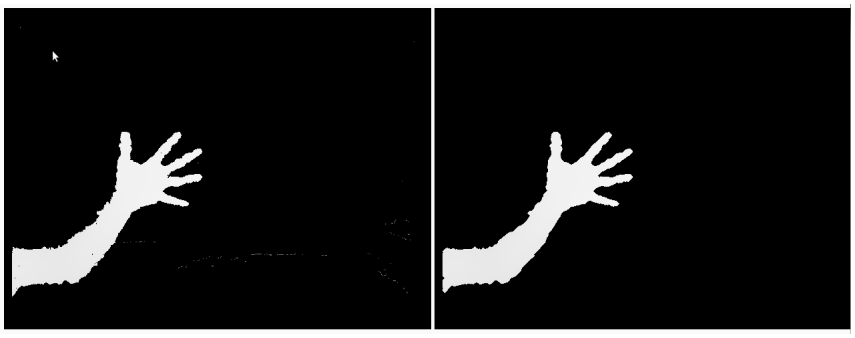

Figura 6.9: A la izquierda imagen de entrada. A la derecha tras aplicar las dilataciones y erosiones.

A partir de ahora ya tenemos la mano segmentada de una forma aceptable y con poco ruido, de forma que podemos pasar al siguiente bloque de procesado, la extracción de características.

6.3.4. ROI

Una mejora que se le aplicó al algoritmo a posteriori fue la inclusión de una ROI alrededor de la mano. En nuestro caso se ha definido la ROI como un cuadrado de 220x220 píxeles centrado en la palma de la mano. De esta forma, y tras encontrar por primera vez la palma en la imagen (explicado más adelante), se reducen el número de datos que utilizamos en el procesado siguiente. Podemos mencionar también que los parámetros de la ROI (ancho y alto), se adaptan dinámicamente de forma que la ROI no abarca espacio fuera de la imagen.

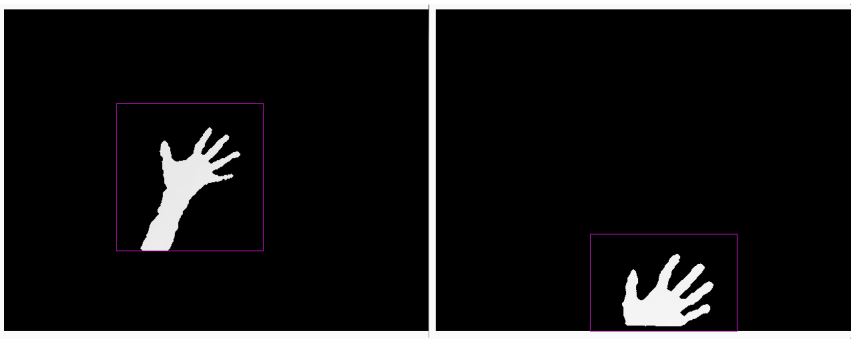

Figura 6.10: ROI's definidas alrededor de la palma de la mano. En la imagen de la derecha se puede apreciar cómo la ROI se adapta a las dimensiones de la imagen.

En la imagen 6.10 se puede observar la ROI y su efecto en la segmentación.

6.4. Extracción de características

Una vez que tenemos identificada la parte de la imagen que vamos a procesar, lo primero que debemos hacer es centrarnos en el objeto principal y extraer su contorno, sobre el que realizaremos las operaciones explicadas a continuación.

6.4.1. Identificación del blob y su contorno

Para la realización de esta parte se ha utilizado una biblioteca para OpenFrameworks llamada ofxContourFinder. Esta biblioteca forma parte del paquete ofxOpenCV que integra fun-

ciones de OpenCV optimizadas para trabajar con las variables de OpenFrameworks. Gracias a esta biblioteca se recorre la imagen binaria y se obtienen los contornos de los objetos o blobs.

La función principal que se utiliza tiene los siguientes parámetros ajustables:

- `minArea` y `maxArea`: Áreas mínima y máxima permitida medida en píxeles cuadrados para los blobs. En nuestro caso están definidas como 100 y 10000 respectivamente.

- `nConsidered`: Numero de blobs máximo a calcular. En nuestro caso solo necesitamos el blob de la mano, por lo que su valor es de 1.

- `bFindHoles`: Permite definir si los agujeros de los objetos se toman como bordes interiores o no. En nuestro caso está desactivado.

- `bUseAproximation`: Hace que se utilicen menos puntos para el borde de manera que se pueden ahorrar puntos en una línea por ejemplo, ya que se puede definir con sólo dos puntos. En nuestro caso está activado.

En la figura 6.11 se puede ver el resultado de aplicar este proceso a la imagen. Además del contorno, ofxContourFinder permite obtener también el mínimo rectángulo que engloba el blob y el centroide del mismo.

Figura 6.11: Aplicación de ofxContourFinder a la imagen segmentada.

6.4.2. Palma de la mano

Para obtener la posición de la palma de la mano se probó primeramente asignándola al centroide del blob. El problema que se tenía era que cuando se colaba parte del antebrazo en la segmentación del blob, el centroide del mismo modificaba bastante su posición. Este problema se puede ver en la figura 6.12.

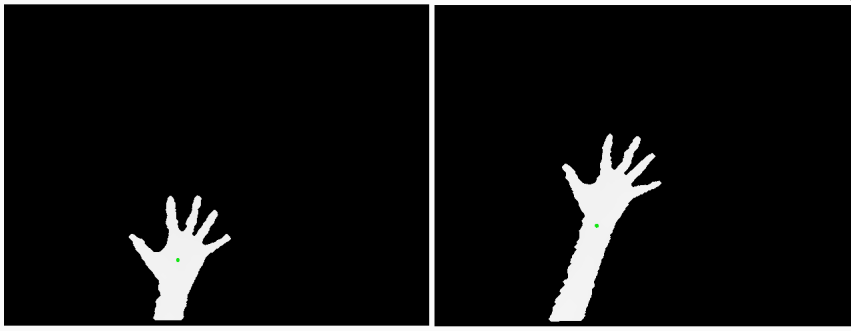

Figura 6.12: Variación de posición en el centroide sin mostrar el antebrazo y mostrándolo.

La solución a este problema está en definir la palma como el centro de la circunferencia de mayor tamaño que cabe dentro del blob. De este modo la variación en la posición de la palma no es tan brusca cuando aparece el antebrazo.

Para lograr obtener este punto en primer lugar se pensó en obtener el diagrama de Voronoi del polígono definido por el contorno del blob. Después, para cada nodo de Voronoi se busca el que tenga la mayor distancia al borde del blob. Sin embargo, esta solución requería de la utilización de más librerías para calcular el diagrama de Voronoi, con el consecuente coste computacional.

Una solución alternativa se basó en hacer una pasada por la imagen y, para los puntos pertenecientes al objeto, calcular la distancia a todos los bordes del mismo y quedarse con la menor. Posteriormente, de entre todas las menores distancias, nos quedamos con la mayor de todas. De esta forma, el punto resultante será el que esté más alejado de los bordes del blob y además pertenezca a él.

Como recorrer toda la imagen y calcular la distancia a todos los puntos requería un coste computacional alto, se decidió procesar 1 de cada 4 píxeles de la imagen y calcular las distancias a 1 de cada 4 puntos del borde del blob. De esta forma hacemos el algoritmo más rápido sin perder apenas precisión.

El resultado de la aplicación de este algoritmo puede verse en la figura 6.13.

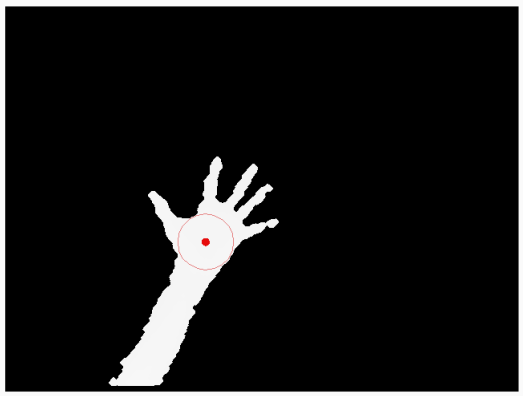

Figura 6.13: Obtención de la palma de la mano.

6.4.3. Cerco convexo (Convex Hull)

Una manera de hacer más robusto el cálculo de las puntas de los dedos, que es el siguiente proceso, es calcular el Convex Hull del contorno. De esta forma filtramos puntos donde es seguro que no se van a encontrar los dedos.

Para aplicar el Convex Hull se intentó primeramente aplicar el algoritmo "Monotone chain" mediante una implementación en C++ propia. Sin embargo, los resultados del algoritmo no fueron satisfactorios, como se puede comprobar en la figura 6.14.

Figura 6.14: Obtención del Convex Hull incorrecta mediante el primer método.

El siguiente intento pasó por utilizar una implementación del algoritmo de "Graham scan". Pero el algoritmo tampoco arrojaba buenos resultados, tal como se puede ver en la figura 6.15.

Figura 6.15: Obtención del Convex Hull incorrecta mediante el segundo método.

La tercera solución fue la única que arrojó buenos resultados. En este caso utilizamos la función `cvConvexHull2` de OpenCV directamente y no del paquete ofxOpenCV, por lo que hubo que adecuar los datos de OpenFrameworks a variables de OpenCV y viceversa. El resultado, ya satisfactorio, de la aplicación de `cvConvexHull2` se observa en la figura 6.16.

Figura 6.16: Obtención del Convex Hull mediante el tercer método con OpenCV.

6.4.4. Puntas de los dedos

Utilizando el valor para la curvatura de la ecuación 4.17 obtenemos un perfil de curvaturas para el borde de la mano de la forma que aparece en la figura 6.17. El valor de n utilizado en la ecuacion 4.15 para calcular los vectores $\vec{A}$ y $\vec{B}$ ha sido de 10. Una vez tenemos los valores de curvatura se definen como puntas de los dedos aquellos puntos que tengan un valor de curvatura mayor que uno definido manualmente y además sean un máximo local en esa zona. El valor de curvatura mínimo está definido en nuestro caso en 0.56. Este valor mínimo aparece representado en la figura 6.17 como una línea verde en horizontal.

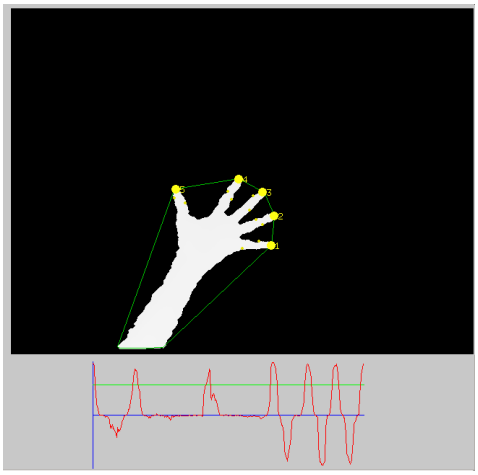

Figura 6.17: Representación de los valores de curvatura para el blob mostrado. Se observan los puntos pertenecientes a los dedos como máximos y los puntos entre ellos como mínimos.

Hay que mencionar que, gracias a la utilización del Convex Hull no es necesario calcular la curvatura de todos los puntos del borde, solo de aquellos pertenecientes al Convex Hull, ahorrando coste computacional.

Además de utilizar las ecuaciones vistas hasta ahora para calcular los dedos, se añadieron una serie de limitaciones a la hora de decidir si un punto con la suficiente curvatura es un dedo o no. Junto con las explicadas anteriormente, las limitaciones usadas para clasificar puntos como dedos son las siguientes:

- Convex Hull: Para que un punto sea considerado dedo, debe pertenecer al Convex Hull.

- Curvatura: El punto debe tener una curvatura, calculada mediante la ecuación 4.17, mayor a un valor predefinido.

- Máximo local: El punto debe ser un máximo local entre los puntos de alrededor con respecto a su curvatura.

- Distancia a la palma: Un punto sólo puede ser un dedo si su distancia a la palma de la mano no supera 4 veces el radio de la misma. De esta forma se eliminaban puntos clasificados como dedos en zonas como el codo, suficientemente alejadas de la palma. Sin embargo, gracias a la utilización de una ROI, esta comprobación no filtra tantos puntos como no utilizándola, pero sigue siendo útil.

- Posición extrema: El punto clasificado como dedo no debe encontrarse en los límites de la ROI definida alrededor de la mano. Gracias a esta comprobación se filtran puntos clasificados como dedos que se encontraban en la muñeca.

Para distinguir cada dedo se aplica una calibración con la mano en vertical y suponiendo que se utiliza la izquierda. De esta manera se etiquetan los dedos tal como aparece en la figura 6.18.

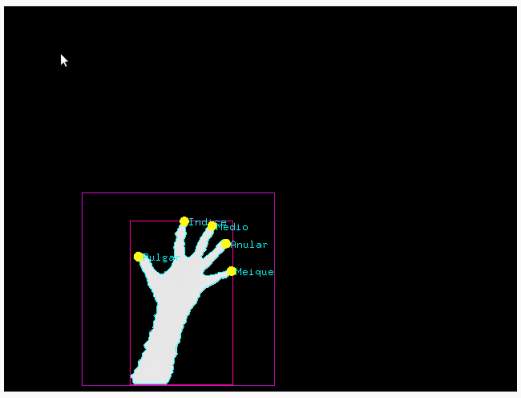

Figura 6.18: Identificación de cada uno de los dedos de la mano izquierda.

6.4.5. Otras características

Además de las características principales explicadas anteriormente, se han utilizado otras de las que destacamos las siguientes:

6.4.5.1. Min Enclosing Circle

Esto es el mínimo círculo que engloba el blob. Se puede observar en la figura 6.19. Se añadió por si su centro podía funcionar como palma, pero al final no se ha utilizado, aunque como su coste computacional es bajo, se puede mostrar si se desea. Se ha implementado utilizando la función de OpenCV `cvMinEnclosingCircle`.

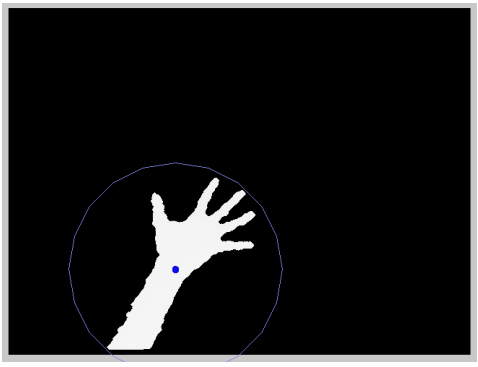

Figura 6.19: Mínimo circulo que engloba el blob y su centro, en azul.

6.4.5.2. Dirección de los dedos

También se ha calculado la dirección a la que apuntan los dedos utilizando un vector $\vec{D}$ obtenido a partir de los puntos utilizados para calcular la curvatura, b_f, b_{f-l} y b_{f+l}, siendo b el conjunto de puntos del borde, y f el índice del punto perteneciente a la punta del dedo cuya dirección queremos calcular. El proceso es el siguiente:

Para facilitar la nomenclatura llamemos A a b_{f-l}, B a b_{f+l} y F a b_f. El vector $\vec{D}$ entonces se calcula según la ecuación 6.5.

$$\vec{D} = (F_x - (\frac{A_x + B_x}{2}), F_y - (\frac{A_y + B_y}{2})) \tag{6.5}$$

Otra forma de calcular este vector, menos precisa, es utilizar el punto perteneciente a la palma, llamémosle P, junto con la punta del dedo F. De esta manera el vector se calcularía según la ecuación 6.6.

$$\vec{D} = (F_x - P_x, F_y - P_y) \tag{6.6}$$

Por último, si se quiere utilizar el vector en 3D en lugar de 2D, solo hay que añadir la componente z a las ecuaciones 6.5 y 6.6, obteniendo lo siguiente:

$$\begin{aligned} \vec{D} = (F_x - (\frac{A_x + B_x}{2}), F_y - (\frac{A_y + B_y}{2}), F_z - (\frac{A_z + B_z}{2})) \\ \vec{D} = (F_x - P_x, F_y - P_y, F_z - P_z) \end{aligned} \tag{6.7}$$

En la figura 6.20 se muestra una imagen con la dirección de los dedos siguiendo el primer método.

Figura 6.20: Dirección de los dedos en verde y puntos para calcularla en amarillo, utilizando la ecuación 6.5.

Y en la figura 6.21 siguiendo el segundo método.

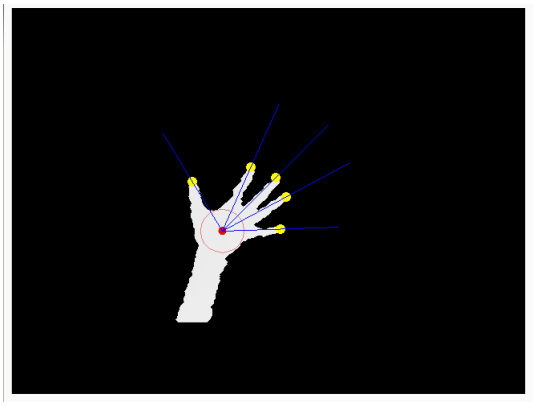

Figura 6.21: Dirección de los dedos utilizando la ecuación 6.6.

6.4.5.3. Rectángulo circunscrito

Por último hemos utilizado también la función de OpenCV `cvMinAreaRect2`, que calcula el rectángulo de área mínima que engloba todo el blob. Este rectángulo se ha calculado de cara a obtener la orientación de la mano según el ángulo del mismo. La aplicación de este método se observa en la figura 6.22.

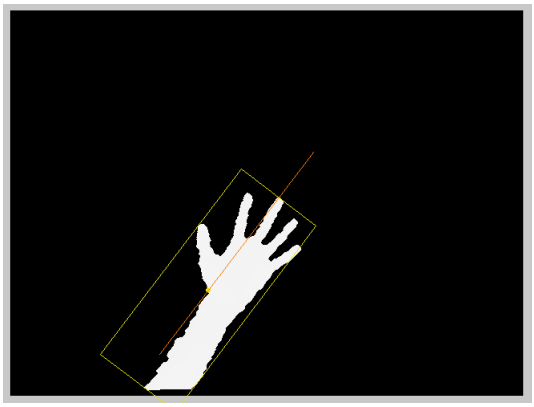

Figura 6.22: Rectángulo circunscrito en el blob en amarillo y orientación en naranja.

6.5. Seguimiento

Una vez que hemos calculado los puntos de interés de la mano (punta de los dedos y palma), tenemos que hacer un seguimiento de los mismos. Esta parte del procesado se denomina tracking, y usaremos un filtro de Kalman para llevarla a cabo.

6.5.1. Filtro de Kalman

Para aplicar el filtro de Kalman en nuestro trabajo debemos definir cinco filtros, uno para cada dedo, además de otro para la palma. Las características de los filtros son las siguientes:

$$\mathbf{x}_k = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \tag{6.8}$$

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.9}$$

$$\mathbf{z}_k = \begin{bmatrix} x \\ y \end{bmatrix} \tag{6.10}$$

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.11}$$

$$\mathbf{Q}_k = \begin{bmatrix} Q & 0 & 0 & 0 & 0 & 0 \\ 0 & Q & 0 & 0 & 0 & 0 \\ 0 & 0 & Q & 0 & 0 & 0 \\ 0 & 0 & 0 & Q & 0 & 0 \\ 0 & 0 & 0 & 0 & Q & 0 \\ 0 & 0 & 0 & 0 & 0 & Q \end{bmatrix} \tag{6.12}$$

$$\mathbf{R}_k = \begin{bmatrix} R & 0 & 0 & 0 & 0 & 0 \\ 0 & R & 0 & 0 & 0 & 0 \\ 0 & 0 & R & 0 & 0 & 0 \\ 0 & 0 & 0 & R & 0 & 0 \\ 0 & 0 & 0 & 0 & R & 0 \\ 0 & 0 & 0 & 0 & 0 & R \end{bmatrix} \tag{6.13}$$

Se puede observar que la matriz $\mathbf{F}_k$ representa las ecuaciones cinemáticas mostradas en 6.14.

$$\begin{aligned} \mathbf{x}_k &= \mathbf{x}_{k-1} + \mathbf{v}_{k-1}\Delta t + \mathbf{a}_{k-1}\frac{\Delta t^2}{2} \\ \mathbf{v}_k &= \mathbf{v}_{k-1} + \mathbf{a}_{k-1}\Delta t \\ \mathbf{a}_k &= \mathbf{a}_{k-1} \end{aligned} \tag{6.14}$$

Se han utilizado dos métodos a la hora de aplicar el filtro de Kalman. La primera manera de aplicar el filtro se hizo con la matriz $\mathbf{F}_k$ fija definiendo $\Delta t = 1$. La segunda implementación incorporó la actualización de $\mathbf{F}_k$ en cada frame calculando Δt por medio de los milisegundos obtenidos mediante OpenFrameworks para saber el tiempo exacto transcurrido en cada frame.

En cuanto a los valores Q y R de las matrices de covarianza $\mathbf{Q}_k$ y $\mathbf{R}_k$, se han definido experimentalmente como $Q = 0{,}0001$ y $R = 0{,}1$. Por lo tanto, las matrices $\mathbf{Q}_k$ y $\mathbf{R}_k$ son fijas durante todo el proceso.

6.5.2. Asociación de datos

Uno de los problemas que nos encontramos antes de aplicar el filtro de Kalman es el de la asociación de los datos entre el frame actual y el anterior. Cuando se hace tracking de un solo objeto no es necesaria ninguna asociación, sin embargo, cuando el tracking es entre varios objetos simultáneos, no es trivial asociar cada elemento con su nuevo valor entre sí.

Para solucionar el problema de la asociación de datos se han usado dos tipos de métodos, por proximidad y mediante una matriz de distancias.

6.5.2.1. Proximidad entre objetos

El primer método que utilizamos para asociar los datos entre sí se trata del más simple. Para cada punto del frame anterior, se le asocia el punto del frame actual que más cerca se encuentre de él. En ese sentido, este método es similar al knn (k nearest neighbour).

Debido a que el número de puntos en un frame a veces no concuerda con el número del frame anterior, se definen como máximo 5 puntos a seguir para los 5 dedos. De esta forma, si en algún momento falta un dedo, éste se asigna a otro punto y comparte posición con otro dedo. Este es uno de los inconvenientes del método.

6.5.2.2. Optimización de distancias

En este método se utiliza una matriz en la que se calcula las distancias de todos los puntos entre si, los nuevos y los viejos. Posteriormente se asocian los puntos entre sí de forma que la suma de las distancias de un objeto nuevo con su correspondiente del frame anterior sea minimizada.

El algoritmo utilizado para este problema de optimización es el llamado Hungarian Algorithm. Gracias a este algoritmo se obtienen los índices de la asociación de forma que no hay elementos en la solución que compartan fila o columna. En nuestro caso esto significa que se asocian los datos de forma que dos nuevos puntos no comparten el mismo punto antiguo ni dos puntos antiguos comparten el mismo nuevo punto.

La matriz sobre la que se aplica el algoritmo tiene la forma de la ecuación 6.15. Las filas representan los puntos del frame anterior, mientras que las columnas son los nuevos puntos. Llamemos al conjunto de puntos antiguos P^{old} y al de los nuevos P^{new}. De esta manera el elemento D_{ij} representa la distancia del punto P_i^{old} al punto P_j^{new}.

$$\mathbf{M} = \begin{pmatrix} D_{00} & D_{01} & D_{02} & D_{03} & D_{04} \\ D_{10} & D_{11} & D_{12} & D_{13} & D_{14} \\ D_{20} & D_{21} & D_{22} & D_{23} & D_{24} \\ D_{30} & D_{31} & D_{32} & D_{33} & D_{34} \\ D_{40} & D_{41} & D_{42} & D_{43} & D_{44} \end{pmatrix} \tag{6.15}$$

Un ejemplo de la matriz **M** en una ejecución del programa se puede ver en la figura 6.23.

Los valores de la matriz son las distancias entre los puntos. Las que tienen un asterisco son las que pertenecen a la solución final. Debajo se encuentra la asociación de índices y la suma total de distancias optimizada.

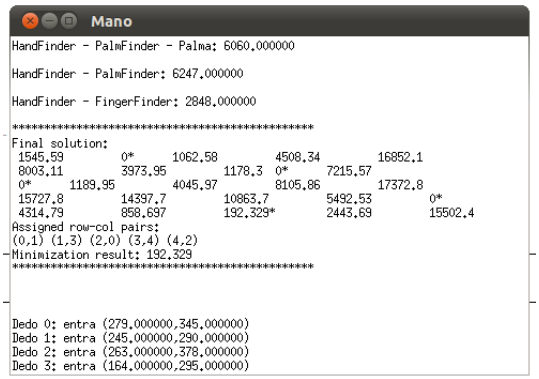

Figura 6.23: Extracto de la consola del programa donde se aprecia la matriz de distancias **M**.

Gracias a este método se resuelve el problema de asignar dos puntos del frame anterior a un mismo punto del nuevo frame, ya que este tipo de algoritmo de asociación provoca que no se puedan compartir puntos.

6.6. Diagrama de flujo

Finalmente se presenta el diagrama de flujo de la aplicación 6.24 donde se representan todos los algoritmos que intervienen en el programa.

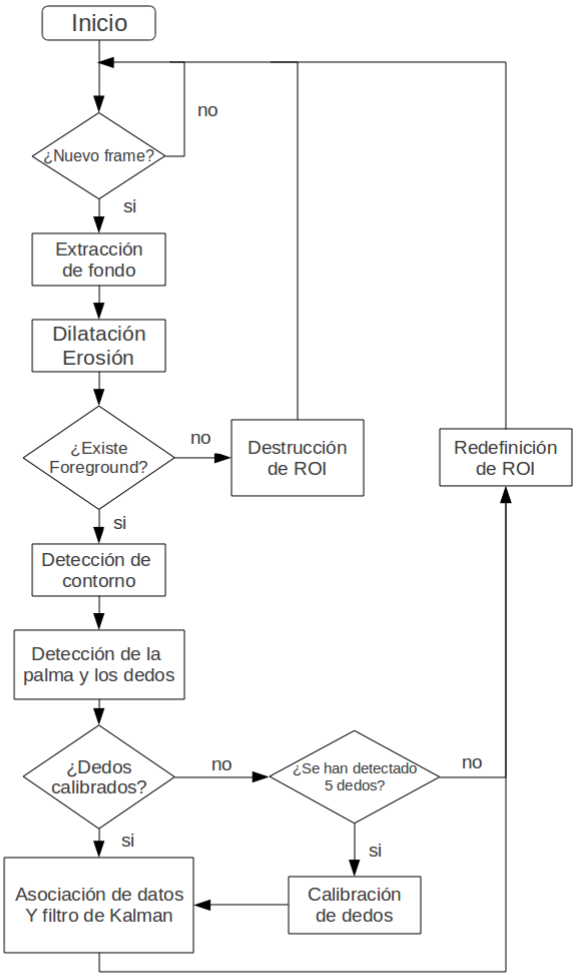

Figura 6.24: Diagrama de flujo.

Como se puede observar, carece de función final, ya que este diagrama de flujo representa los procesos que tienen lugar en cada frame. El final de la aplicación llega al pulsar la tecla escape en cualquier momento de su ejecución.

Capítulo 7

Resultados

En esta sección discutiremos algunos de los resultados obtenidos del programa, se revisarán los problemas que nos encontramos y se estudiará la velocidad de la aplicación mediante los datos de fps.

7.1. Estudio de fps

Hay que tener en cuenta que los fps máximos que pudiese alcanzar la aplicación vienen determinados por la tasa de transferencia de la Kinect, y están en aproximadamente 30fps.

Se ha hecho un estudio de la tasa de refresco de la aplicación mostrado en la siguiente tabla:

	Primera mitad	**Segunda mitad**
Mostrando las imágenes	8.23	7.05
Sin mostrar imágenes	15,73	16,12

Como hemos comprobado, la muestra de imágenes por pantalla es una de las cosas que más coste computacional tiene, por lo que para aumentar los fps de nuestra aplicación, podremos prescindir de mostrarlas. Sin embargo, como algunos de los datos son necesarios, como las puntas de los dedos, sólo mostraremos aquellos imprescindibles en la ventana. El perfil de fps obtenido en este caso es el siguiente:

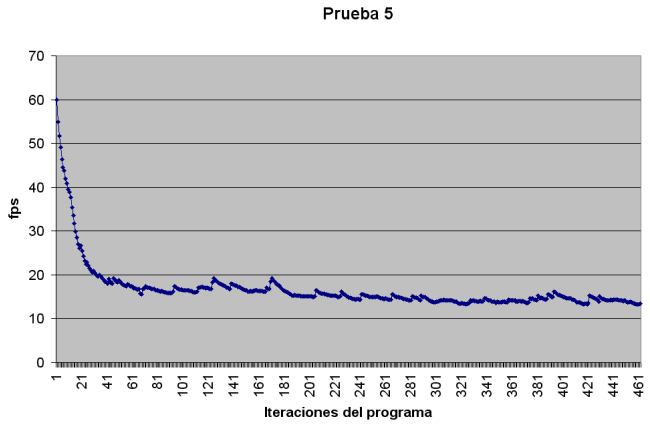

Figura 7.1: Aplicación con fps optimizados.

La media de fps conseguida por el programa finalmente es de: $\bar{fps} = 15{,}204525$

7.2. Coste computacional de los algoritmos

A continuación hablaremos sobre cada algoritmo por separado y sus problemas, si los hay. También calcularemos cuánto tarda cada algoritmo en procesar la imagen de cara a mejorar la velocidad de la aplicación, al igual que hemos explicado en el apartado anterior con los fps en general. Las gráficas que se muestran pertenecen todas a la misma ejecución, y sigue el modelo que hemos utilizado anteriormente: 30 segundos en total divididos en dos mitades, la primera sin mostrar la mano y la segunda mostrándola.

7.2.1. Extracción de fondo

La extracción de fondo se ejecuta en cada frame y, a no ser que haya un objeto en pantalla, se aplica a toda la imagen. En caso de que se detecte un objeto mediante el procesado posterior, en los siguientes frames solo se actualiza la parte de la imagen comprendida dentro de la ROI del objeto, por lo que la carga de proceso disminuye bajo ese supuesto.

A continuación se muestra una gráfica del tiempo que requiere el algoritmo en aplicarse

durante cada frame.

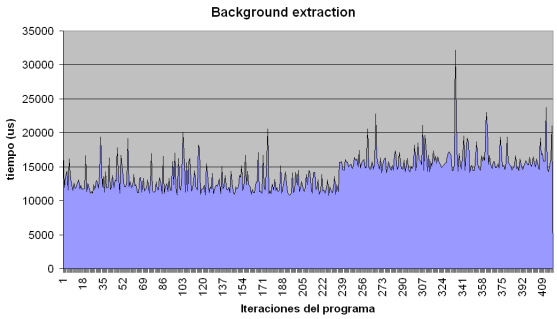

Figura 7.2: Coste computacional de la extracción de fondo.

Al comprobar la figura 7.2, nos damos cuenta de que nuestro supuesto sobre la disminución de la carga de proceso al aplicar la ROI no es del todo cierto. Se observa que aumenta el tiempo medio del proceso al mostrar la mano. Esto puede significar que la eliminación de los píxeles que no pertenezcan a la ROI no es suficiente de cara a mejorar la extracción de fondo. Sin embargo es una ayuda a tener en cuenta para el procesado posterior, por lo que se seguirá utilizando, dejando para un futuro una mejor optimización de la extracción de fondo mediante la utilización de una ROI.

7.2.2. Dilate-Erode

En cuanto al procesado de la parte donde aplicamos las erosiones y dilataciones, éste aparece en la figura 7.3. En la imagen se puede observar que el tiempo es bastante estable durante toda la ejecución. Esto se debe a que este proceso se aplica siempre, al igual que la extracción de fondo, pero su coste no depende tanto del tipo de imagen y es bajo con respecto a otros procesos.

Con respecto a los procesos de erosión y dilatación no hay problemas que mencionar, ya que se aplican sin problema a la imagen previamente binarizada. La única pega que se le puede poner es la pérdida de información de dos dedos cuando se juntan, ya que se "fusionan" por así decirlo. Pero es un coste aceptable dada la cantidad de ruido que se filtra en esta etapa.

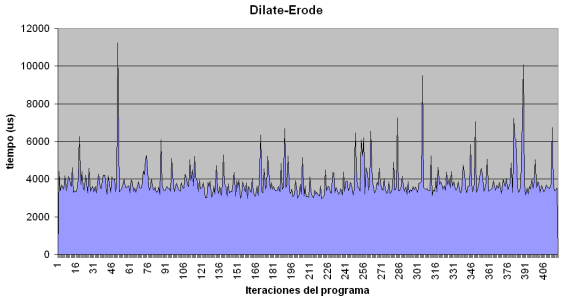

Figura 7.3: Coste computacional de la etapa de dilate-erode.

7.2.3. Extracción de contorno

La figura 7.4 representa el tiempo invertido en la extracción del contorno del objeto. En esta etapa es donde se decide si el elemento es un objeto o no dependiendo de su tamaño, por lo que se ejecuta siempre en todos los frames si existe al menos un píxel de foreground. Al igual que con el proceso anterior, el tiempo es relativamente estable y continuo durante toda la ejecución del ejemplo.

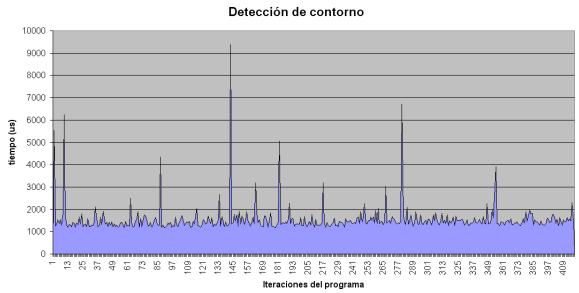

Figura 7.4: Coste computacional de la extracción de contornos.

En esta etapa se tienen las mismas consideraciones que en la anterior, ya que no hay ningún problema relevante que comentar. La extracción de fondo funciona satisfactoriamente en todos los casos y su velocidad de cálculo no depende en exceso de la imagen, según la figura 7.4.

7.2.4. Convex Hull

El proceso para calcular el Convex Hull y los siguientes algoritmos solo se ejecuta cuando se detecta un objeto en escena lo suficientemente grande como para considerarse un blob y contener la mano. Por ello, en la figura 7.5 se observa que, durante la primera mitad de la ejecución solo se ejecuta accidentalmente por culpa del ruido. Esto es porque a veces existe ruido que se segmenta, y si es lo bastante grande como para considerarse un objeto, entonces se procesa aplicando el Convex Hull.

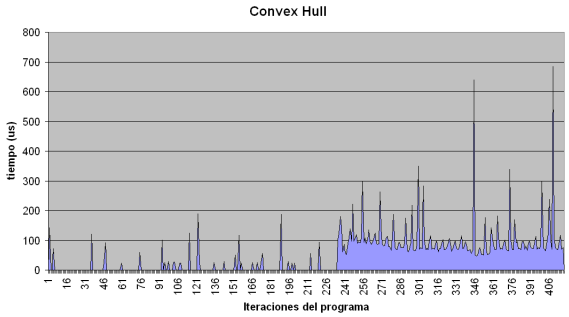

Figura 7.5: Coste computacional del Convex Hull.

La aplicación de Convex Hull no tiene ningún problema, ya que se basa en los puntos del contorno extraídos anteriormente.

7.2.5. Algoritmo de palma de la mano

El algoritmo que hemos desarrollado para calcular el punto perteneciente a la palma de la mano es de los que más carga de proceso consume, son contar la extracción de fondo, que es el más limitante. Se puede ver en la figura 7.6 como, aunque en la primera mitad prácticamente no se ejecuta, cuando lo hace llega hasta los 5000 μs.

Además se puede observar que no se mantiene constante, sino que varía enormemente dependiendo de la imagen. Esto se puede deber a que el algoritmo hace el cálculo de distancias para todos los puntos del blob, por lo que si el blob es más grande, el procesado requerirá más

tiempo. Debido a ello se aprecia en la gráfica un "valle" sobre la iteración 350 perteneciente a un blob pequeño.

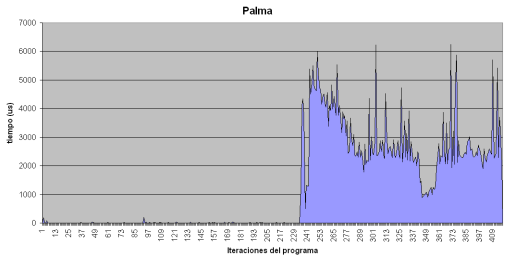

Figura 7.6: Coste computacional de la obtención de la palma.

Como el coste de esta parte es alto, se optó por reducir los puntos para los que se hacen los cálculos tal como está explicado en el capítulo anterior. Utilizando 1 de cada 4 puntos del blob y 1 de cada 4 puntos del contorno se logra disminuir el coste del algoritmo en un 40 % aproximadamente.

7.2.6. Identificación de dedos

En cuanto al perfil de tiempos del algoritmo que clasifica los dedos, su gráfica se encuentra en la figura 7.7. El perfil es bastante similar al del cálculo de la palma, pero su coste es menor.

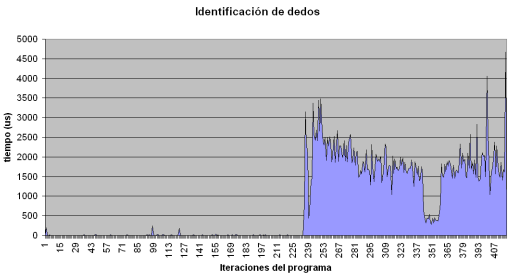

Figura 7.7: Coste computacional de la clasificación de dedos.

7.2.7. Otras características

Para el rectángulo y la circunferencia circunscritos sólo mostraremos su gráfica (fig. 7.8) y mencionaremos que el tiempo invertido en estos procesos es mínimo y casi despreciable con

respecto a los demás.

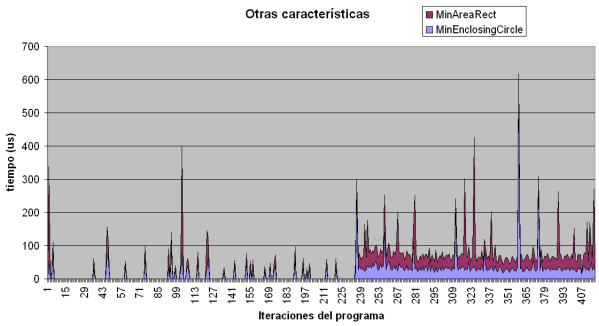

Figura 7.8: Coste computacional del rectángulo (granate) y circunferencia (azul) circunscritos.

7.2.8. Asociación de datos y filtro de Kalman

Para esta parte del programa podemos mencionar que los procesos de asociación de datos y aplicación del filtro de Kalman son relativamente estables y con un coste computacional bajo. La gráfica de los dos procesos en conjunto se muestra en la figura 7.9.

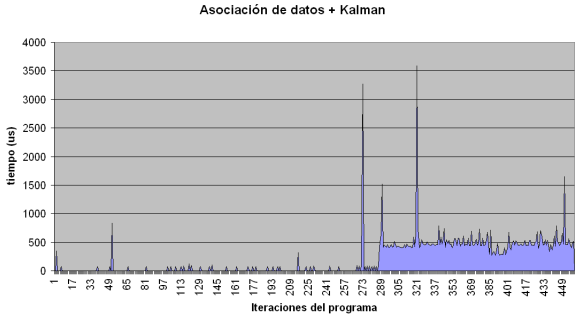

Figura 7.9: Coste computacional de la asociación de datos y aplicación del filtro de Kalman.

Para el filtro de Kalman simplemente hay que ajustar el valor de las covarianzas $\mathbf{Q}_k$ y $\mathbf{R}_k$. Se ha observado que, manteniendo una covarianza $Q = 0{,}1$, al variar la otra covarianza R, la de las medidas, se lograba evitar movimientos bruscos en la posición de los puntos filtrados. Esto es así porque el filtro admite más ruido en las medidas y le cuesta más cambiar de estado. Esto evitaba en cierta medida el problema antes mencionado de identificación del pulgar, y podía

servir para evitar que el punto de la palma se moviese hacia la muñeca. Sin embargo creaba el problema de ser excesivamente lento si movíamos la mano entera de lugar, como se aprecia en la figura 7.10. Finalmente se llegó a una solución intermedia utilizando un valor de $R = 10$.

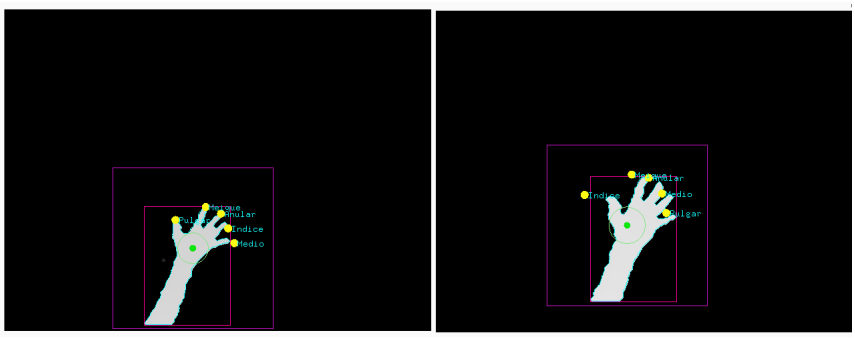

Figura 7.10: Filtro de Kalman con R = 10 y R = 100.

Para la asociación de datos se añadió una comprobación de forma que si los puntos a asociar están a una distancia menor de 100 píxeles, se fuerza la asociación colocando en la matriz un 0 para la distancia entre esos puntos. También, cuando de los 5 puntos seguidos alguno se ha perdido, es decir, no hay datos de ese punto, se coloca el valor de distancia 640000 en su columna. De este modo se prioriza asociar los demás puntos y el que quede suelto se asocia con el punto nuevo que quede.

7.3. Aplicación completa

Finalmente se muestra la gráfica del tiempo total de la aplicación para cada frame, donde se observa la suma de todos los tiempos calculados anteriormente. Se han omitido los datos de la circunferencia y rectángulo circunscrito y se han elegido colores que destaquen entre sí. Se observa un aumento del tiempo de proceso durante la segunda mitad del ejemplo, como era de esperar.

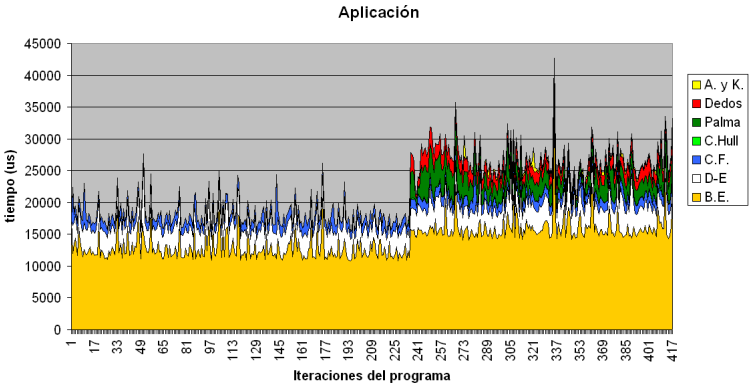

Figura 7.11: Coste computacional total de la aplicación.

7.4. Robustez de los algoritmos

En esta sección se llevarán a cabo experimentos para determinar en la medida de lo posible la robustez de los diversos procesos utilizados y anotando sus fallos en cada ejecución si los hubiese.

7.4.1. Experimento 1: Extracción de fondo

Durante estas pruebas se ha ejecutado la aplicación varias veces durante un tiempo de 1 minuto cada una sin mostrar la mano a la cámara. De esta manera hemos comprobado el funcionamiento de la extracción de fondo, el filtrado dilate-erode y la detección de blobs si los hubiese. Los resultados esperados para este experimento son una imagen procesada totalmente negra por la ausencia de objetos en el foreground y el filtrado, y una falta de blobs por pantalla. Se han aplicado las ejecuciones en tres situaciones diferentes, para el mismo fondo, con un umbral de 5, 10 y 15. Los resultados se muestran en la siguiente tabla.

	Ejecuciones totales	Detecciones de foreground	% de aciertos
T = 5	10	7	30 %
T = 10	10	2	20 %
T = 15	10	0	100 %

Un ejemplo de ejecución correcta sin blobs en pantalla y una detección falsa aparecen en las figuras 7.12 y 7.13 respectivamente.

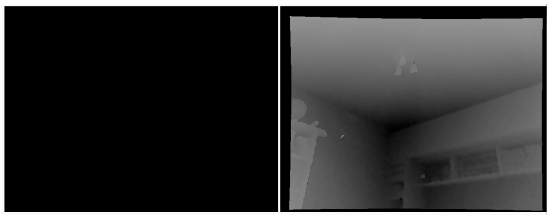

Figura 7.12: Resultado satisfactorio del experimento 1.

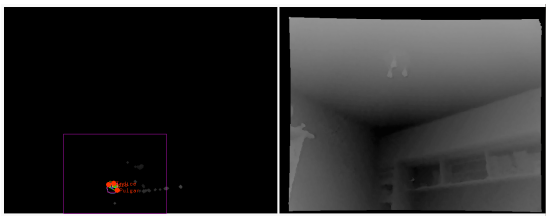

Figura 7.13: Resultado incorrecto del experimento 1.

Estos resultados evidencian que la robustez de la extracción de fondo depende en principio del umbral que utilicemos a la hora de comparar la imagen con el fondo. De esta forma, con un umbral bajo se definen mejor los objetos cercanos al fondo, pero aparece más ruido y por lo tanto detecciones falsas. Por el contrario, con un umbral alto se evita este problema, pero los objetos deben estar suficientemente alejados del fondo para ser detectados. Este hecho puede observarse en las figuras 7.14 y 7.15.

También se ha descubierto la sensibilidad de esta parte del procesado al movimiento de la cámara, ya que gran parte de los fallos originados en las ejecuciones se debían a un pequeño movimiento de la misma que modificaba ligeramente la escena detectada.

En nuestro caso se ha utilizado una solución intermedia usando $T = 10$.

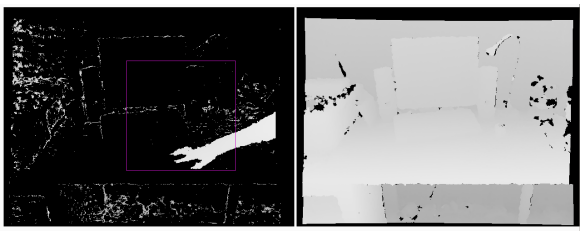

Figura 7.14: Extracción de fondo sin filtrado (dilate-erode) y con un umbral de 1. La mano está apoyada en la mesa, pero es reconocible.

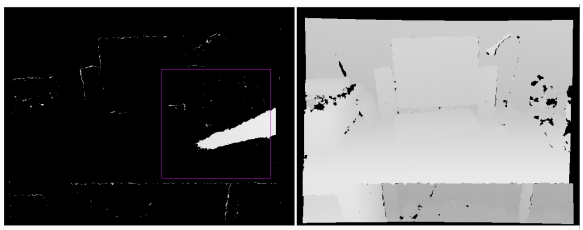

Figura 7.15: Extracción de fondo sin filtrado (dilate-erode) y con un umbral de 10. La mano está apoyada en la mesa, pero ya no se aprecia.

7.4.2. Experimento 2: Reconocimiento de la palma

En este segundo experimento se ha ejecutado la aplicación varias veces mostrando la mano a la cámara en diferentes posiciones. Se ha comprobado la detección correcta de la palma de la mano y por consiguiente, la definición de la ROI asociada. Además se han hecho las pruebas a diferentes distancias de la cámara anotando las veces que la detección ha sido correcta, y cuándo ha fallado. Los resultados aparecen en las siguientes tablas:

Distancia a la cámara: $0{,}6m$ aprox.

	Ejecuciones totales	**Detecciones correctas**	**% de aciertos**
Mano abierta vertical	10	7	70 %
Mano abierta horizontal	10	8	80 %
Mano cerrada	10	5	50 %

Distancia a la cámara: $1{,}5m$ aprox.

	Ejecuciones totales	Detecciones correctas	% de aciertos
Mano abierta vertical	10	5	50 %
Mano abierta horizontal	10	4	40 %
Mano cerrada	10	1	10 %

Las detecciones incorrectas que aparecen en las tablas se deben a un problema que se ha observado al utilizar este algoritmo y que tiene que ver con la ROI. Al definir la ROI en función de la palma de la mano, la hacemos muy sensible a los cambios de este punto. Por ello, y debido a que de vez en cuando se calcula un punto cercano a la muñeca como la palma, la ROI "viaja" desde la mano hasta el codo. Este efecto queda reflejado en la figura 7.16.

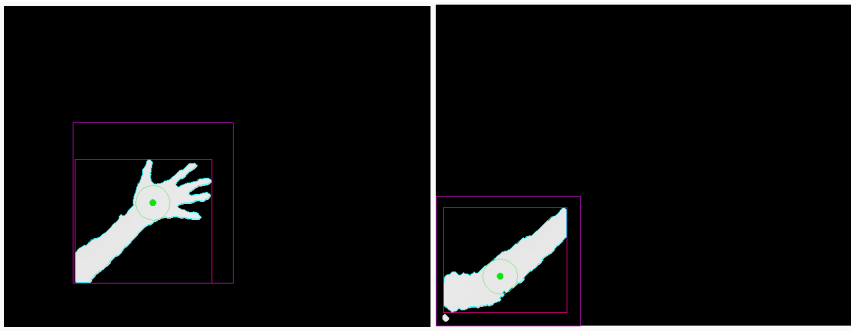

Figura 7.16: Identificación correcta de la palma de la mano a la izquierda y posterior error a la derecha.

Los resultados evidencian un alto grado de fallo si la mano está lejos de la cámara. Esto es debido a que la diferencia entre el tamaño de la palma y el tamaño del codo cada vez es más pequeña, lo que provoca que tarde o temprano la ROI migre hacia esa zona. En cuanto a la diferencia entre la mano abierta y cerrada se observa que el algoritmo falla más cuando la mano se encuentra cerrada que cuando se encuentra abierta, ya que al no ser una superficie plana la mano cerrada tiene sombras que distorsionan el contorno de la misma.

7.4.3. Experimento 3: Clasificación de dedos

Este experimento se ha centrado en comprobar la robustez del algoritmo que calcula los puntos pertenecientes a la punta de los dedos. Al igual que en los anteriores se han utilizado

varias ejecuciones del programa, en este caso 3, y para cada ejecución de 30 segundos se han anotado las veces que se producía un fallo en la detección de los dedos. Se han tenido en cuenta dos tipos de errores: falta de posición de un dedo o clasificación de punto erróneo.

La falta de información en la posición del dedo ocurre cuando los valores de curvatura alrededor de la punta del dedo en cuestión no superan todas las limitaciones que definen si un punto es dedo o no. De esta manera simplemente perdemos la información sobre ese punto hasta que se vuelva a recuperar.

En cuanto a la clasificación errónea, suele ocurrir que el punto que define al pulgar se pierde y en su lugar, de vez en cuando, aparece un segundo punto en el índice, por lo que se asocia a éste. Esto provoca que el punto salga "disparado" debido a la aplicación del filtro de Kalman. Ocurre porque el punto pasa de estar en el pulgar a estar en el índice, y posteriormente perderse, de forma que las predicciones del filtro de Kalman muestran el punto moviéndose hacia fuera.

Otro punto conflictivo y que a veces se interpreta como el pulgar se encuentra en el limite de la ROI, en la muñeca, tal como se muestra en la figura 7.17. En esta figura, los dedos aparecen descolocados, ya que a medida que avanza la aplicación y dependiendo de lo puntos que se pierdan, éstos van cambiando de posición.

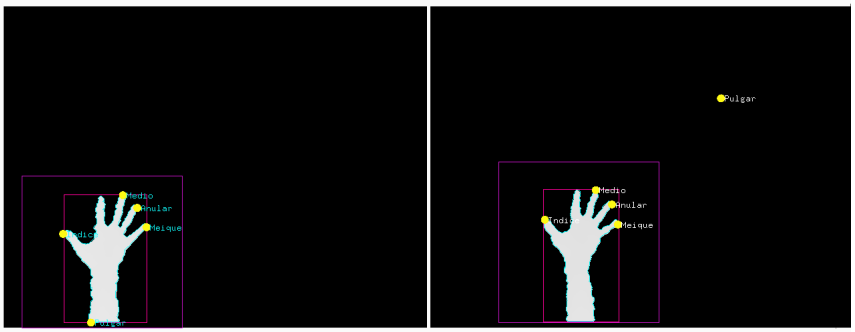

Figura 7.17: Clasificación errónea del pulgar en la muñeca (izq.) y en el índice y posterior estimación de Kalman (der.).

En esta tabla se muestran los resultados al probar diferentes valores para los parámetros

de curvatura mínima c_{min} y el valor l de la ecuación 4.15. La distancia a la cámara es de $0{,}6m$ aproximadamente.

	Ejecuciones	**Falta de información**	**Dedo en posición errónea**	**Fallos por ejecución**
$c_{min} = 0{,}25$ y $l = 5$	3	9	9	6
$c_{min} = 0{,}5$ y $l = 5$	3	11	13	8
$c_{min} = 0{,}75$ y $l = 5$	3	26	8	11.3
$c_{min} = 0{,}25$ y $l = 15$	3	9	7	5.3
$c_{min} = 0{,}5$ y $l = 15$	3	5	6	3.6
$c_{min} = 0{,}75$ y $l = 15$	3	11	8	6.33

Los resultados indican que la identificación de dedos es más robusta cuando se utiliza $l = 15$ que un número mucho más bajo, $l = 5$. Además, la pérdida de puntos aumenta al aumentar el valor de curvatura mínima. Esto tiene sentido, ya que al aumentar este valor, hacemos más restrictiva la clasificación.

Los valores de curvatura mínima y l que hemos utilizado finalmente son $c_{min} = 0{,}56$ y $l = 10$.

Capítulo 8

Conclusiones y trabajo futuro

Gracias a la ejecución de este proyecto hemos sido capaces de desarrollar una aplicación que consigue los siguientes objetivos:

1. Obtener las imagenes calibradas provenientes de una cámara 3D con tecnología Primesense, en este caso una Kinect de Microsoft.

2. Utilizar la información 3D proveniente de la cámara para extraer el fondo de la escena y segmentar los objetos que no pertenecen a él.

3. Crear una ROI (Region Of Interest) alrededor del objeto detectado y suavizar su forma para eliminar el ruido de la imagen.

4. Segmentar la mano humana e identificar en ella la punta de los cinco dedos además de la palma de la mano, obteniendo sus posiciones en 3D además de otras características relevantes para calcular su orientación y localización.

5. Hacer un seguimiento (tracking) de los puntos calculados asociando cada "track" mediante el "Hungarian Algorithm" y posteriormente la aplicación de un filtro de Kalman.

6. Mostrar por pantalla todos los datos anteriores mediante diferentes colores para su fácil identificación por parte del usuario.

Además, los fps (frames por segundo) conseguidos con el programa están en torno a 15.2, por lo que se puede considerar como una aplicación en tiempo real. Las conclusiones que obtenemos del proyecto son el enorme potencial de las cámaras 3D, ya permiten modelar el entorno

de una manera más fiel que las cámaras en 2D, y en concreto las cámaras Primesense por su precio, facilidad de uso y robustez.

También se ha puesto de manifiesto la dificultad del procesamiento de imágenes y la visión artificial, ya que, aunque para un humano la segmentación de objetos y la localización de los mismos parece algo trivial, para un computador no lo es.

Por último, algunas mejoras que se pueden aplicar al programa de este trabajo y que se pueden considerar como trabajo futuro del mismo son:

- Limitar la identificación de puntos clasificados como dedos introduciendo un modelo paramétrico de la mano humana e identificar, además de las puntas de los dedos, las falanges de los mismos.
- Aplicar los algoritmos para reconocer, segmentar y seguir las dos manos del usuario al mismo tiempo.
- Añadir una segmentación por color para que sirva de apoyo a la extracción de fondo de cara a no identificar otros objetos que no sean la mano.

Bibliografía

[1] A. A. Argyros and M. I. A. Lourakis. Real-time tracking of multiple skin-colored objects with a possibly moving camera. *Proc. European Conference on Computer Vision*, pages 368–379, 2004.

[2] A. A. Argyros and M. I. A. Lourakis. Vision-based interpretation of hand gestures for remote control of a computer mouse. *ECCV Workshop on HCI*, pages 99–108, 2006.

[3] Antonis A. Argyros and Manolis I. A. Lourakis. Tracking skin-colored objets in real-time. In *Invited Contribution to the Çutting Edge Robotics Book", ISBN 3-86611-038-3, Advanced Robotics Systems International*, 2005.

[4] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2002.

[5] A. Blake, B. North, and M. Isard. Learning multi-class dynamics. *Proc. Advances in Neural Information Processing Systems (NIPS)*, 11:389–395, 1999.

[6] G. Bradski. Real time face and object tracking as a component of a perceptual user interface. *IEEE Workshop on Applications of Computer Vision*, pages 214–219, 1998.

[7] M. Breig and M. Kohler. Motion detection and tracking under constraint of pan-tilt cameras for vision-based human computer interaction. *Technical Report 689, Informatik VII, University of Dortmund/Germany*, 1998.

[8] D. Chai and K. Ngan. Locating the facial region of a head and shoulders color image. *IEEE Int. Conference on Automatic Face and Gesture Recognition*, pages 124–129, 1998.

[9] J. Crowley, F. Berard, and J. Coutaz. Finger tracking as an input device for augmented reality. *International Workshop on Gesture and Face Recognition*, 1995.

[10] Y. Cui and J. Weng. Hand sign recognition from intensity image sequences with complex background. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 88–93, 1996.

[11] J. Davis and M. Shah. Visual gesture recognition. *Vision, Image, and Signal Proccesing*, 1994.

[12] A. Downton and H. Drouet. Image analysis for model-based sign language coding. *Int. Conf. Image Analysis and Proccesing*, pages 637–644, 1991.

[13] D. Gavrila and L. Davis. Towards 3D model-based tracking and recognition of human movement: a multi-view approach. *Int. Workshop on Automatic Face and Gesture Recognition*, pages 272–277, 1995.

[14] D. Gavrila and L. Davis. 3d model-based tracking of humans in action: a multi-view approach. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 73–80, 1996.

[15] L. Goncalves, E. di Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3D. *Proc. International Conference on Computer Vision (ICCV)*, pages 764–770, 1995.

[16] Martin Haker, Martin Bohme, Thomas Martinetz, and Erhardt Barth. Geometric invariants for facial feature tracking with 3D TOF cameras. Technical report, University of Lubeck, Germany, 2007.

[17] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *Int. Journal of Computer Vision*, 1998.

[18] J. Lin, Y. Wu, and T. S. Huang. Capturing human hand motion in image sequences. *Proc. IEEE workshop on Motion and Video Computing*, pages 99–104, 2002.

[19] Xuming Luan. *Experimental investigation of photonic mixer device and development of TOF 3D ranging systems based on PMD technology*. PhD thesis, University of Siegen, 2001.

[20] J. MacCormick and M. Isard. Partitioned sampling, articulated objects and interface-quality hand tracking. *Proc. European Conference on Computer Vision*, pages 3–19, 2000.

[21] Shazad Malik. Real-time hand tracking and finger tracking for interaction. *CSC2503F Project Report*, 2003.

[22] Cristina Manresa, Javier Varona, Ramon Mas, and Francisco J. Perales. Real-Time Hand Tracking and Gesture Recognition for Human-Computer Interaction. *Electronic Letters on Computer Vision and Image Analysis*, 2000.

[23] J. Martin, V. Devin, and J. Crowley. Active hand tracking. *IEEE Conference on Automatic Face and Gesture Recognition*, pages 573–578, 1998.

[24] Hans P. Moravec. Robot spatial perception by stereoscopic vision and 3D evidence grids. *Carnegie Mellon University*, 1996.

[25] R. O'Hagan and A. Zelinsky. Finger track - a robust and real-time gesture interface. *Australian Joint Conference on Artificial Intelligence*, pages 475–484, 1997.

[26] Iasonas Oikonomidis, Nikolaos Kyriazis, and Antonis A. Argyros. Markerless and efficient 26-DOF hand pose recovery. *Institute of Computer Science, FORTH and Computer Science Department, University of Crete*, 2010.

[27] Iasonas Oikonomidis, Nikolaos Kyriazis, and Antonis A. Argyros. Efficient model-based 3D tracking of hand articulations using Kinect. *Institute of Computer Science, FORTH and Computer Science Department, University of Crete*, 2012.

[28] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. *Proc. European Conference on Computer Vision*, pages 661–675, 2002.

[29] J. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. *Proc. International Conference on Computer Vision (ICCV)*, pages 612–617, 1995.

[30] D. Saxe and R. Foulds. Toward robust skin identification in video images. *IEEE Int. Conf. on Automatic Face and Gesture Recognition*, pages 379–384, 1996.

[31] Stefan Stegmueller. Candescent NUI webpage. http://blog.candescent.ch.

[32] B. Stenger, R. Mendonca, and R. Cippola. Model-based 3D tracking of an articulated hand. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 126–133, 2002.

[33] A. Utsumi and J. Ohya. Image segmentation for human tracking using sequential-image-based hierarchical adaptation. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 911–916, 1998.

[34] A. Utsumi and J. Ohya. Multiple-hand-gesture tracking using multiple cameras. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 473–478, 1999.

[35] Y. Wu, J. Lin, and T. Huang. Capturing natural hand articulation. *Proc. International Conference on Computer Vision (ICCV)*, pages 426–432, 2001.

[36] J. P. Zhou and J. Hoang. Real time robust human detection and tracking system. *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 149–149, 2005.