Estudio sobre el uso de sensores RGB-D para el conteo de personas y su caracterización

---

Proyecto Fin de Carrera
Ingeniería Informática
25 de mayo de 2015

Autor:
Quesada Díaz, Eduardo

Tutores:
Castrillón Santana, Modesto
Lorenzo Navarro, Javier
Hernández Sosa, Daniel

Proyecto Fin de Carrera

Proyecto Fin de Carrera de la Escuela de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

Apellidos y nombre del alumno: Quesada Díaz, Eduardo
Título: Estudio sobre el uso de sensores RGB-D para el conteo de personas y su caracterización
Fecha: Mayo de 2015
Tutores: Castrillón Santana, Modesto
Lorenzo Navarro, Javier
Hernández Sosa, Daniel

Dedicatoria

A mis padres,
mi familia,
y mi novia.

Agradecimientos

Tras estos años de carrera, se me hace difícil agradecer en meras palabras todo la ayuda y el apoyo que he recibido en esta etapa de mi vida.

En primer lugar quiero darle las gracias a mis tutores: Modesto, Javier y Daniel. Sin su ayuda, paciencia y conocimientos no podría haber pensado siquiera en completar este proyecto. Además, nuestras reuniones semanales han sido una fuente de inspiración con los diversos problemas que han ido surgiendo en el desarrollo del mismo. No sólo he de agradecer a mis tutores, ya que también el resto de profesores que he tenido en la carrera me han ayudado a llegar hasta aquí; muchas gracias a todos.

Estas reuniones me llevan también a los componentes de nuestro grupo de proyectos: Rubén, Luis, Pedro, Alberto y Dani. Ellos han sido testigos de las pequeñas victorias y las grandes frustraciones que he vivido. Muchas gracias también por las sugerencias, consejos, trucos e ideas que me habéis aportado, y por amenizar el viaje hasta llegar aquí.

También a todos mis compañeros de la carrera, que me han ayudado en diferentes puntos de la misma, y con los que he pasado muchos momentos buenos y algunos algo más estresantes. Quiero hacer una mención especial a Alberto y a Dani, que también forman parte del grupo de proyectos, por echar una mano siempre que me hiciera falta y por ser excelentes amigos.

A Mayca, no sólo por ser el apoyo más grande que podía encontrar en esta carrera, sino por estar siempre de mi lado pasara lo que pasara. Hemos pasado por muchísimas cosas durante estos años de carrera, pero no importaba lo que fuera, ella siempre ha estado conmigo. Muchísimas gracias por todo, no me imagino cómo habría llegado hasta aquí si no fuera por tí.

Por último, agradecer a toda mi familia y sobre todo a mis padres, el constante apoyo que me han dado. Me han ayudado, comprendido y ofrecido toda la ayuda que me podían ofrecer; un gracias se me queda corto. Desde antes de comenzar me han estado animando, y por fin van a poder verme finalizar esta etapa de mi vida.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Un contador de personas consiste en un sistema que permite determinar el número de individuos que atraviesan una determinada región. Originalmente, estas tareas se realizaban de forma manual, pero el desarrollo de la tecnología ha permitido utilizar diversos recursos electrónicos e informáticos para llevar a cabo esta tarea de manera automática. En este proyecto se presenta un contador de personas automático que utiliza técnicas de visión por computador para realizar dicho conteo utilizando secuencias de imágenes recogidas con un sensor Kinect, así como un estudio sobre la caracterización del sexo de las personas capturadas.

1.1. Motivación

El conteo de personas es una característica deseable para un sistema automático con aplicaciones potenciales en múltiples escenarios. Por ilustrar algunos ejemplos, es necesario conocer el número de pasajeros que entran y salen de un medio de transporte público para llevar a cabo su control y gestión. En pubs y discotecas los protocolos de evacuación están diseñados de acuerdo con la capacidad del local, no debiendo sobrepasarse para evitar situaciones peligrosas. El control de presencia es también esencial para la implantación de políticas de ahorro energético. En otro contexto, la obtención automática de datos de audiencia, así como la medida de los tiempos de atención, es también una capacidad de interés para empresarios y anunciantes. Dadas las diferentes aplicaciones para esta tarea y su creciente interés, se han propuesto diferentes soluciones para conocer, lo más exactamente posible, el

número de personas que accede y sale de un espacio delimitado. En este documento presentamos un método de conteo automático utilizando técnicas de visión por computador.

Desde un punto de vista personal, he buscado que este proyecto ampliara mis conocimientos en visión por computador, un área que me interesa pero que no he podido ver en profundidad a lo largo de la carrera. También he buscado un proyecto que pueda ser útil en el futuro, y dado que es un tema relativamente común en la comunidad que trabaja con visión por computador, también cumple este requisito.

1.2. Estado del arte

Se han utilizado diferentes tecnologías en sistemas de conteo de personas; desde sistemas con rayos infrarrojos, en los que el conteo se produce al interrumpirse el haz de luz infrarroja entre el emisor y el receptor de infrarrojos, hasta sensores térmicos, que detectan el calor emitido por las personas, pasando por sistemas que utilizan técnicas de visión por computador, como el implementado en este proyecto.

Los sistemas de visión por computador están compuestos por una cámara que envía imágenes a un ordenador o dispositivo integrado, que realiza el procesamiento de las mismas. Esta tecnología destaca por su flexibilidad y capacidad de integración con otros sistemas. En este caso, las imágenes son procesadas con el fin de localizar y contabilizar personas. Dependiendo del tipo de cámara usada se puede diferenciar entre sistemas RGB y sistemas basados en sensores de profundidad. Los sistemas de visión por computador con cámaras RGB suelen ser vulnerables a los cambios de iluminación, lo que puede ocasionar problemas con el conteo. Los sistemas basados en sensores de profundidad pueden dividirse a su vez en dos tipos: los que funcionan mediante transmisión y recepción de patrones de luz infrarroja, y los que lo hacen mediante visión estéreo.

Los sistemas con sensores de profundidad que emplean infrarrojos presentan problemas ante la incidencia de luz natural directa. Sin embargo, funcionan bien en escenas iluminadas con luz natural no directa, y/o luz artificial, siendo capaces de trabajar incluso en la oscuridad. El sensor que utiliza el sistema descrito en este documento incluye un sensor de profundidad con emisor y receptor de luz infrarroja. Este tipo de sensores son los que suelen utilizarse para proponer soluciones a problemas similares.

Estas herramientas, sin embargo, presentan un problema a la hora de mantener la privacidad de los transeúntes que capturan. Una vista frontal de una persona puede atentar contra la privacidad de la misma. Además, puede provocar problemas de oclusión entre los diferentes sujetos. Por eso, a la hora de utilizar estos dispositivos se suelen situar de forma que las imágenes sean capturadas desde una visión aérea o cenital. De esta forma solucionamos los dos problemas anteriores: protegemos la identidad del transeúnte al no capturar directamente sus rasgos faciales, y solucionamos la oclusión, ya que de una forma normal, dos transeúntes no caminarán uno por encima del otro.

Encontramos en la bibliografía una serie de trabajos que proponen diferentes alternativas sobre el mismo problema y utilizan herramientas similares. En primer lugar tenemos (Lorenzo-Navarro et al., 2013), donde se realiza un estudio sobre características biométricas que utiliza un sensor RGB-D para tomar imágenes desde una posición cenital. Este trabajo no se centra en la información acerca de la apariencia de los usuarios que puede aportar el sensor RGB, sino en los datos de profundidad obtenidos. Su objetivo es probar que las cámaras de profundidad comerciales pueden mejorar la información proporcionada por los descriptores geométricos de identidad para tareas de re-identificación, lo cual las convierte en herramientas valiosas para sistemas multisensores en red.

En (Castrillón-Santana et al., 2014a) se demuestra que la información de profundidad proporcionada por una cámara RGB-D comercial de bajo coste es una fuente fiable de datos para realizar de forma robusta el conteo automatizado de personas. De nuevo, se escoge una posición de la cámara cenital al paso de personas, y se varían las condiciones de iluminación durante las secuencias de imágenes, demostrando la fiabilidad de los datos de profundidad proporcionados.

Utilizando un método de captura similar, en (Zhang et al., 2012) encontramos un sistema que implementa un método denominado *water filling*, el cual permite a los autores encontrar a los transeúntes de manera robusta y sin tener en cuenta la varianza por la escala, y que ofrece unos resultados bastante positivos. En resumen, esta solución plantea una "inundación" sobre la imagen de profundidad, de manera que el agua quedará en los huecos situados entre las personas aislándolas, permitiendo su fácil individualización y conteo.

Por último, tenemos (Fernández-Pereira, 2013), donde con un planteamiento similar, se implementa un contador de personas que utiliza una variación del algoritmo *Mean-shift* para la detección de los transeúntes y un filtro de Kalman para el seguimiento y conteo de los mismos.

1.3. Objetivos del proyecto

El principal objetivo de este proyecto consiste en desarrollar un sistema que permita el conteo de personas, y la realización de un estudio acerca de las variables que lo componen y su viabilidad. Este objetivo podemos dividirlo en diferentes subobjetivos:

- Estudio de las herramientas disponibles para tareas de visión por computador.
- Detección de personas. Esta tarea comprende, además del cálculo del modelo de fondo, el eliminado del fondo de cada imagen y la separación de cada persona para su posterior tratamiento.
- Conteo de personas. Debemos hacer un seguimiento de la trayectoria de cada persona para poder contabilizarla.
- Extracción de descriptores biométricos para su caracterización. De cada persona obtendremos medidas que serán utilizadas *a posteriori* para su análisis.
- Determinación del sexo. Se utilizarán los datos recogidos para obtener un clasificador para el sexo.

1.4. Organización de la memoria

En el capítulo actual se han introducido los contadores automáticos de personas, así como su utilidad. Además, hemos estudiado otros sistemas similares al que proponemos en este documento. Por último, vemos los objetivos que nos planteamos al realizar este proyecto.

Más adelante, en el capítulo 2 se describirán las herramientas empleadas durante el desarrollo del proyecto. Describiremos el sensor utilizado, así como las bibliotecas necesarias para poder trabajar con el mismo, y otras herramientas software utilizadas.

El capítulo 3 trata de los fundamentos de ingeniería del software empleados. Describiremos el enfoque que hemos tomado con el proyecto a la hora de desarrollarlo, su división interna en clases y la planificación temporal que hemos seguido.

Posteriormente, el capítulo 4 muestra en detalle los pasos que hemos seguido para implementar este sistema. Se hablará de la adquisición de imagen, su preprocesado, cómo detectamos personas dentro de una imagen, cómo las contabilizamos y el proceso que hemos realizado para detectar su sexo.

A continuación, en el capítulo 5 mostraremos los resultados obtenidos, mostrando el funcionamiento del contador y la clasificación de sexo de las personas detectadas.

Por último, el capítulo 6 da por finalizado este documento, donde comentamos las conclusiones obtenidas tras la realización de este proyecto y proponemos futuras líneas de trabajo empleando este proyecto como base.

Capítulo 2

Herramientas utilizadas

En este segundo capítulo se describen cada uno de los componentes empleados en el desarrollo de este proyecto, tanto físicos como lógicos. En cuanto al soporte físico, hemos utilizado un sensor RGB-D, Kinect y un equipo cuyas prestaciones serán descritas también aquí. El soporte lógico comprende el entorno de desarrollo de software empleado, las bibliotecas software necesitadas y una suite de aprendizaje automático utilizada a la hora de la caracterización de sexo.

2.1. Sensor Kinect

Kinect es un *controlador de juego libre y entretenimiento* creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360, y desde junio del 2011 para PC a través de Windows 7 y Windows 8. Kinect da la posibilidad a sus usuarios de controlar e interacturar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz, objetos e imágenes.

El sensor Kinect (Mathé et al., 2012), que podemos ver en la figura 2.1, es un dispositivo conectado a un pivote, diseñado para estar en una posición horizontal. El dispositivo incluye una cámara RGB, un sensor de profundidad y un micrófono multi-array bidireccional que, en conjunto, capturan imágenes y movimientos de los cuerpos en 3D, además de ofrecer reconocimiento facial y aceptar comandos de voz.

Figura 2.1: Imagen de un sensor Kinect (Microsoft Corporation, 2015).

El sensor de Kinect adquiere imágenes de vídeo con un sensor CMOS a una frecuencia de 30 Hz, en RGB de 32-bits y resolución VGA de 640x480 píxeles. El canal de vídeo monocromo CMOS es de 16-bit y su resolución QVGA es de 320x240 píxeles con hasta 65,536 niveles de sensibilidad.

Para calcular distancias entre un cuerpo y el sensor, este emite un haz láser infrarrojo que proyecta un patrón de puntos sobre los cuerpos cuya distancia se determina. Una cámara infrarroja capta este patrón y por hardware calcula la profundidad de cada punto. El rango de profundidad detectable por Kinect se encuentra entre los 0.4 y 4 metros. Existen 2 modos (*Default* y *Near*) para determinar distancias. El modo *Default* permite medir entre 0.8 y 4 metros de distancia con respecto al sensor, mientras que el modo *Near* es capad de detectar distancias entre los 0.4 y los 3 metros. El ángulo de vista (FOV) es de 58º horizontales y 43º verticales. Por otro lado el pivote permite orientar en elevación, hacia arriba o hacia abajo, incrementando el FOV hasta en 27º más. El array del micrófono tiene cuatro cápsulas, y opera con cada canal procesando en 16-bit de audio con un ratio de frecuencia de 16 kHz. La cámara de Kinect funciona con hardware y software propios para el reconocimiento de imagen. Dicha cámara tiene dos funciones principales: generar un mapa en tres dimensiones de la imagen que se encuentra en su campo visual y reconocer humanos en movimiento entre los objetos de la imagen a partir de diferentes segmentos de las articulaciones del cuerpo y un esquema en escala de grises. El sensor Kinect puede llegar a distinguir la profundidad de cada objeto con una resolución de 1 centímetro y calcular estimaciones de la altura y anchura con una exactitud de aproximadamente 3 milímetros.

Durante la realización de este proyecto, más concretamente el 15 de julio de 2014, una versión actualizada del sensor Kinect fue lanzada al mercado, denominada Kinect para Xbox One o Kinect 2.0, mostrada en la figura 2.2. Utilizando una tecnología similar pero con mayores prestaciones, este nuevo sensor es capaz de capturar imágenes en resolución 1080p y procesa 2 gigabits de datos por segundo. Presenta una precisión y rendimiento muy superiores

a los de su predecesor, pero al lanzarse cuando este proyecto se encontraba en avanzadas fases de desarrollo, no se contempló su utilización.

Figura 2.2: Imagen de un sensor Kinect 2.0 (Microsoft Corporation, 2015).

La configuración de la captura de imágenes realizada con Kinect será explicada más adelante, en el apartado 4.1.

2.2. Entorno software

A la hora de trabajar con el sensor Kinect, existen diferentes alternativas de software que pueden ser utilizadas. Cuando Kinect fue lanzado al mercado, captó la atención de diversos grupos de investigación por sus capacidades y resolución con respecto a su precio, ya que otros sensores dedicados a visión por computador no eran capaces de alcanzar las mismas prestaciones a pesar de tener un precio superior. Esto llevó a los desarrolladores a programar sus propios controladores para este sensor utilizando ingeniería inversa, lo cual permitió utilizar Kinect como un sensor barato pero bastante efectivo en problemas de visión por computador. Posteriormente, Microsoft publicó su propio kit de desarrollo de software, que permite dar un rendimiento aún mayor a este sensor, abriendo las posibilidades de uso a características como el micrófono integrado en el mismo, no disponible con otros controladores. Este SDK tiene una documentación muy completa, pero al no tratarse de software libre, no permite la distribución del software desarrollado con el mismo. Esto hace que muchos desarrolladores se dediquen a utilizar bibliotecas de código abierto para trabajar con Kinect, como las que se han empleado en este proyecto; en concreto hemos utilizado las bibliotecas de OpenNI con los controladores de Primesense, que serán introducidos en los apartados 2.3.3 y 2.3.4 respectivamente.

En cuanto al entorno software en el que se han instalado las herramientas necesarias, se ha optado por desarrollar el presente proyecto en Ubuntu 12.04, ejecutado en un equipo con las siguientes especificaciones:

- *Procesador*: Intel® Core™2 Duo CPU T5800 @ 2.00GHz x 2
- *Memoria*: 4 GB Dual-channel DDR2 @ 332 Mhz
- *Gráficos*: 256 MB GeForce 9300M GS
- *Sistema operativo*: Ubuntu 12.04 LTS 64 bit

2.3. Herramientas software

2.3.1. Biblioteca OpenCV

Figura 2.3: Logo de OpenCV (Itseez, 2015).

OpenCV (*Open Computer Vision*, figura 2.3) es una biblioteca libre de visión por computador originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en enero de 1999, se ha utilizado en infinidad de aplicaciones: desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere el reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran

gama de áreas en el proceso de visión, como reconocimiento de objetos y facial, calibración de cámaras, visión estéreo y visión robótica. Actualmente se encuentra en su versión 2.4.10, lanzada en octubre de 2014, mientras que su versión beta 3.0.0 está también disponible desde noviembre de 2014.

El proyecto pretende proporcionar un conjunto de funciones fácil de utilizar y altamente eficiente. Esto se ha conseguido realizando su programación en código C y C++ optimizado, aprovechando asimismo las capacidades que poseen los procesadores multi-núcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integrado de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

En este proyecto se han utilizado una serie de funciones de OpenCV para la carga, almacenamiento, transformación y visualización de imágenes, así como segmentación sobre ellas. OpenCV tiene una amplia gama de funciones que trabajan sobre imágenes, así que ha sido una herramienta imprescindible para tratarlas junto con PCL.

2.3.2. Biblioteca PCL

Figura 2.4: Logo de PCL (Open Perception Foundation, 2015).

PCL (*Point Cloud Library*, figura 2.4) es una biblioteca de código abierto descrita por primera vez por Rusu and Cousins (2011) que contiene algoritmos para el procesamiento de nubes de puntos y para procesamiento de geometría en 3D, tareas que encontramos en visión por computador tridimensional. Esta biblioteca contiene además algoritmos para estimación de características, reconstrucción de superficies, registro de imágenes, ajuste de

modelos y segmentación. Está escrita en C++ y distribuida bajo licencia BSD. Su última versión disponible es la 1.7.1, lanzada en octubre de 2013.

Esta herramienta se ha usado, por ejemplo, para percepción en robótica en el filtrado de datos ruidosos, unión de nubes de puntos en 3D, segmentación de las partes relevantes de una escena, extracción de puntos clave, cómputo de descriptores para el reconocimiento de objetos basado en su apariencia geométrica y creación y visualización de superficies desde nubes de puntos.

Dentro de este proyecto, ha sido utilizada para ayudar en la segmentación de las imágenes. PCL nos permite crear, a partir de la imagen RGB y del mapa de profundidad, una nube de puntos que represente la escena actual, que posteriormente utilizamos para segmentar las zonas de la imagen donde se encuentra cada persona de la escena.

2.3.3. Biblioteca OpenNI

Figura 2.5: Logo de OpenNI (Structure Sensor Team, 2015).

OpenNI (*Open Natural Interaction*, figura 2.5) es una organización sin ánimo de lucro e impulsada por la industria centrada en la certificación y mejora de la interoperabilidad de la interfaz natural de usuario y la interfaz orgánica de usuario para dispositivos de interacción natural, las aplicaciones que usan estos dispositivos y el middleware que facilita el acceso y uso de tales dispositivos.

Dentro de esta organización, uno de los miembros principales era PrimeSense, la empresa encargada de crear la tecnología utilizada en Kinect. Asimismo, posee una serie de bibliotecas necesarias para utilizar el sensor Kinect en sistemas Linux, que han sido utilizadas en este proyecto.

2.3.4. Controladores Primesense

PrimeSense, figura 2.6, era una empresa que trabajaba con sensores 3D, hasta que en noviembre de 2013 fue comprada por Apple Inc. Es conocida

Figura 2.6: Logo de Primesense.

como la creadora de la tecnología que forma parte del sensor Kinect. También desarrolló los controladores para el mismo y el middleware NiTE, un software que analiza los datos recibidos por el hardware Kinect entre otros, y aplica las bibliotecas de gestos y detección de esqueleto desarrolladas por OpenNI. En el presente proyecto no se ha hecho uso de NiTE, pero sí de los controladores a bajo nivel para el sensor Kinect.

2.3.5. Plataforma de aprendizaje automático Weka

Figura 2.7: Logo de Weka (Machine Learning Group at the University of Waikato, 2015).

Weka (acrónimo en inglés para *entorno para el análisis del conocimiento de la Universidad de Waikato* (Hall et al., 2009), figura 2.7), es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Weka es software libre distribuido bajo la licencia GNU-GPL.

El paquete Weka contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. La versión original de Weka consistía en un *front-end* para modelar algoritmos implementados en diferentes lenguajes de programación, además de unas utilidades para preprocesamiento de datos desarrolladas en `C` para hacer

experimentos de aprendizaje automático. Esta versión original se diseñó inicialmente como herramienta para analizar datos procedentes del dominio de la agricultura, pero la versión más reciente basada en Java, que empezó a desarrollarse en 1997, se utiliza en muchas y muy diferentes áreas, en particular con fines docentes y de investigación. La versión estable más reciente de Weka es la 3.6.11, mientras que la versión beta, utilizada en este proyecto, es la 3.7.11, lanzada en abril de 2014.

Se ha utilizado Weka con el fin de analizar los datos extraídos de las diferentes colecciones de imágenes, y para la obtención de un clasificador de sexo basado en dichos datos.

2.3.6. Otro software

Se han empleado otras herramientas software para el desarrollo de este proyecto, pero cualquiera de ellas podía haber sido sustituida por una herramienta similar; hablamos de programas como el entorno de desarrollo o el encargado de la generación de documentación. Dichas herramientas son las que siguen:

- **Entorno de desarrollo de software o SDK**: Como entorno de desarrollo de software se ha utilizado *NetBeans*, un entorno de desarrollo de software gratuito y multiplataforma, en su versión 7.3.1.
- **Sistema de control de versiones**: Existen multitud de sistemas de control de versiones online para mantener un proyecto software, y el escogido en este proyecto ha sido *BitBucket*, que funciona de manera idéntica a *GitHub*, pero permite añadir opciones de privacidad a los proyectos.
- **Documentación**: Para la documentación del proyecto hemos utilizado dos herramientas; la primera de ellas, *Doxygen*, nos ha servido para generar un pequeño manual a partir de nuestro código, mientras que la segunda, *TexMaker*, ha sido empleada en la elaboración de esta memoria con LaTeX como lenguaje.

Capítulo 3

Análisis y diseño

Este capítulo se centra en explicar el diseño de la solución adoptada para el problema que afrontamos. En primer lugar, en la sección 3.1, comentamos la metodología seguida durante el desarrollo de este proyecto. Posteriormente, en la sección 3.2 describimos las clases que comprenden nuestra solución y la función principal de cada una. Por último, en la sección 3.3 mostramos el tiempo empleado en cada una de las etapas de este proyecto.

3.1. Metodología

Para desarrollar este proyecto hemos optado por el modelado de prototipos (Alemán-Flores, 2011). Este paradigma de ingeniería del software consiste en la recolección de requisitos que serán utilizados para construir un prototipo que los cumpla, el cual deberá ser evaluado y refinado a lo largo de múltiples iteraciones. El ciclo que sigue este modelo queda descrito en el diagrama 3.1.

Este modelo es común a muchas formas de ingeniería, y facilita la creación del modelo software a construir. Puede adoptar tres formas posibles: la de un prototipo que describa la interacción hombre-máquina, la de un prototipo que implemente un subconjunto de las funciones requeridas, o la de un programa existente que ejecute parte o toda la funcionalidad deseada pero que tenga otras características que deban ser mejoradas. Permite identificar claramente los requisitos del usuario y sirve como herramienta para identificar los requisitos del software.

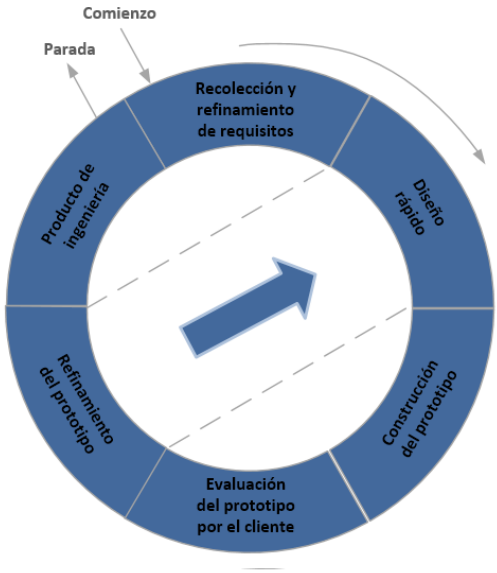

Figura 3.1: Diagrama que resume el ciclo de vida de un producto desarrollado mediante prototipos.

El motivo de la elección de este paradigma se debe a que se ajusta bastante bien a nuestro problema. Inicialmente conocemos los requisitos que ha de cumplir, pero no sabemos qué requisitos tiene el software. De ahí que comenzáramos planteando un prototipo, que fuimos refinando sucesivamente hasta obtener el sistema finalizado.

3.2. División en clases

Hemos dividido la estructura de este sistema en dos clases principales. La primera de ellas, denominada `CounterPCL`, implementa la mayor parte del contador de personas. Se encarga de cargar las imágenes y realizar su preprocesado, calcular el modelo de fondo y detectar las personas dentro de cada imagen.

La otra clase principal es `PeopleClass`. Esta clase almacena toda la información referida a las personas que tenemos actualmente trazando trayectorias en la escena. En cuanto a funcionalidad, se encarga de realizar el conteo y el seguimiento de cada una de las personas y de mostrar las trayectorias que dichas personas siguen.

Estas dos clases se comunican una vez se han detectado las personas dentro del fotograma. `CounterPCL` calcula la posición en la que se encontraba cada persona en la imagen anterior (el proceso se explicará con más detalle en la sección 4.4) y le envía estas dos posiciones (la posición actual y donde se encontraba en la última imagen) a `PeopleClass`, la cual buscará a la persona por su posición anterior y le añadirá una posición nueva. Esta clase, una vez reciba todas las posiciones nuevas de la imagen actual, actualizará todas las trayectorias y contabilizará a aquellas personas que ya no se encuentren en la escena, y las mostrará por pantalla sobre la imagen que `PeopleClass` le pasa. En esto consiste el funcionamiento de nuestro sistema a grandes rasgos; veremos todo esto con mucho más detalle en el capítulo 4.

3.3. Temporización

Originalmente, se planteó una temporización para el desarrollo de este proyecto, la cual podemos ver en la gráfica 3.2.

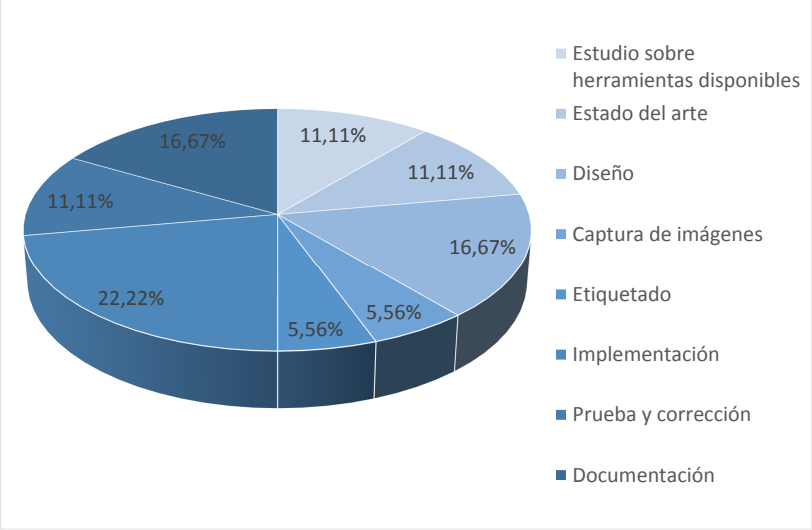

Figura 3.2: Temporización original para las diferentes fases del proyecto.

Una vez comenzamos a desarrollar este proyecto, algunas de estas franjas de tiempo fueron cambiando, ya que ciertas partes requerían una mayor

cantidad de tiempo del que habíamos asignado inicialmente. El tiempo real empleado aproximadamente en cada una de las partes de este proyecto se representa en la gráfica 3.3.

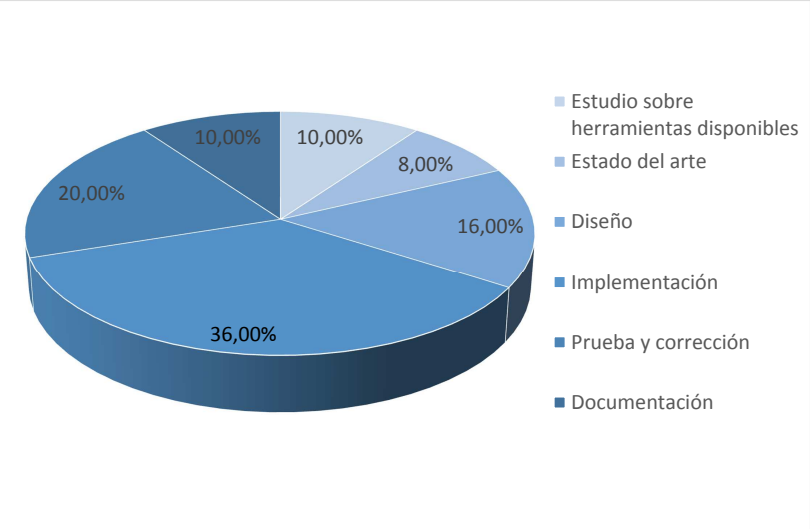

Figura 3.3: Temporización aproximada de las diferentes fases del proyecto.

Como podemos observar, el desarrollo del sistema ha sido la etapa que más tiempo ha empleado, seguido de las pruebas y correcciones al sistema. Este segundo apartado se ha llevado más tiempo del supuesto inicialmente debido a que incluimos las pruebas realizadas con los clasificadores para sexo dentro de este apartado. Además, nótese que tenemos dos franjas adicionales en la temporización original: captura de imágenes y etiquetado. Inicialmente se pensaba capturar las imágenes directamente con el sensor, pero como veremos en el apartado 4.1, la adquisición de imagen consistió en utilizar una serie de imágenes tomadas anteriormente por los tutores, lo cual nos hizo obviar estos dos pasos.

Capítulo 4

Implementación del sistema

El capítulo actual describe los diferentes pasos en los que podemos dividir la implementación del proyecto. Primero, en la sección 4.1, hablaremos de la adquisición de imagen con Kinect y de los conjuntos de imágenes utilizados con nuestro sistema. Luego pasaremos al preprocesado de imagen en la sección 4.2, que incluye el proceso de cálculo del modelo de fondo. Posteriormente, en la sección 4.3 mostraremos la detección de personas y del tratamiento que hacemos con *blob* detectado para comprobar si es una persona o no. En la sección 4.4 tratamos el conteo de personas y el método utilizado para seguir las personas que transcurren en la escena. Por último hablamos de la detección del sexo en la sección 5.2, de los valores que utilizamos para ello y de la construcción de un clasificador para detectarlo.

4.1. Adquisición de imagen

A la hora de implementar y probar nuestro sistema tenemos a nuestra disposición dos posibles flujos de datos entrantes. Podemos tomar las imágenes directamente desde Kinect, aunque también se han proporcionado una serie de imágenes tomadas con Kinect y divididas en sesiones. Capturadas por los tutores y etiquetadas manualmente, constituyen una fuente de referencia excelente a la hora de comparar el funcionamiento de nuestro programa con el de un contador manual.

Estas sesiones han sido grabadas a lo largo de tres días, y cada día dividido en dos sesiones, correspondientes a la entrada y salida a una de las aulas de la

Universidad de Las Palmas de Gran Canaria, para un total de seis sesiones. Se ha hecho uso de una de las sesiones principalmente para la implementación de nuestro sistema, y posteriormente se ha comprobado el funcionamiento del sistema con el resto de sesiones. Este conjunto de imágenes, denominado `DEPTHVISDOOR`, junto con sus anotaciones, puede encontrarse en (Castrillón-Santana et al., 2014b). Podemos encontrar un ejemplo de las imágenes en este conjunto en la figura 4.1.

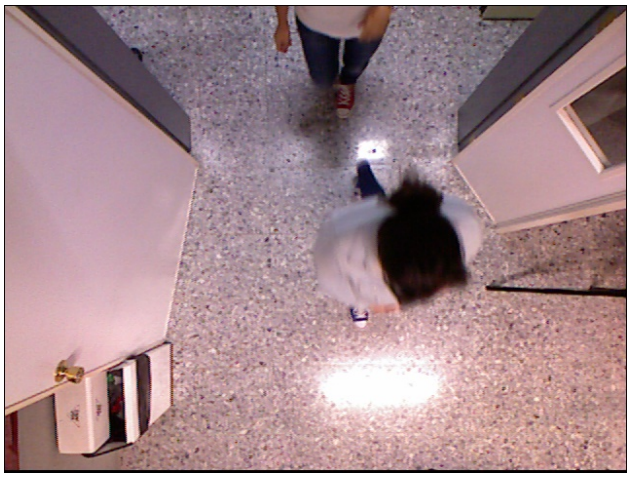

Figura 4.1: Ejemplo de una de las imágenes del conjunto de datos usado.

La geometría del espacio de captura de imágenes se describe en la figura 4.2. Las especificaciones aportadas por Microsoft para el sensor Kinect determinan que su campo de visión vertical es de 43º; con este dato podemos calcular la longitud r mediante la expresión:

$$r = 2\tan\left(\frac{43^{\circ}}{2}\right)(T-h) \tag{4.1}$$

en la cual T es la distancia del sensor al suelo y h la altura de la persona. Para una persona con una altura promedio $h = 1{,}75m$ y un sensor situado a $T = 3m$ del suelo, obtenemos una longitud $r = 0{,}98m$. Si tenemos en cuenta una velocidad de $1{,}4m/s$, el sensor capturará una media de 18 imágenes de cada persona atravesando el campo de visión, asumiendo una frecuencia de captura entre 15 y 25 imágenes por segundo; información suficiente para el conteo y etiquetado de cada persona.

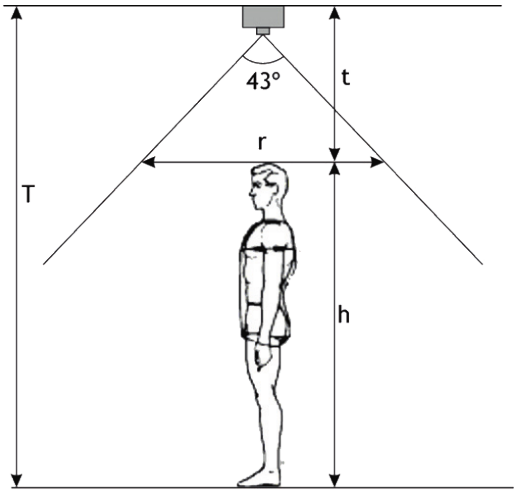

Figura 4.2: Configuración de la captura de imágenes con Kinect.

4.2. Preprocesado de imagen

4.2.1. Cálculo de la nube con el modelo de fondo

Para lograr la separación de blobs dentro de cada imagen, una técnica frecuente en visión por computador es la sustracción de fondo (Bradski and Kaehler, 2008, p. 281). Como su nombre indica, consiste en extraer el fondo de cada imagen para su posterior procesamiento; en nuestro caso, para reconocer las personas que tenemos en cada imagen. Se usa ampliamente cuando el objetivo es diferenciar objetos móviles que entran dentro del campo de visión de una cámara fija, lo que se corresponde con una parte de nuestro problema.

Al aplicar el proceso de extracción del fondo, primero hemos de calcular un modelo del fondo de la escena. Una vez tengamos este modelo, comparándolo con cada imagen perteneciente a la secuencia a analizar, podremos extraer fácilmente aquellas partes de dicha imagen que no pertenecen al fondo. Existen diversas técnicas de cálculo del modelo de fondo, siendo las siguientes las más comunes:

- **Diferencia de fotogramas**. El más simple de los métodos de extracción de fondo consiste en calcular la diferencia entre dos imágenes consecutivas y tomar cualquier diferencia significativa como *foreground*. Este método, aunque sencillo, sólo proporciona regiones en las que se aprecia movimiento, y resulta demasiado simple para nuestras necesidades.

- **Fondo promediado**. Este método calcula, para todo píxel de la imagen, la media y la desviación estándar con respecto a un número de fotogramas determinado, y esto constituye nuestro modelo de fondo. Resulta más robusto y fiable que el método anterior, y su cálculo no resulta muy costoso.

- **Algoritmo *Watershed***. Utilizado generalmente cuando no se tiene una imagen de fondo, el algoritmo *Watershed* convierte las líneas de una imagen en "montañas" y áreas uniformes en "valles" que se utilizan para segmentar objetos. El algoritmo primero toma el gradiente de la imagen, formando valles en los puntos donde no hay textura, y montañas donde hay líneas en la imagen. Después, inunda los valles a partir de puntos especificados o bien por el usuario o algorítmicamente hasta que las regiones se encuentran. Si dos regiones se unen, se considera que forman parte del mismo objeto. De esta forma, todos los valles conectados al punto especificado forman parte del objeto a segmentar. Aunque es más costoso que el promediado de fondo, es más versátil. Sin embargo, requiere supervisión, al menos en su idea inicial, para poder determinar, una vez segmentada la imagen, qué parte de la misma es el fondo.

- **Segmentación *Mean-shift***. Este algoritmo intenta separar los diferentes elementos dentro de una imagen. Para ello es capaz de detectar picos que representan los valores de cada segmento, determinar cuántos son y agrupar o etiquetar los píxeles según el centro al que se parezcan más, para luego realizar la segmentación. Aunque la idea detrás del algoritmo es sencilla, este algoritmo tiene un coste computacional elevado. Sin embargo, es ampliamente usado en aplicaciones de visión por computador. De la misma forma que el algoritmo anterior, la versión inicial del algoritmo requiere supervisión para determinar qué parte de la imagen segmentada corresponde al fondo.

Nuestro sistema utiliza imágenes obtenidas de un sensor en una posición fija y cenital, por lo que no necesitamos un modelado del fondo de la escena complejo. Con un promediado de las primeras imágenes de la secuencia, podemos extraer dos matrices con el modelo de fondo, una de ellas con la imagen RGB del fondo y la otra con los datos de profundidad. Gracias al uso de las herramientas proporcionadas por PCL, podemos unir estas dos matrices en una nube de puntos para tener el modelo de fondo en una sola entidad, como vemos en la figura 4.3. En dicha imagen podemos ver los datos de profundidad a la izquierda, mientras que a la derecha tenemos los datos RGB.

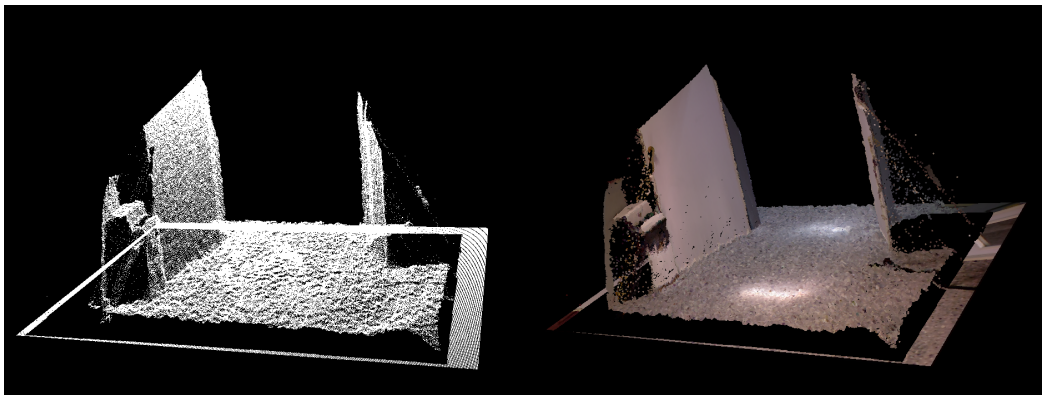

Figura 4.3: Nube generada usando PCL que incluye el modelo de fondo calculado para una de las sesiones.

4.3. Detección de personas

La detección y el conteo de personas son dos procesos consecutivos donde se analiza todo lo que se encuentra en la imagen y no forma parte del fondo. Para dar a conocer en líneas generales cómo se realizan estos procesos de una forma simplificada, tenemos el pseudocódigo 1, el cual muestra el proceso de análisis de las imágenes una vez se ha calculado el modelo de fondo.

Algoritmo 1 Resumen del proceso de detección y conteo de personas.

```
1: while hay una imagen disponible do
2:     Leemos la imagen de la lista o de la Kinect
3:     Calculamos diferencia entre imagen y modelo de fondo, obteniendo
       foreground
4:     Convertimos el foreground a una nube de PCL
5:     if hay un mínimo de píxeles visibles then
6:         Calculamos normales del foreground
7:         Obtenemos contornos del foreground
8:         for cada contorno encontrado do
9:             for cada blob detectado en el contorno do
10:                Obtenemos características del blob
11:                if características indican que es persona then
12:                    Realizamos seguimiento de la persona
13:                    Contabilizamos la misma
14:                else
15:                    Pasamos al siguiente contorno
```

```
16:           end if
17:         end for
18:       end for
19:     else
20:       Pasamos a la siguiente imagen
21:     end if
22:     Mostramos imagen con personas detectadas y trayectorias
23: end while
```

4.3.1. Procesado para cada *blob*

Con el modelo de fondo ya calculado, podemos extraer fácilmente cualquier elemento diferente al fondo capturado por la cámara. Realizamos este proceso a la vez que convertimos la imagen a una nube de puntos; al recorrer los puntos de una imagen y calcular su proyección en la nube, comprobamos si cada punto quedaría en el fondo y, si no es así, lo añadimos a una nueva nube que contendrá todo lo que no forme parte del fondo de la misma, también llamado *foreground*. El método es similar al utilizado en el promediado para el cálculo del fondo.

Una vez realizado este proceso, el siguiente paso consiste en analizar la parte de la imagen detectada como *foreground*. Aquí hemos de tener en cuenta las diferentes situaciones que se pueden dar:

- Si encontramos que el *foreground* contiene un número de píxeles distintos de cero muy bajo (un 3 % del tamaño original de la imagen), consideramos que no hay suficiente información para que una persona se encuentre presente y continuaremos con el conteo en el siguiente fotograma.
- En caso contrario, podemos tener una persona, varias o algún objeto que se haya dejado en la escena después del cálculo del modelo de fondo. En este caso, buscaremos personas dentro del foreground intentando localizar sus cabezas, como se explicará a continuación.

Una vez extraído el *foreground* de la imagen y comprobado que hay suficientes píxeles diferentes de cero en él, comprobaremos si esos píxeles corresponden a una persona o no. Para ello intentaremos localizar la cabeza de la

persona, ya que es la parte del cuerpo humano que queda más cerca del sensor de captura en la configuración descrita en el apartado 4.1 de adquisición de imagen.

En primer lugar hemos de localizar la zona de la imagen donde se encuentra la persona a analizar. Para ello utilizamos la función de OpenCV `findContours`, la cual nos devuelve una lista con los contornos presentes en la imagen. Inmediatamente después comenzamos a recorrer esta lista de contornos, calculando el área interna para cada uno de ellos. Si dicha área comprende más de un 3 % del tamaño total de la imagen, continuamos su proceso. Este margen nos permite discriminar aquellos contornos demasiado pequeños para tratarse de una persona, que serían considerados como ruido. El resultado de este proceso se puede apreciar en la figura 4.4.

Figura 4.4: Resultado de la eliminación del fondo y del ruido restante en la imagen. En la parte superior tenemos el fondo eliminado, mientras que en la inferior se ha eliminado tanto fondo como ruido.

Después de esto, lanzamos un bucle para recorrer todas las posibles personas dentro de dicho contorno. Durante el proceso de desarrollo del sistema, encontramos casos donde un mismo blob podía contener más de una persona, como vemos en la figura 4.5; después de eliminar el fondo, si dos personas

Figura 4.5: Varias personas en el mismo blob dificultan el conteo.

están lo suficientemente próximas, la operación `findContours` detectará un solo contorno que incluya ambas. Debido a esto, hemos optado por recorrer cada contorno en múltiples ocasiones, hasta que no podamos encontrar más posibles personas dentro del mismo. En el apartado 4.3.2, se explicará el proceso por el que identificamos cada persona dentro de este bucle.

El siguiente paso consiste en extraer una máscara con el contenido de este contorno, sobre la que calcularemos la distancia de los puntos al suelo. Para ello, en primer lugar acumulamos la altura de cada uno de los puntos del modelo de fondo que entran dentro de la máscara, y luego dividimos entre el número de puntos para obtener la altura media del suelo. Tras esto, recorremos los puntos del *foreground* correspondientes a esta máscara comparando su profundidad con el valor de la altura del suelo calculado anteriormente; el valor máximo de esta diferencia nos dará la altura de la persona. Este valor nos ayudará a llevar a cabo la separación de la cabeza en el siguiente apartado.

4.3.2. Separación de la cabeza

El proceso descrito en este apartado se encuentra resumido en el pseudocódigo 2. En este punto ya hemos conseguido la altura de la posible persona dentro de la imagen. Utilizando este dato, podemos seleccionar una parte de los datos correspondiente a un cierto porcentaje de la altura como la cabeza de la misma. Teniendo en cuenta el canon antropométrico de Richer y Langer (Horcajada-González, 2011) se considera que el cuerpo humano mide unas siete veces y media el tamaño de la cabeza, como vemos en la figura 4.6; y que dentro de esta encontramos que la parte superior de la misma, que será vista desde una posición cenital, corresponde a un 40 % del total de la altura de la cabeza.

Algoritmo 2 Pseudocódigo que resume el proceso de separación de la cabeza.

```
1: for cada blob detectado en el contorno do
2:    Obtenemos distancia al suelo
3:    Obtenemos altura máxima del blob
4:    Obtenemos centroide alrededor de la altura máxima
5:    Extraemos blob a una imagen como posible cabeza
6:    if el centroide está en el ROI then
7:       Dilatamos la cabeza y la añadimos a una máscara para el siguiente
         fotograma
8:       Calculamos circularidad de la cabeza
9:       Creamos nube sólo con la cabeza
10:      Calculamos normales para dicha nube
11:      Contabilizamos a qué coordenada corresponden la mayoría de nor-
         males
12:      Determinamos si está entrando o saliendo del ROI
13:      if con circularidad y normales detectamos que es persona then
14:         Realizamos el seguimiento de la persona
15:      end if
16:   else
17:      Pasamos al siguiente contorno
18:   end if
19: end for
```

Con estas proporciones en mente, podemos calcular qué porcentaje de la altura total corresponderá a la parte superior de la cabeza: alrededor de un 95 %. Utilizando este valor, podemos separar la parte del blob que corresponde a la cabeza para determinar si se trata de una persona o no; en la figura 4.7 mostramos de forma visual la altura que representa. Este valor, sin embargo, no ha sido fijado desde el principio del proceso de desarrollo; se ha realizado un pequeño análisis sobre la relación entre la altura que tomamos como parte superior de la cabeza y otros parámetros que se verán en los siguientes apartados. Además de eso, calculamos también el centroide del blob que contiene la cabeza para su posterior uso.

Después del proceso anterior, tenemos una máscara que contiene tan sólo la parte superior de la cabeza, suponiendo que sea una persona. Para continuar el proceso de detección, primero comprobamos que la persona está dentro del área de interés de la escena. Este área de interés (o ROI, *Region*

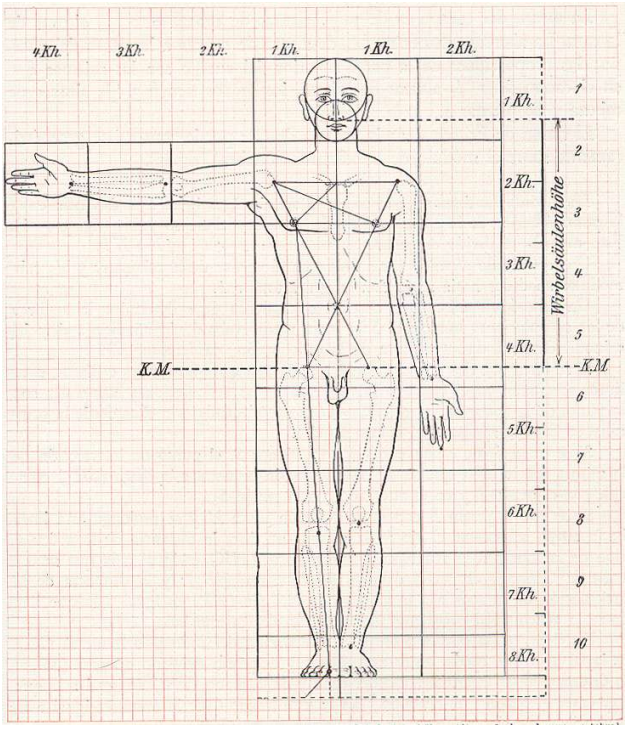

Figura 4.6: Canon antropométrico de Richer y Langer.

Figura 4.7: Porción de la cabeza de la persona desde una vista frontal que capturamos con Kinect y que tenemos en cuenta para realizar la segmentación.

Of Interest como se le llamará de ahora en adelante) corresponde al 70 % central de la imagen, contando además con zonas donde se detectará la entrada y salida del mismo. Si el centroide se encuentra dentro de nuestro ROI, continuaremos su proceso.

Aplicamos sobre la máscara una operación de cierre (también conocida como *closing*). Para explicar en qué consiste esta operación, es necesario saber en qué consisten las operaciones de erosión (más conocida por su nombre en inglés, *erosion*) y dilatación (*dilation* en inglés). La erosión y la dilatación son operaciones morfológicas sobre imágenes (generalmente binarias), y ambas toman dos parámetros: la imagen a tratar y un conjunto de coordenadas conocido como *kernel*. La dilatación añade píxeles a los bordes de los objetos encontrados en una imagen, mientras que la erosión elimina píxeles en los bordes. El número de píxeles añadido o eliminado depende del tamaño y forma del *kernel* anteriormente mencionado, que determina una vecindad para cada píxel. En estas operaciones, el estado de cada píxel depende de los estados de la vecindad del mismo antes de realizar la operación. Así, para la dilatación, el valor de un píxel en la salida es el valor máximo de todos los píxeles en su vecindad (para una imagen binaria, si cualquiera de los píxeles en la vecindad tiene valor 1, en la salida tendremos un 1) mientras que para la erosión el valor de un píxel en la salida es el valor mínimo de todos los píxeles en su vecindad (para una imagen binaria, si cualquiera de los píxeles en la vecindad tiene valor 0, en la salida tendremos un 0).

La operación de cierre consiste en realizar una dilatación seguida de una erosión sobre la misma imagen utilizando el mismo *kernel* para ambas operaciones. Este procedimiento suele utilizarse para eliminar huecos dentro de una imagen, utilidad que hemos aprovechado. En nuestro caso, una vez tenemos un *blob* que representa la parte superior de una cabeza, le aplicamos una operación de cierre para eliminar los huecos dentro de la misma y suavizar sus bordes, como vemos en la figura 4.8. Este *blob* retocado será añadido a una máscara, que ayudará a excluir esa zona de la imagen y buscar otras cabezas dentro de la misma imagen.

Es en este punto cuando consideramos que tenemos suficiente información para realizar una estimación fiable de si se trata de una persona o no. El cráneo humano tiene forma elíptica, por lo que en nuestra máscara, si se trata de la cabeza de una persona, nos encontraremos una elipse. Para determinar qué forma es la que encontramos dentro de la máscara, se investigó primero el uso de los momentos invariantes de Hu.

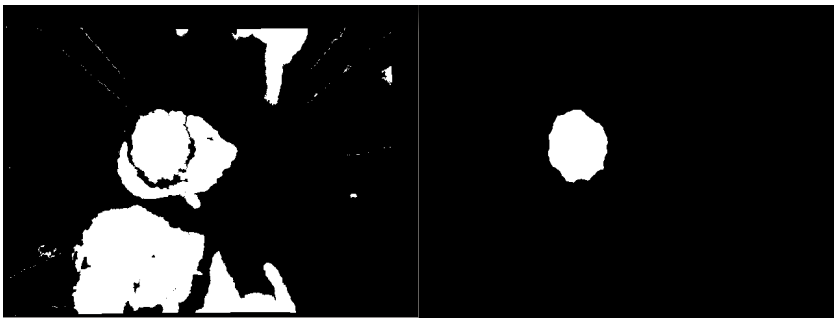

Figura 4.8: Resultado de aplicar la operación de closing sobre una cabeza.

Momentos invariantes de Hu

En procesamiento de imagen y visión por computador, un momento de imagen (Bradski and Kaehler, 2008, p. 252) consiste en una media ponderada de las intensidades de los píxeles de dicha imagen o una función de dichos valores, escogidos porque presentan alguna propiedad o interpretación interesante. Suelen usarse para describir objetos tras una segmentación. En general, se define un *momento (p, q)* de imagen de la siguiente forma:

$$m_{p,q} = \sum_{i=0}^{n} I(x,y) x^p y^q \tag{4.2}$$

En esta ecuación p es el orden x y q es el orden y, donde *orden* significa la potencia a la que elevamos cada componente en la suma. El sumatorio recorre todos los píxeles de un área de la imagen o un contorno. Por lo tanto, si p y q son ambos iguales a cero, el momento m_{00} corresponde al área de la imagen, o al número de píxeles del contorno.

Los *momentos centrales* funcionan de la misma manera que los momentos descritos anteriormente, pero los valores de x e y usados en la fórmula quedan desplazados por los del centroide de la imagen:

$$\mu_{p,q} = \sum_{i=0}^{n} I(x,y)(x - x_{\text{avg}})^p (y - y_{\text{avg}})^q \tag{4.3}$$

donde $x_{\text{avg}} = m_{10}/m_{00}$ y $y_{\text{avg}} = m_{01}/m_{00}$.

Sin embargo, los momentos obtenidos de este cálculo no son los mejores parámetros para realizar comparaciones en muchos casos; en concreto, sería

más conveniente utilizar *momentos normalizados*, de forma que objetos de la misma forma pero de tamaños diferentes den valores similares. Estos momentos son similares a los momentos centrales excepto que están divididos por una potencia de m_{00}:

$$\eta_{p,q} = \frac{\mu_{p,q}}{m_{00}^{\frac{p+q}{2}+1}} \tag{4.4}$$

Finalmente, a partir de combinaciones lineales de los anteriores, podemos calcular los momentos invariantes de Hu, cuya definición es la siguiente:

$$\begin{aligned}
h_1 &= \eta_{20} + \eta_{02} \\
h_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
h_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
h_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
h_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + 3\eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2) \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + 3\eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \\
h_6 &= (\eta_{20} - \eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})) \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
h_7 &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + 3\eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \\
&\quad + (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})(3(\eta_{30} + 3\eta_{12})^2 - (\eta_{21} + \eta_{03})^2)
\end{aligned} \tag{4.5}$$

Los momentos de Hu, descritos por primera vez en (Hu, 1962) son llamados invariantes porque una forma descrita con ellos tendrá los mismos momentos al ser trasladada, escalada y rotada, mientras que al ser reflejada uno de dichos momentos cambiará su signo. Debido a estas características, se suelen utilizar en el reconocimiento de patrones y objetos, y por el mismo motivo fueron considerados para realizar la detección de cabezas.

La idea detrás del uso de los momentos invariantes de Hu consiste en realizar una ejecución de la separación de cabezas y extraer dichos momentos sobre ellas utilizando las funciones `moments` y `HuMoments` de OpenCV, que calculan los momentos centrales y los momentos de Hu respectivamente. Calculando la media y la desviación típica de una muestra significativa de ellos podemos utilizarlos para clasificar las personas que nos encontramos en nuestro ROI. Sin embargo, tras extraer varias muestras de momentos de Hu, comprobamos que su dispersión no se ajusta a una distribución normal, con lo cual no hemos podido realizar la detección de cabezas con este método.

Circularidad

Después de realizar un análisis sobre los momentos de Hu en nuestro conjunto de datos, buscamos una solución alternativa para identificar las posibles cabezas como elipses. La circularidad (Sonka et al., 2008, p. 374) de un objeto es una cantidad numérica adimensional que nos indica cuán compacta es dicha forma. Queda definida por la siguiente fórmula:

$$circularidad = \frac{4 \cdot \pi \cdot área}{perímetro^2} \tag{4.6}$$

Para comprender mejor las dimensiones de esta medida, pongamos un ejemplo. Calcularemos la circularidad de dos figuras, la de un cuadrado cuyo lado mide la unidad y la de un círculo cuyo radio tiene el mismo valor. Para nuestro cuadrado, la circularidad tiene el valor $circ_{cuadrado} = \frac{4 \cdot \pi \cdot 1}{(1+1+1+1)^2} = 0{,}79$, mientras que para un círculo, el valor de esta medida es $circ_{círculo} = \frac{4 \cdot \pi \cdot \pi \cdot r^2}{(2 \cdot \pi \cdot r)^2} = 1$, el valor máximo para esta medida. Al calcular la circularidad de cada cabeza que detectamos, cuanta más cercana sea a la unidad, mayor posibilidad tendremos de que sea realmente una cabeza, ya que habremos detectado una forma compacta.

Esta medida, aunque más simple que los momentos de Hu, es también más fácil de calcular, y no es tan dependiente de las diferentes formas de las cabezas. Por ejemplo, con los momentos de Hu podemos obtener variaciones bastante grandes al encontrarnos una cabeza más alargada, o de lado, o con algún adorno en el pelo, mientras que con la circularidad, si encontramos una forma compacta, obtendremos resultados positivos. Además, los momentos de Hu por lo general suelen dar valores de órdenes de magnitud bastante pequeños, lo que hace poco intuitiva su comparación y uso, mientras que la circularidad da un valor comprendido en el rango $[0, 1]$, lo cual hace mucho más fácil su comprensión y uso en comparación. Esto la hace más adecuada para nuestro problema.

Medida de normales

La circularidad es un valor que nos ayuda a discriminar las posibles cabezas, pero por sí mismo no es suficiente para identificarlas. Aprovechando que tenemos también una nube con la cabeza separada, podemos calcular las normales de cada uno de los puntos de dicha nube. Estas normales nos

pueden ayudar a determinar si lo que tenemos en la nube es una cabeza o no. Como vemos en la imagen 4.9, las normales de la parte superior de una cabeza tienen una componente vertical mayor que el resto. Utilizando esta información de la cabeza y contando cuántas normales dentro de la nube tienen una componente vertical mayor a las otras, tenemos una medida que nos es de utilidad a la hora de identificar personas en cada imagen.

Figura 4.9: Normales de los puntos de la cabeza calculadas para dos personas.

Confirmación de cabeza presente en la imagen

Las dos medidas anteriores son las que utilizamos principalmente para la detección de cabezas. Estas medidas se combinan en una ponderación para determinar si el blob detectado es una cabeza o no. Dicha ponderación otorga un mayor peso a la medida de las normales, debido a que, aunque la mayoría de los *blobs* asociados a las cabezas presentan una forma compacta cercana a la de una elipse o un círculo, existen elementos como coletas o lazos que en algunos casos hacen que la medida de circularidad falle al reconocer una persona. Encontramos un ejemplo de este caso en la figura 4.10, donde obtendremos una circularidad baja debido a la coleta en el pelo de la persona que estamos detectando.

El valor para el que se acepta un blob como cabeza ha sido calculado aplicando esta ponderación a los valores para las dos medidas mostradas en las secciones anteriores. Ambas medidas han de tener un valor mínimo por separado, y su ponderación ha de superar también un umbral para poder identificar el blob como una cabeza. Estos valores han sido obtenidos empíricamente a través del análisis de las diferentes sesiones de imágenes de las

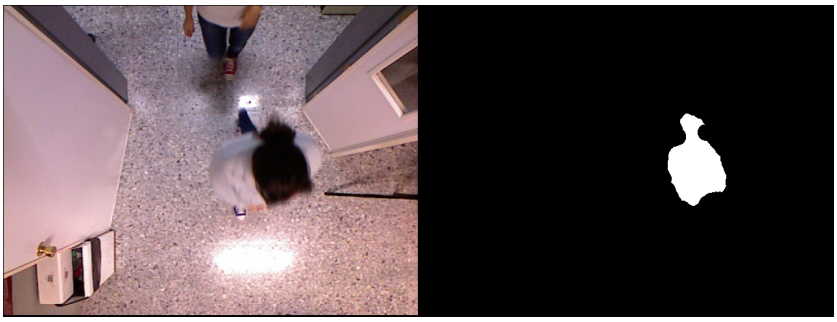

Figura 4.10: Una cabeza con una forma inusual hará más difícil la detección.

que disponemos, de forma experimental. Podemos ver la fórmula final para la poderación de los valores en la ecuación 4.7.

$$0{,}65 \cdot normales + 0{,}35 \cdot circularidad > 0{,}7 \tag{4.7}$$

Sin embargo, esta detección no es perfecta. Dos personas con alturas similares y con una proximidad suficiente pueden pasar por la misma persona debido a que formarán parte del mismo blob, como podemos ver en la imagen 4.11. En concreto, en dicha imagen las personas pasan con las cabezas unidas y el sistema no es capaz de reconocerlas individualmente. Además, existen ciertos problemas con casos donde la cabeza no presenta una forma tan circular, por ejemplo el caso de una persona que lleve el pelo recogido en una coleta. Este problema se ha aliviado otorgando a la medida de las normales un peso mayor en la ponderación anterior; no obstante, hemos encontrado casos en que esta corrección no es suficiente.

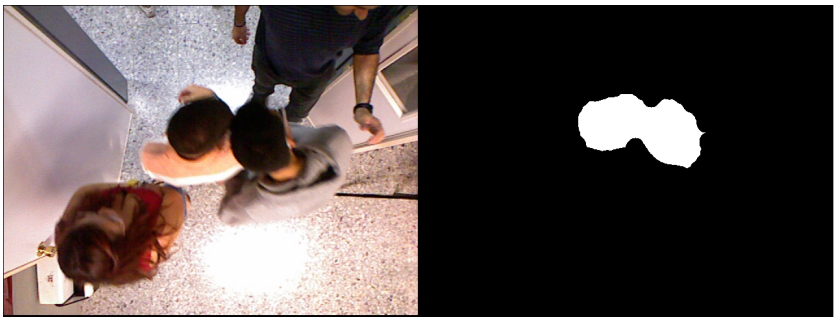

Figura 4.11: Dos personas durante una de las sesiones pasan con sus cabezas unidas.

Es por esto que se toma un cierto margen de error para la detección de personas. Aunque nuestro discriminador de cabezas indique que lo que

estamos analizando en la imagen actual no se trata de una cabeza, si es la primera entidad dentro del *foreground* que analizamos, la consideraremos una cabeza "extraña", siempre y cuando su circularidad no sea extremadamente baja. De esta forma, contamos como cabezas extrañas aquellas personas cuya cabeza puede no presentar una forma tan compacta como lo que se espera.

Estos valores (altura de la persona, tamaño de su cabeza, circularidad de la misma y medida de las normales) serán volcados en un fichero aparte para su posterior análisis y clasificación del sexo de las personas a las que se les han calculado, lo cual veremos con más profundidad en el apartado 5.2.

4.4. Conteo de personas

A la hora de explicar el conteo de personas dividimos el proceso en dos partes, seguimiento y conteo propiamente dicho. La estructura general de esta parte del proyecto queda plasmada en el pseudocódigo 3.

Algoritmo 3 Pseudocódigo que resume el proceso de seguimiento y conteo.

```
1: if con circularidad y normales detectamos que es persona then
2:    Añadimos persona a las detectadas en este contorno
3:    if persona se solapa con alguna de la imagen anterior then
4:       Actualizamos su posición
5:    else
6:       Insertamos en lista de personas de esta imagen
7:    end if
8: else
9:    if es la primera entidad en el contorno then
10:      Añadimos persona a las detectadas en este contorno
11:      if persona se solapa con alguna de la imagen anterior then
12:         Actualizamos su posición
13:      else
14:         Insertamos en lista personas de esta imagen
15:      end if
16:   end if
17: end if
18: Cuando terminamos con un frame, llamamos a update()
```

4.4.1. Seguimiento de *blobs*

Cuando hemos aislado satisfactoriamente una persona dentro de una imagen, pasamos a contabilizarla. Para poder contabilizarla correctamente, en primer lugar comprobamos si se encontraba en la imagen anterior. Esto lo conseguimos con una máscara que actualizamos en un fotograma cada vez que encontramos una cabeza nueva. Una vez terminamos con una imagen, guardamos esta máscara como la máscara del fotograma anterior en la secuencia, y será con la que contrastemos las personas que encontremos en la siguiente imagen. Para comprobar si estaba o no una persona en la imagen anterior, calcularemos la intersección entre la máscara con la cabeza actual y la máscara calculada en la imagen previa; si esta intersección no es cero, podemos decir que en dos fotograma consecutivos hemos encontrado a la misma persona, y pasamos a actualizar la posición de dicha persona. Esto es posible porque, como se ha explicado anteriormente, debido a la velocidad de paso de una persona debajo de la cámara existen solapes entre detecciones sucesivas de la misma persona, lo cual nos permite realizar el seguimiento de esta manera.

Si la persona se encontraba en la imagen anterior, hemos de realizar un análisis de la máscara de dicho fotograma para averiguar a cuál de las personas que estamos siguiendo hemos de actualizarle la posición. Esto lo conseguimos obteniendo los contonos pertenecientes a cabezas almacenadas del fotograma anterior, y comprobando cuál de ellos comprende el solape que calculamos anteriormente. Una vez obtenido el contorno, calculamos su centroide, por el cual identificaremos a la persona a actualizar dentro de nuestra lista de personas.

Obtenidos todos los valores significativos de la persona (altura del *blob*, área de la cabeza, circularidad, medida de normales, posición en el fotograma anterior) podemos actualizar la posición de la misma en una lista de personas que mantenemos gracias a la clase `PeopleClass`. Esta clase ha sido creada con el propósito de mantener una lista de transeúntes actualizable imagen por imagen y que almacena tanto los datos extraídos de la persona como su trayectoria (sentido de entrada y salida de la escena, últimas cinco posiciones). Cada persona que detectamos en una imagen va siendo almacenada en una lista temporal, que actualizará la lista global cuando el análisis de esta imagen haya concluido. Con la información de esta lista global, mostraremos por pantalla las detecciones en este fotograma y se realizará el conteo como será explicado en el siguiente apartado.

Si tenemos una trayectoria de una persona aislada con un número de posiciones registradas por segundo elevado, el contador detectará siempre un solape entre dos posiciones consecutivas de la trayectoria. Este es el caso más común, donde la trayectoria de una persona no se ve obstaculizada y el seguimiento puede realizarse correctamente.

Sin embargo, el seguimiento de las personas no es tan inmediato como localizar la posición anterior de la persona y actualizarla con la nueva posición calculada mediante el solape de ambas. Encontramos casos donde la persona se mueve por la escena con una velocidad superior a la de otras personas o que quede dentro de un blob con otras personas y no sea posible su separación, y debido a la gran diferencia de las dos posiciones de la persona en dos posiciones consecutivas, el solape puede no llegar a ocurrir, como vemos en la figura 4.12.

En estos casos, el procedimiento `update` que une la lista temporal con la global analiza las posiciones anteriores por las que ha pasado una persona y, calculando su trayectoria y su velocidad utilizando el método de mínimos cuadrados para extraer la trayectoria a través de las posiciones anteriores, es capaz de actualizar la posición de una persona que de inmediato no podíamos conocer mediante los solapes entre fotogramas.

4.4.2. Conteo de *blobs*

Para el problema del conteo de personas, se ha designado un ROI dentro de la imagen, como se ha comentado en el apartado 4.3.2, que utilizamos para determinar de manera sencilla por dónde entran y salen las personas de la escena. Este ROI comprende el 70 % central de la escena. Además, tenemos unas bandas de entrada y salida, con las que comenzamos a detectar el instante en que una persona está entrando o saliendo. Dichas bandas ocupan cada una un 10 % de la imagen original, y se encuentran en los bordes del ROI, en la parte interna, como se puede ver en la figura 4.13. La escena que analiza este sistema consiste en una puerta de entrada a un aula. Tal y como tenemos colocada nuestra cámara, la puerta queda en la parte superior de cada imagen, y los fotogramas han sido tomados desde dentro de la misma. Asimismo, consideramos que cualquier persona que se desplace de la parte superior de la escena a la parte inferior está entrando en el aula y será contado en nuestro sistema como IN, mientras que si se desplaza de la parte inferior a la superior está saliendo y se etiqueta como OUT. Las entradas y salidas laterales son posibles; como hemos indicado anteriormente, la puerta

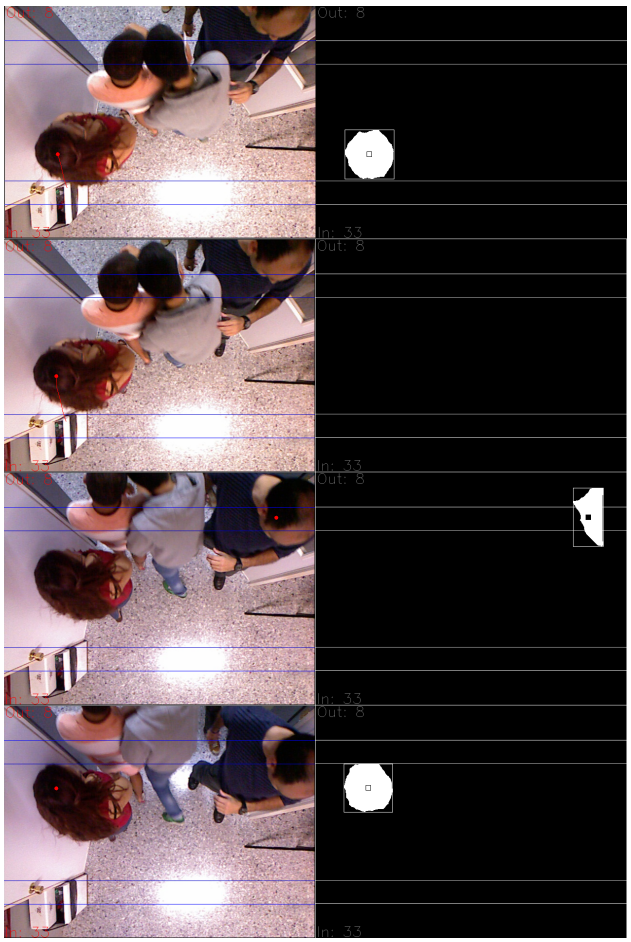

Figura 4.12: Cabeza perdida. La persona a la izquierda se une demasiado al resto y su trayectoria no se registra durante dos imágenes de la secuencia.

del aula esta en la parte superior, y es completamente posible que una persona se aproxime desde uno de los laterales. Cuando una persona se desplaza desde la parte superior de la escena hasta uno de los laterales, consideramos que entra en el aula (IN), mientras que si se desplaza de los laterales hacia la parte superior saldrá de la misma (OUT).

El resto de trayectorias que puedan ocurrir las consideramos trayectorias errantes o *wanderers*, y son omitidas de la cuenta. Pongamos un ejemplo para dar una explicación a esta decisión. Supongamos que una persona entra a la escena por la parte inferior, se desplaza hasta el centro de la escena, da la vuelta y sale por donde entró. Si contamos la persona cuando entra a la escena como IN y cuando sale como OUT, estamos falsificando el conteo, ya que realmente no ha entrado ni salido ninguna persona nueva al aula; simplemente, una persona que estaba dentro se aproximó al área de salida, pero no llegó realmente a salir; y si hubiera sido una persona saliendo se hubiera cuantificado como OUT, no como IN y luego como OUT como hemos ilustrado en este ejemplo. Debido a esto, almacenamos junto a cada persona en qué direccción entra (por la parte superior, inferior o uno de los laterales) y en qué dirección sale de la escena, y es en el momento de la salida cuando contrastamos las direcciones de entrada y salida y cuantificamos a la persona como IN u OUT.

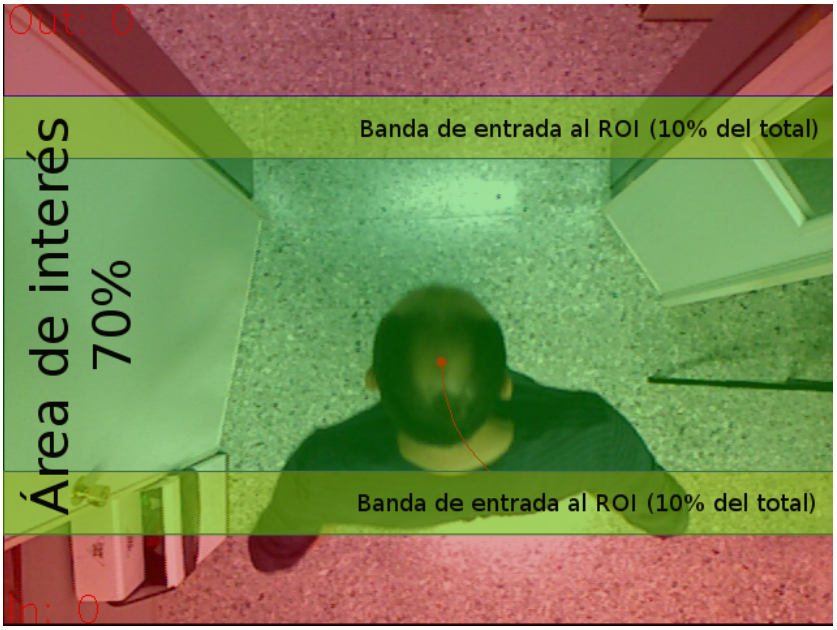

Figura 4.13: Área de interés escogida para la escena.

Capítulo 5

Pruebas y resultados obtenidos

Una vez construido nuestro sistema pasamos a describir las pruebas realizadas sobre el mismo, y los resultados que dichas pruebas arrojan. En la sección 5.1 contrastamos los resultados obtenidos con nuestro sistema en los conjuntos de imágenes que hemos utilizado con los valores reales de dichos conjuntos, mientras que en la sección 5.3 mostramos las tasas de acierto para los diferentes clasificadores entrenados con los datos extraídos de las sesiones, como explicamos en el apartado 5.2.

5.1. Conteo de personas durante diferentes sesiones

El primer resultado de este proyecto es la aplicación para conteo de personas *CounterPCL*. Este programa toma pares de imágenes (color y profundidad) ubicadas dentro de una carpeta o directamente de un sensor Kinect, y los analiza para detectar y contabilizar personas a su paso. También es capaz de extraer descriptores para cada persona, que se pueden utilizar para detectar el sexo de los transeúntes.

Este contador ha sido diseñado y probado con una serie de sesiones de imágenes, cuya captura fue descrita en el apartado 4.1. En total, nos encontramos con seis sesiones diferentes de imágenes, donde en cada una de ellas tenemos un número de personas que entra a la sala, que sale, o que describe una trayectoria errante o *wander*, donde entra y sale por el mismo sitio; estas

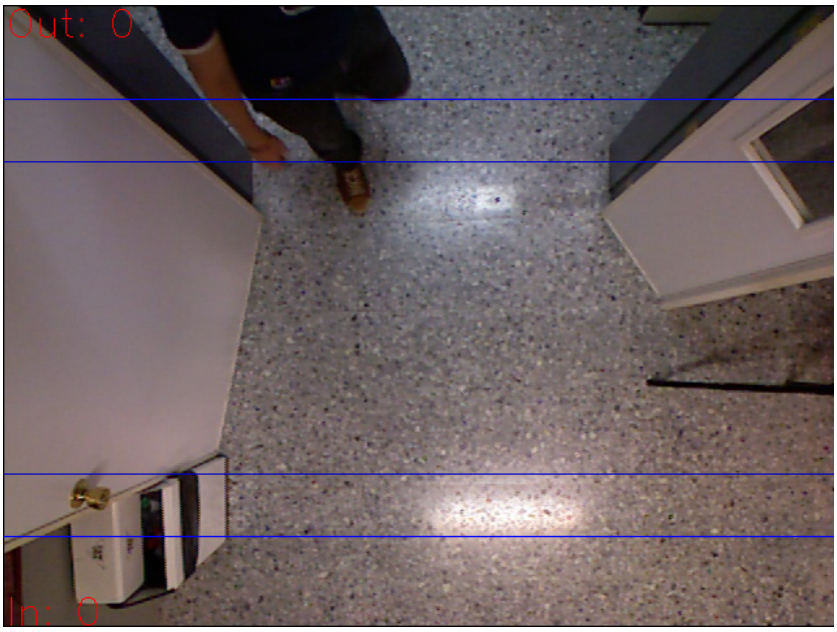

Figura 5.1: Interfaz de la aplicación realizada.

trayectorias no serán contabilizas por nuestro programa, como se ha explicado anteriormente. Las estadísticas de cada una de las sesiones (número de personas que entran o salen, número de fotogramas por segundo al capturar las imágenes) se encuentran en la tabla 5.1, mientras que los resultados del conteo de personas para las diferentes sesiones puede consultarse en la tabla 5.2.

Sesión	In	Out	W	Frames	Inicio	Fin	Seg.	FPS
Día 1, inicio	28	32		2184	07:58:51	08:02:09	198	11,03
Día 1, fin	30	10		3146	10:01:34	10:05:11	217	14,50
Día 2, inicio	37	39	1	2456	08:00:17	08:03:18	181	13,57
Día 2, fin	34	8	2	5940	10:01:15	10:05:40	265	22,42
Día 3, inicio	23	31	7	8315	07:57:21	08:02:51	330	25,20
Día 3, fin	18	10		812	10:01:41	10:04:21	160	5,08

Tabla 5.1: Resumen de las sesiones; datos extraídos de las anotaciones tomadas por los tutores en el momento de su captura. W = Wandering.

Hemos incluido dos versiones de nuestra aplicación, para ilustrar el progreso que se ha ido alcanzando. La primera de ellas corresponde al desarrollo inicial, donde tomamos tan sólo una de las sesiones para la implementación

Sesión	*Versión inicial*				*Versión final*			
	In	TF	Out	TF	In	TF	Out	TF
Día 1, inicio	32	14,29 %	33	3,13 %	27	3,57 %	33	3,13 %
Día 1, fin	44	46,67 %	15	50,00 %	27	10,00 %	10	0,00 %
Día 2, inicio	41	10,81 %	44	12,82 %	36	2,70 %	36	7,69 %
Día 2, fin	62	82,35 %	14	75,00 %	31	8,82 %	9	12,50 %
Día 3, inicio	41	78,26 %	44	41,94 %	26	13,04 %	33	6,45 %
Día 3, fin	23	27,78 %	12	20,00 %	8	55,56 %	9	10,00 %
Media TF		43,36 %		33,81 %		15,62 %		6,63 %

Tabla 5.2: Comparativa de personas contadas. TF = Tasa de Fallo.

del sistema y las pruebas de conteo iniciales. La segunda versión tomó todas las sesiones en consideración; durante el desarrollo de la misma se corrigieron diversos errores y se ajustaron parámetros dentro del código para que el contador tuviera un mejor funcionamiento.

Las tasas de fallo de conteo para la primera versión de la aplicación son bastante elevadas, pasando del 50 % en muchos casos, lo que hacía que una optimización fuera completamente necesaria. Una vez tomadas en consideración todas las sesiones y habiendo modificado la aplicación, conseguimos reducir considerablemente las tasas de fallo, alcanzando una media de 15 % como tasa de fallo el conteo de entrada y un 6 % para el de salida. Sin embargo, en la tabla podemos comprobar que una de las sesiones tiene una tasa de fallo de más del 50 %; necesitamos conocer lo que ocurre durante las diferentes sesiones para comprender esta gran cantidad de fallos.

- *Día 1, inicio.* Esta secuencia de imágenes presenta trayectorias simples y con apenas incidencias, ya que la mayoría de ellas se realizan individualmente, y una tasa de imágenes por segundo aceptable.

- *Día 1, fin.* En algunas de las trayectorias se aprecian algunos saltos y, por lo general, las trayectorias de los transeúntes se mantienen durante pocos fotogramas, indicando una velocidad alta por parte de los mismos o un ratio de imágenes por segundo más bajo en comparación con otras sesiones. En la información de profundidad, se dan casos de oclusiones entre dos personas. Cerca del final de esta sesión se cierra una puerta pequeña que hay de entrada al aula, detectando varias partes del cuerpo de la persona que la cierra.

- *Día 2, inicio.* Aquí tenemos trayectorias de transeúntes rápidas, pero no lo suficiente como para que no ocurran solapes dentro de la trayectoria. Tenemos algunas oclusiones, incluso existe un caso donde dos personas pasan por la escena con sus cabezas unidas, pero en general se trata de una sesión donde el conteo se puede realizar sin demasiados incidentes.

- *Día 2, fin.* En esta sesión encontramos que la puerta del aula se abre sin haber estado incluida en el modelo de fondo. Esto hace que se cuente en varias ocasiones como una persona que entra al aula, sobre todo mientras se abre, aunque después de un tiempo detectándola en el ROI acabemos registrándola en nuestra lista pero no contándola. Aparte de esto, en esta sesión las trayectorias de los transeúntes son lentas y fluidas.

- *Día 3, inicio.* Uno de los transeúntes en concreto transcurre por el ROI en varias ocasiones describiendo trayectorias de *wandering* y no es contabilizado. Además, en una de las trayectorias de esta naturaleza ocurre un salto, que hace que se le cuente dos veces.

- *Día 3, fin.* En esta sesión es donde más problemas tenemos para realizar el conteo. El número de fotogramas por segundo de esta sesión es demasiado bajo, y apenas ocurren solapes entre las diferentes posiciones de la trayectoria de la misma persona. Esto hace que nuestro método de conteo se haga prácticamente imposible, ya que tampoco tenemos una serie de posiciones para calcular la trayectoria y la velocidad de la persona que estamos intentando contar; de ahí la tasa de fallo tan elevada para esta sesión.

Si no tenemos en cuenta el conteo de la última sesión por los problemas que comentamos anteriormente, las tasas de fallo media para la versión final de nuestro contador quedarían en 7,63 % para las personas que entran y 5,95 % para las personas que salen, resultados bastante satisfactorios. En general podemos decir que nuestro contador tiene una tasa de fallo del 6,79 %.

5.2. Pruebas iniciales de clasificación del sexo

Durante la detección de personas se extrajeron una serie de valores que incluyen la altura de la persona, la medida de las normales de la cabeza, la circularidad de la misma, el área de la cabeza y si nuestro contador ha

decidido si se trata de una persona o no. Estos valores se almacenaron en formato `ARFF` (*Attribute-Relation File Format*, Formato de Fichero de Atributos Relacionados) para utilizarlos en la clasificación del sexo de las personas detectadas.

Un fichero en formato `ARFF` es un fichero de texto que describe una serie de instancias que comparten un conjunto de atributos. Este formato fue desarrollado en la Universidad de Waikato para su uso con el software Weka. Un fichero en este formato se divide en dos secciones: el encabezado y los datos. El encabezado del fichero `ARFF` contiene el nombre del conjunto de datos, una lista de atributos y sus tipos. Los datos dentro del fichero comienzan a partir de la declaración `@DATA`; en cada línea encontramos una instancia de los datos, con cada uno de los atributos separados por coma. En nuestro caso, hemos imprimido directamente los datos utilizados para la detección del sexo siguiendo este formato, añadiendo *a posteriori* simplemente el encabezado del fichero y almacenándolo como `.arff`.

Una vez obtenido el fichero con las componentes de cada persona en cada imagen de la secuencia en formato `ARFF`, ha sido contrastado con las anotaciones tomadas durante la grabación de las sesiones y se ha etiquetado cada tupla correspondiente a una persona en una imagen con un sexo, *MALE* o *FEMALE*. Además, se han creado dos archivos con el conjunto de datos para cada sesión; uno de ellos contiene la decisión de nuestro contador acerca de si el *blob* contabilizado ha sido detectado como una persona, mientras que el otro no contiene esta información. Al no estar seguros inicialmente de si este factor puede ayudar a discernir hombres de mujeres, realizamos pruebas aparte con él. Tras esto podemos pasar a trabajar con Weka.

Al abrir Weka, nos encontramos con cuatro opciones en la parte derecha de la interfaz:

- *Explorer*, la cual nos permite analizar los datos utilizando diferentes clasificadores, visualizarlos de diferentes formas o agruparlos en *clusters*, entre otras opciones. Hemos utilizado esta herramienta para entrenar diversos tipos clasificadores con nuestros datos y obtener alguno que nos clasifique el sexo de la mejor manera posible.
- *Experimenter*, donde podemos configurar una serie de experimentos para calcular clasificadores y ejecutarlos en modo *batch*, y recoger los resultados en un fichero en formato `.arff`. Después de las pruebas iniciales, hemos ejecutado el *Experimenter* con los diferentes datasets para cada sesión y un conjunto de clasificadores para su entrenamiento y posterior análisis.

- *KnowledgeFlow*, que consiste en una interfaz alternativa al *Explorer*, donde los componentes (fuentes de datos, filtros, clasificadores...) pueden colocarse en un lienzo y ser interconectados entre sí de manera visual.
- *Simple CLI*, que nos ofrece una interfaz por comandos para trabajar con Weka.

De las anteriores opciones hemos usado el *Explorer* y el *Experimenter*. Con el *Explorer* hemos realizado una serie de pruebas iniciales sobre los conjuntos de datos para comprobar cómo se comportan los diferentes clasificadores con ellos; posteriormente, hemos ejecutado conjuntos de pruebas con el *Experimenter* para calcular clasificadores utilizando nuestros conjuntos de datos, y hacer un análisis de qué clasificador es el mejor para nuestros datos. En este apartado se explicará el proceso realizado con el *Explorer*, mientras que el uso que se le ha dado al *Experimenter* será detallado en el apartado 5.2.

Al comenzar con la parte de detección del sexo de las personas contadas, se propusieron una serie de clasificadores que han dado buenos resultados con problemas similares. Para cada uno de ellos se realizaron diferentes pruebas y se optimizaron sus parámetros hasta obtener la mayor tasa de acierto posible con el clasificador entrenado. Estas pruebas se realizaron con los datos extraídos de la sesión del segundo día al inicio de la clase, tanto en su vertiente con la información de reconocimiento de persona (*With Detection*) como sin ella (*Without Detection*). Se utiliza un método de entrenamiento de validación cruzada de diez *folds* (en este método se divide el número de instancias en diez subconjuntos y se utiliza uno de ellos para prueba y el resto para entrenamiento, repitiendo el proceso durante diez iteraciones; en cada iteración se toma un subconjunto diferente para prueba, y al final del proceso se calcula la media de los diez clasificadores calculados) para el entrenamiento y prueba de los clasificadores, y se compararán los resultados entre los clasificadores para ver cuál de ellos se comporta mejor.

Además, hemos utilizado tres combinaciones de clasificadores como optimizadores: `CVParameterSelection` (Kohavi, 1995), `AdaBoost` (Freund and E. Schapire, 1996) y `Bagging` (Breiman, 1996), los cuales hemos ejecutado de manera independiente para cada sesión con la intención de maximizar todos los parámetros posibles dentro de cada uno de los clasificadores estudiados. Sólo `CVParameterSelection` permite una selección de los rangos de variación de los parámetros directa, y hemos aprovechado el resultado que aporta dicha combinación de clasificadores como punto de inicio para la optimización con los otros dos métodos. Hemos podido ejecutar estas combinaciones

de clasificadores para los tres primeros clasificadores estudiados inicialmente, ya que el resto no posee las características necesarias para poder utilizarlos, las cuales pasan por tener un parámetro numérico que optimizar. En total hemos hecho las pruebas iniciales con cinco clasificadores, los cuales veremos a continuación:

- `J48` (Quinlan, 1993). Este clasificador del tipo árbol es una implementación del algoritmo `C4.5`, referenciado en la bibliografía. Puede optimizarse mediante el parámetro C, llamado también factor de confianza, que influye sobre la decisión de poda del árbol. Los resultados de la optimización y la validación cruzada los podemos encontrar en la tabla 5.3.

	With Detection	*Without Detection*
Default, C=0,25	85,74 %	86,27 %
CVParameterSelection, C=0,1	84,85 %	86,27 %
AdaBoost, C=0,25	85,74 %	86,63 %
AdaBoost, C=0,1	85,03 %	85,38 %
Bagging, C=0,25	85,92 %	86,10 %
Bagging, C=0,1	85,21 %	85,56 %

Tabla 5.3: Tasas de acierto para J48 / C4.5.

  Podemos ver que presenta unas tasas de acierto bastante elevadas, de alrededor del 85 % en ambas versiones del conjunto de datos, y que aunque la información de detección de cabeza positiva genera peores resultados, la ganancia obtenida es muy poco significativa con respecto al conjunto de datos sin dicha información.

- `SVM` (*Support Vectors Machine*, Máquina de Vectores Soporte) (Cortes and Vapnik, 1995). Intuitivamente, se puede pensar en una SVM como un modelo que representa a los puntos de muestra en el espacio, separando las diferentes clases lo más posible. Cuando se añaden nuevas muestras al espacio, se clasifica como una u otra clase dependiendo de la proximidad a ellas. En concreto, hemos usado `LibSVM` (EL-Manzalawy, 2005; Chang and Lin, 2011), un *wrapper* para Weka de una implementación de este algoritmo. Pertenece a la clase de funciones, y para su optimización tenemos el parámetro gamma G. Vemos los resultados en la tabla 5.4.

	With Detection	*Without Detection*
Default, G=0,01	61,50 %	54,55 %
CVParameterSelection, G=0,001	72,01 %	78,43 %
AdaBoost, G=0	53,83 %	53,12 %
Adaboost, G=0,001	72,91 %	74,69 %
Bagging, G=0	54,37 %	54,37 %
Bagging, G=0,001	71,30 %	77,54 %

Tabla 5.4: Tasas de acierto para SVM.

Se aprecia mucho más el efecto de la optimización con este clasificador, aumentando en un 20 % en algunos casos. Sin embargo, este clasificador aporta resultados peores que el resto de clasificadores estudiados, incluso teniendo en cuenta las optimizaciones.

- `RandomForest` (Breiman, 2001). Otro miembro de la familia de los árboles, el cual genera varios árboles con elementos aleatorios para la clasificación. Podemos optimizar el número de árboles a generar, llamado I. Tenemos las tasas de acierto para este clasificador en la tabla 5.5.

	With Detection	*Without Detection*
Default, I=10	86,10 %	85,92 %
CVParameterSelection, I=50	86,45 %	86,99 %
AdaBoost, I=10	85,38 %	85,92 %
Adaboost, I=50	87,55 %	86,81 %
Bagging, I=10	86,81 %	85,74 %
Bagging, I=50	87,17 %	86,45 %

Tabla 5.5: Tasas de acierto para RandomForest.

Ofrece unos resultados similares a los obtenidos con `J48`, presentando por lo general una tasa de acierto ligeramente más elevada. Con el conjunto de datos que incluye la información de detección de cabeza tenemos un comportamiento mejor en la mayoría de los casos; sin embargo, ocurre lo mismo que con `J48`, ya que a pesar de haber un aumento en la tasa de acierto, no es demasiado significativo. Lo mismo ocurre con las optimizaciones; encontramos una mejora en la tasa de acierto, pero no demasiado significativa, ya que se encuentra alrededor del 1 % en la mayoría de casos.

- `BayesNet` (Bouckaert, 2008). Implementa una red bayesiana la cual podemos entrenar para que funcione como un clasificador para nuestro problema. No podemos aplicarle los métodos de optimización anteriormente mencionados ya que no tiene parámetros directos que podamos controlar; en su lugar, podemos modificar el estimador que utiliza y el algoritmo de búsqueda. En nuestro caso, hemos probado con variaciones sobre el algoritmo de búsqueda que utiliza con `simpleEstimator` como estimador, como vemos en la tabla 5.6.

Search Algorithm	*With Detection*	*Without Detection*
K2	82,17 %	82,17 %
GeneticSearch	81,82 %	80,93 %
HillClimber	82,00 %	82,17 %
LAGDHillClimber	75,94 %	75,97 %
RepeatedHillClimber	82,00 %	82,17 %
SimulatedAnnealing	81,82 %	81,64 %
TabuSearch	82,35 %	82,17 %
TAN	82,17 %	82,35 %

Tabla 5.6: Tasas de acierto para BayesNet.

Utilizando diferentes algoritmos de búsqueda dentro de nuestra red bayesiana obtenemos resultados bastante similares en todos los casos, exceptuando en `LAGDHillClimber`, el cual ofrece un rendimiento algo más bajo. El hecho de que con la gran mayoría de algoritmos de búsqueda obtengamos unos resultados tan similares parece indicar que el clasificador funciona de una manera bastante uniforme independientemente del algorimo de búsqueda seleccionado; posiblemente, las diferencias entre las tasas de acierto para los algoritmos de búsqueda exceptuando el nombrado anteriormente se deban a que las muestras para entrenamiento y clasificación son seleccionadas aleatoriamente para la validación cruzada.

- `NaiveBayes` (John and Langley, 1995). Otro clasificador bayesiano, el `NaiveBayes` está fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales. Para este clasificador no podemos utilizar tampoco los métodos de optimización usados en otros métodos; se ha probado a entrenar un clasificador de este tipo sin optimizadores, y posteriormente usando `AdaBoost` y `Bagging`, y los resultados, que podemos ver en la tabla 5.7, presentan un comportamiento similar al del

clasificador anterior, donde las variaciones entre los diferentes métodos son mínimas, indicando que probablemente lo que más influya en este caso sean las muestras escogidas para entrenamiento en la validación cruzada.

	With Detection	*Without Detection*
Default	82,35 %	81,82 %
AdaBoost	82,17 %	81,82 %
Bagging	81,46 %	81,64 %

Tabla 5.7: Tasas de acierto para NaiveBayes.

En general, tenemos una tasa de acierto superior al 80 % para la mayoría de las pruebas iniciales. El mejor que se comporta aquí es `RandomForest`, que habiendo sido optimizado con los métodos anteriormente descritos, alcanza un 87 % de aciertos en el conjunto de datos con información de detección de cabeza y un 86 % en el conjunto sin esa información. Sin embargo, hemos encontrado que las combinaciones de clasificadores usadas para la optimización no generan una mejora tan significativa, ya que como máximo tenemos un incremento que no alcanza el 2 % en los clasificadores con mejor comportamiento. Uno de ellos, `SVM`, sí que presenta un incremento mayor, pero aún así, queda muy por debajo de los otros clasificadores.

Estas pruebas iniciales nos han servido para desarrollar un conjunto de clasificadores a entrenar con Weka, teniendo en cuenta también las optimizaciones, cuyos resultados serán mostrados a continuación.

5.3. Determinación del sexo de las personas contadas

Como se comentó en la sección 5.2, para la clasificación del sexo se ha preparado un *batch* con todos los clasificadores posibles, optimizando en los casos donde nos sea posible y, para combinaciones de clasificadores o clasificadores que usen a otros, utilizaremos alguno de los analizados en detalle anteriormente. Para cada conjunto de datos mostraremos los diez mejores clasificadores en las dos variantes del conjunto (con información de detección

de cabeza proporcionada por el programa y sin ella, llamados *With Detection* o *Without Detection* manteniendo la nomenclatura utilizada anteriormente) y realizaremos una comparativa entre ellos.

- *Día 1, inicio*

  Ambas variantes del conjunto de datos, que encontramos en las tablas 5.8 y 5.9, aportan unos resultados similares con las tasas de acierto más altas alrededor del 92 %, si bien la versión *With Detection* consigue una tasa de acierto ligeramente superior. En cuanto a los clasificadores que funcionan mejor en este conjunto de datos, la mejor tasa de acierto es para `LogitBoost` en ambos casos, con `Ibk` y `RandomCommittee` ocupando los siguientes puestos. Nótese que tanto `LogitBoost` como `RandomCommittee`, que son ambos combinaciones de clasificadores, utilizan `RandomForest` como su clasificador base, hecho que veremos bastante con el resto de conjuntos de datos. De hecho, para la versión *Without Detection* tenemos que cuatro de los diez mejores resultados utilizan este clasificador como base, mientras que en la versión *With Detection* nos encontramos con siete resultados que lo utilizan, incluyendo `RandomForest` sin ninguna combinación de clasificadores con la cuarta mejor tasa de acierto.

Clasificador	*Tasa de acierto*
LogitBoost (con RandomForest)	92,46 %
RandomCommittee (con RandomForest)	91,69 %
Ibk	91,66 %
RandomCommittee (con RandomTree)	91,17 %
AttributeSelectedClassifier (con RandomForest)	90,94 %
Bagging (con J48)	90,92 %
AdaBoost (con RandomForest)	90,92 %
RandomizableFilteredClassifier (con RandomProjection)	90,91 %
ClassificationViaRegression (con M5P)	90,70 %
AdaBoost (con J48)	90,69 %

Tabla 5.8: Resultados de la detección de sexo para el primer conjunto de datos (Without Detection).

- *Día 1, fin*

  De nuevo, nos encontramos con una tasa de acierto que ronda el 92 % para la variante del conjunto de datos *Without Detection* (tabla 5.10),

Clasificador	*Tasa de acierto*
LogitBoost (con RandomForest)	92,96 %
Ibk	92,17 %
RandomCommittee (con RandomForest)	91,94 %
RandomForest	91,69 %
ClassificationViaRegression (con RandomForest)	91,17 %
Kstar	91,17 %
AdaBoost (con RandomForest)	90,95 %
AdaBoost (con J48)	90,94 %
AttributeSelectedClassifier (con RandomForest)	90,94 %
Bagging (con RandomForest)	90,93 %

Tabla 5.9: Resultados de la detección de sexo para el primer conjunto de datos (With Detection).

mientras que en la variante *With Detection* (tabla 5.11) tenemos una tasa de acierto que supera el 93 %. Los resultados obtenidos con la variante *With Detection* son ligeramente superiores en este conjunto de datos. Encabezando las listas tenemos los clasificadores `LogitBoost` para *Without Detection* y `AdaBoost` para *With Detection*, ambos utilizando `RandomForest`; en las posiciones sucesivas ambas variantes tienen a `RandomCommittee` usando `RandomForest` y a `Ibk` en ese orden. Podemos comprobar que el clasificador `RandomForest` continúa estando muy presente entre los mejores clasificadores, con cuatro apariciones en *Without Detection* y seis en *With Detection*.

- *Día 2, inicio*

  Para este conjunto de datos tenemos unas tasas de acierto inferiores a las de los conjuntos vistos anteriormente; para la versión *Without Detection* (tabla 5.12) la mejor tasa de acierto es de alrededor del 86 %, mientras que para la versión *With Detection* (tabla 5.13) esta tasa de acierto se sitúa sobre el 87 %. De nuevo volvemos a obtener valores ligeramente superiores en la versión del conjunto de datos *With Detection*; sin embargo, tenemos que las tasas de acierto para los últimos clasificadores de la tabla son algo mejores para *Without Detection*. Con las mejores tasas de acierto tenemos a `AdaBoost` usando `J48` en la versión *Without Detection*, seguido de `RandomCommittee` usando `RandomForest` y de `J48`, mientras que para la versión *With Detection* encontramos a `MultilayerPerceptron` con la mejor tasa de acierto, y

Clasificador	*Tasa de acierto*
LogitBoost (con RandomForest)	92,66 %
RandomCommittee (con RandomForest)	92,17 %
Ibk	91,90 %
RandomCommittee (con RandomTree)	90,45 %
AttributeSelectedClassifier (con RandomForest)	84,32 %
Bagging (con J48)	91,19 %
AdaBoost (con RandomForest)	90,95 %
RandomizableFilteredClassifier (con RandomProjection)	87,73 %
ClassificationViaRegression (con M5P)	87,77 %
AdaBoost (con J48)	89,72 %

Tabla 5.10: Resultados de la detección de sexo para el segundo conjunto de datos (Without Detection).

Clasificador	*Tasa de acierto*
AdaBoost (con RandomForest)	93,15 %
RandomCommittee (con RandomForest)	92,66 %
Ibk	92,39 %
LogitBoost (con RandomForest)	92,18 %
Bagging (con RandomForest)	92,17 %
MultilayerPerceptron	91,42 %
ClassificationViaRegression (con RandomForest)	91,18 %
RandomForest	90,71 %
AdaBoost (con J48)	90,46 %
RandomCommittee (con RandomTree)	90,20 %

Tabla 5.11: Resultados de la detección de sexo para el segundo conjunto de datos (With Detection).

por detrás tenemos a `LMT` y `RandomCommittee` usando `RandomForest`. Este último clasificador lo hemos encontrado siempre entre las tres mejores tasas de acierto de los conjuntos de datos vistos hasta ahora. Vemos que `RandomForest` sigue participando de forma numerosa en las mejores tasas de acierto, con cinco apariciones en la tabla de *Without Detection* y cuatro en *With Detection*.

Clasificador	*Tasa de acierto*
AdaBoost (con J48)	86,63 %
RandomCommittee (con RandomForest)	86,63 %
J48	86,28 %
ClassificationViaRegression (con RandomForest)	86,27 %
MultilayerPerceptron	86,09 %
LogitBoost (con RandomForest)	86,09 %
Bagging (con J48)	86,09 %
AdaBoost (con RandomForest)	85,92 %
RandomForest	85,91 %
Bagging (con REPTree)	85,91 %

Tabla 5.12: Resultados de la detección de sexo para el tercer conjunto de datos (Without Detection).

- *Día 2, fin*

  Aquí encontramos que, aunque hayamos conseguido una tasa de acierto máxima ínfimamente superior con la versión *Without Detection* (tabla 5.14), las tasas de acierto obtenidas para la versión *With Detection* (tabla 5.15) son por lo general más altas. Estas tasas de acierto rondan el 89 % para ambas versiones del conjunto de datos. Los clasificadores que mejor se comportan son `LogitBoost` usando `RandomForest`, `RandomCommittee` usando `RandomForest` y `AdaBoost` usando `J48` para la versión del conjunto de datos *Without Detection*, mientras que `RandomCommittee` usando `RandomForest`, `AdaBoost` usando `RandomForest` y `LogitBoost` usando también `RandomForest` obtienen los mejores resultados para la versión *With Detection*. Volvemos a encontrarnos con la combinación de `RandomCommittee` y `RandomForest` entre las mejores tasas de acierto para ambos conjuntos de datos y de nuevo `RandomForest` tiene numerosas apariciones en nuestras tablas, con cinco en la versión *Without Detection* y seis en la versión *With Detection*.

- *Día 3, inicio*

Clasificador	*Tasa de acierto*
MultilayerPerceptron	87,35 %
LMT	87,34 %
RandomCommittee (con RandomForest)	87,16 %
Bagging (con RandomForest)	87,16 %
LogitBoost (con RandomForest)	86,99 %
RandomCommittee (con RandomTree)	86,28 %
RandomForest	86,10 %
Bagging (con J48)	85,91 %
AdaBoost (con J48)	85,74 %
ClassificationViaRegression (con M5P)	85,74 %

Tabla 5.13: Resultados de la detección de sexo para el tercer conjunto de datos (With Detection).

Clasificador	*Tasa de acierto*
LogitBoost (con RandomForest)	88,98 %
RandomCommittee (con RandomForest)	88,75 %
AdaBoost (con J48)	88,09 %
Bagging (con RandomForest)	88,07 %
RandomCommittee (con RandomTree)	88,06 %
AdaBoost (con RandomForest)	87,86 %
Bagging (con J48)	87,84 %
ClassificationViaRegression (con RandomForest)	87,39 %
Kstar	87,18 %
Bagging (con REPTree)	85,81 %

Tabla 5.14: Resultados de la detección de sexo para el cuarto conjunto de datos (Without Detection).

Clasificador	*Tasa de acierto*
RandomCommittee (con RandomForest)	88,97 %
AdaBoost (con RandomForest)	88,96 %
LogitBoost (con RandomForest)	88,75 %
ClassificationViaRegression (con RandomForest)	88,52 %
Kstar	88,07 %
Bagging (con RandomForest)	87,62 %
Bagging (con J48)	87,61 %
AdaBoost (con J48)	87,60 %
RandomForest	87,41 %
LMT	86,72 %

Tabla 5.15: Resultados de la detección de sexo para el cuarto conjunto de datos (With Detection).

Para este conjunto de datos, obtenemos valores para las tasas de acierto muy similares en ambas versiones, alrededor de un 95 % como el máximo de cada una de ellas. Estos valores se encuentran en las tablas 5.16 y 5.17. Con los mejores resultados para *Without Detection* encontramos a `Ibk`, `LogitBoost` usando `RandomForest` y `AdaBoost` usando también `RandomForest`, mientras que para *With Detection* tenemos también a `Ibk` con los mejores resultados seguido de `RandomCommittee` usando `RandomForest` y `AdaBoost` usando `RandomForest`. Mientras que `RandomForest` sigue manteniendo unas tasas de acierto bastante elevadas, en este caso no encontramos la combinación de `RandomCommittee` y `RandomForest` entre los tres mejores en la versión *Without Detection*; se encuentra como el quinto mejor. Sin embargo, eso no quiere decir que no sea un clasificador viable para la detección de sexo en este caso, ya que su tasa de acierto no alcanza a bajar un 1 % con respecto a la mejor en este conjunto de datos.

- *Día 3, fin*

  Obtenemos tasas de acierto muy similares, con las mejores situadas en casi un 88 % para ambas versiones del conjunto de datos, como podemos ver en las tablas 5.18 y 5.19. Por lo general, las tasas de acierto de la versión *Without Detection* presentan valores ligeramente más altos, aunque son casi coincidentes con los de la otra versión del conjunto de datos. Como mejores clasificadores tenemos, para la versión *Without Detection* a `Bagging` con `RandomForest`, `RandomTree` y

Clasificador	*Tasa de acierto*
Ibk	95,24 %
LogitBoost (con RandomForest)	94,97 %
AdaBoost (con RandomForest)	94,91 %
AdaBoost (con J48)	94,75 %
RandomCommittee (con RandomForest)	94,69 %
ClassificationViaRegression (con RandomForest)	94,69 %
Kstar	94,69 %
Bagging (con RandomForest)	94,64 %
RandomForest	94,37 %
RandomCommittee (con RandomTree)	94,36 %

Tabla 5.16: Resultados de la detección de sexo para el quinto conjunto de datos (Without Detection).

Clasificador	*Tasa de acierto*
Ibk	95,40 %
RandomCommittee (con RandomForest)	95,19 %
AdaBoost (con RandomForest)	95,13 %
Bagging (con RandomForest)	94,80 %
RandomForest	94,80 %
LogitBoost (con RandomForest)	94,69 %
LMT	94,64 %
AdaBoost (con J48)	94,64 %
ClassificationViaRegression (con RandomForest)	94,42 %
Kstar	94,20 %

Tabla 5.17: Resultados de la detección de sexo para el quinto conjunto de datos (With Detection).

`LogitBoost` usando `RandomForest`, mientras que para la versión *With Detection* encontramos a `MultilayerPerceptron`, `LogitBoost` usando `RandomForest` y `RandomForest` como los mejores clasificadores. De nuevo, tenemos muchas instancias de `RandomForest` entre las mejores tasas de acierto, localizando cinco en la versión *Without Detection* y seis en *With Detection*. Podemos ver la combinación de `RandomCommittee` con `RandomForest` entre las mejores tasas de acierto para ambas versiones, pero no están presentes entre las tres mejores como ha ocurrido con otros conjuntos de datos.

Clasificador	*Tasa de acierto*
Bagging (con RandomForest)	87,94 %
RandomTree	87,43 %
LogitBoost (con RandomForest)	87,35 %
RandomCommittee (con RandomForest)	87,35 %
AdaBoost (con RandomForest)	87,35 %
RandomCommittee (con RandomTree)	86,76 %
SMO	86,14 %
ClassificationViaRegression (con RandomForest)	86,14 %
MultilayerPerceptron	86,10 %
AttributeSelectedClassifier (con J48)	86,10 %

Tabla 5.18: Resultados de la detección de sexo para el sexto conjunto de datos (Without Detection).

- *Conjunto de datos completo*

  Finalmente, tenemos el conjunto de datos completo, que comprende todas las instancias contenidas en las seis sesiones anteriores y las trata como un solo conjunto de datos. Obtenemos una tasa de acierto ligeramente superior en la versión del conjunto *Without Detection* (tabla 5.20), con un 89,5 % frente al 89,4 % de *With Detection* (tabla 5.21); además, el resto de tasas de acierto se mantienen superiores en la primera versión del conjunto. Como mejores clasificadores encontramos, para *Without Detection*, `AdaBoost`, `LogitBoost` y `RandomCommittee` utilizando todos `RandomForest`; mientras que para la versión *With Detection* tenemos `Adaboost`, `RandomCommittee` y `Bagging` utilizando también todos `RandomForest`. No es de extrañar el buen funcionamiento de este clasificador, ya que ha conseguido de las mejores tasas de acierto con diferentes grupos de clasificadores para todos los conjuntos de datos anteriores.

Clasificador	*Tasa de acierto*
MultilayerPerceptron	87,94 %
LogitBoost (con RandomForest)	87,94 %
RandomForest	86,80 %
RandomSubSpace (con RandomForest)	86,73 %
AdaBoost (con RandomForest)	86,73 %
ClassificationViaRegression (con RandomForest)	86,69 %
RandomCommittee (con RandomForest)	86,14 %
CVParameterSelection (con JRip)	86,10 %
MultiScheme (con ZeroR, JRip)	86,10 %
AttributeSelectedClassifier (con J48)	86,10 %

Tabla 5.19: Resultados de la detección de sexo para el sexto conjunto de datos (With Detection).

Clasificador	*Tasa de acierto*
AdaBoost (con RandomForest)	89,54 %
LogitBoost (con RandomForest)	89,41 %
RandomCommittee (con RandomForest)	89,17 %
ClassificationViaRegression (con RandomForest)	89,04 %
Bagging (con RandomForest)	88,91 %
AdaBoost (con J48)	88,46 %
RandomCommittee (con RandomTree)	88,46 %
Bagging (con REPTree)	88,36 %
Kstar	88,33 %
Ibk	88,09 %

Tabla 5.20: Resultados de la detección de sexo para el conjunto de datos completo (Without Detection).

Clasificador	*Tasa de acierto*
AdaBoost (con RandomForest)	89,41 %
RandomCommittee (con RandomForest)	89,38 %
Bagging (con RandomForest)	89,12 %
LogitBoost (con RandomForest)	89,12 %
ClassificationViaRegression (con RandomForest)	88,51 %
RandomForest	88,44 %
Bagging (con REPTree)	88,36 %
Bagging (con J48)	88,25 %
Ibk	88,20 %
AdaBoost (con J48)	88,17 %

Tabla 5.21: Resultados de la detección de sexo para el conjunto de datos completo (With Detection).

- *Comparativa entre Without Detection y With Detection*

  Hemos recogido en una gráfica los valores máximos de tasas de acierto para los conjuntos de datos anteriores, para comparar de manera sencilla si afecta o no el tener un valor más a la hora de detectar el sexo de los transeúntes. Esta gráfica es la 5.2.

  Tenemos unos resultados bastante interesantes. En todos los conjuntos de datos que representan una sola sesión, al añadir la información de si es o no una cabeza según nuestro sistema, obtenemos unas tasas de acierto ligeramente mayores; sin embargo, al utilizar el conjunto completo como entrenamiento y prueba, obtenemos una tasa de fallo mayor no utilizando dicha información. Es posible que al tener un mayor número de muestras en el mismo conjunto de datos la importancia de uno de los campos del conjunto de datos disminuya, y esto cause este efecto tan interesante. Sin embargo, y aunque una tasa de acierto sea por lo general mayor que la otra, las diferencias son insignificantes, tanto que la mayor diferencia entre ambas no llega a un 1 %. Es fácil asumir que la inclusión de la información de si se ha detectado una cabeza no aporta demasiada utilidad, y se podría eliminar de los conjuntos de datos sin que ello afectara notablemente a la detección del sexo.

- *Utilizando diferentes conjuntos de datos para entrenamiento y prueba*

  Basándonos en los resultados arrojados por los clasificadores calculados con cada una de las sesiones individuales, decidimos entrenar uno de los

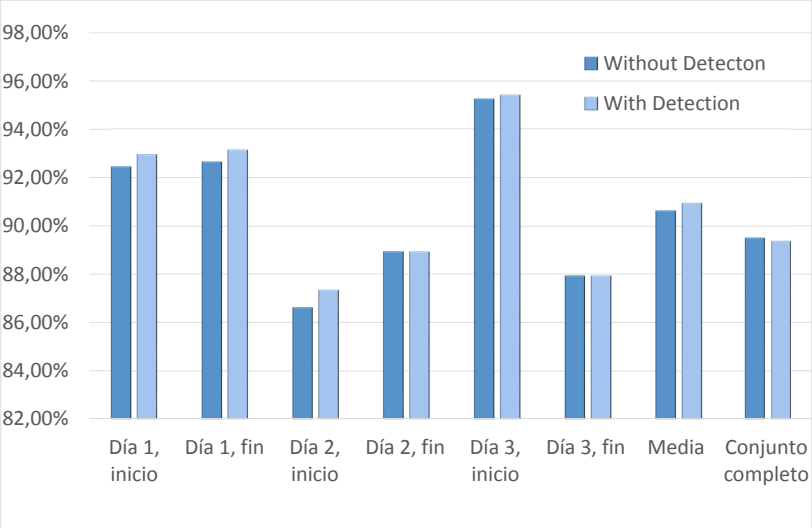

Figura 5.2: Comparativa entre Without Detection y With Detection.

clasificadores con mejores resultados (`RandomCommittee` como grupo de clasificadores con `RandomForest`) utilizando diferentes conjuntos de datos para entrenamiento y para prueba, en lugar de utilizar el método de validación cruzada que hemos usado hasta el momento, cuyos resultados se encuentran en la tabla 5.22. De esta forma podemos afianzar la robustez del conjunto de datos, y dar por válidas las características utilizadas para la detección del sexo de las personas contadas.

Los resultados, aunque con tasas de acierto por lo general algo menores que tomando la misma sesión para entrenamiento y validación, nos ofrecen una tasa de acierto máxima de 89,58 % y una mínima de 65,68 %, con una media de 82,43 %. Este descenso en la tasa de acierto puede estar causado por las diferencias en las personas capturadas en las diferentes sesiones; al trabajar dentro de una misma sesión, como hicimos anteriormente, y utilizar el método de validación cruzada, imágenes de una misma trayectoria podrán encontrarse tanto en el conjunto de entrenamiento como en el de prueba, resultando en un mayor número de detecciones de sexo correctas. En este caso, no podemos estar seguros de que las mismas personas estén en ambos conjuntos de datos, e incluso si lo estuvieran, es posible que ciertos elementos físicos en la persona que pueden cambiar de una sesión a la otra tales

Validación	*Entrenamiento*					
	Día 1, i	Día 1, f	Día 2, i	Día 2, f	Día 3, i	Día 3, f
Día 1, inicio	100,00 %	83,92 %	82,66 %	77,14 %	77,14 %	70,35 %
Día 1, fin	81,48 %	100,00 %	79,26 %	80,25 %	74,07 %	65,68 %
Día 2, inicio	82,71 %	77,36 %	100,00 %	82,71 %	74,15 %	79,32 %
Día 2, fin	75,23 %	66,82 %	72,73 %	100,00 %	68,18 %	81,59 %
Día 3, inicio	82,58 %	84,12 %	89,58 %	89,03 %	100,00 %	86,00 %
Día 3, fin	83,03 %	72,12 %	82,42 %	84,24 %	81,82 %	100,00 %

Tabla 5.22: Tasas de acierto al utilizar diferentes conjuntos de datos para entrenamiento y prueba.

como un peinado diferente influyan directamente en su caracterización en nuestro sistema y, por lo tanto, en la detección del sexo, debido a las características escogidas para realizar esta tarea. No obstante, el descenso no es tan pronunciado como cabría esperar, afirmando la validez de las características elegidas para la detección del sexo.

Una variante sobre este método de entrenamiento y detección consiste en calcular el sexo, en lugar de por instancias de una persona en una imagen, por trayectorias detectadas. Al realizar el conteo de personas estamos también almacenando su trayectoria, la cual podemos utilizar para determinar las imágenes correspondientes a una persona y detectar el sexo de la misma a lo largo de dicha trayectoria. Para ello obtendremos el sexo a partir de nuestro clasificador para cada imagen de la trayectoria, y concluiremos que su sexo es el sexo obtenido para la mayoría de imágenes de la trayectoria. Los resultados obtenidos para esta variante se muestran en la tabla 5.23.

La mayor tasa de acierto obtenida ha sido de 88,52 % mientras que la menor ha sido de 67,50 %, con una media de 83,03 %. Aunque la tasa de acierto máxima sea algo menor que en el caso anterior, podemos ver que presenta un comportamiento más estable, ya que entre las diferentes sesiones las tasas de acierto se mantienen ligeramente más altas que sin utilizar la información de las trayectorias. Podemos apreciar que incluir la información acerca de las trayectorias de la sesión mejora la detección, pudiendo considerarse una pequeña mejora en el sistema.

Validación	*Entrenamiento*					
	Día 1, i	Día 1, f	Día 2, i	Día 2, f	Día 3, i	Día 3, f
Día 1, inicio	100,00 %	88,52 %	78,69 %	80,33 %	81,97 %	75,41 %
Día 1, fin	82,50 %	100,00 %	77,50 %	85,00 %	67,50 %	70,00 %
Día 2, inicio	81,94 %	79,17 %	100,00 %	84,72 %	75,00 %	83,33 %
Día 2, fin	79,55 %	72,73 %	75,00 %	100,00 %	75,00 %	84,09 %
Día 3, inicio	81,36 %	83,05 %	88,14 %	84,75 %	100,00 %	88,14 %
Día 3, fin	71,43 %	71,43 %	78,57 %	82,14 %	82,14 %	100,00 %

Tabla 5.23: Tasas de acierto al utilizar diferentes conjuntos de datos para entrenamiento y prueba, promediando el sexo por cada trayectoria detectada.

Capítulo 6

Conclusiones y futuras líneas de trabajo

En este capítulo concluyente comentamos las diferentes conclusiones a las que hemos llegado después de realizar este proyecto, y posteriormente ofrecemos una serie de líneas de trabajo para la mejora del sistema que hemos construido.

6.1. Conclusiones

El objetivo principal del proyecto consiste en desarrollar un sistema contador de personas utilizando un sensor RGB-D para capturar imágenes de las mismas, y realizar un estudio sobre la viabilidad de este sistema y de la detección del sexo de las personas contabilizadas.

El sistema que hemos desarrollado permite realizar el conteo de personas que atraviesan el campo de visión del sensor. Debido a que también utilizamos información acerca de la profundidad de la escena, presenta ventajas sobre sistemas basados en sensores RGB, como una mayor tolerancia a cambios de iluminación en la escena. El punto de vista cenital que hemos escogido para el sistema evita también ciertos problemas de oclusión entre diferentes personas además de respetar la privacidad de las personas contadas.

Hemos obtenido resultados favorables con el conteo de personas, con una tasa de acierto del 93 % para las trayectorias contadas. Nuestro sistema implementa mecanismos para la diferenciación de personas en la escena, la separación de las mismas cuando encontremos varias agrupadas y el seguimiento

de ellas, e incluso intenta localizar de nuevo una persona si su detección se ve dificultada en medio de su trayectoria. Sin embargo, encontramos también problemas que afectan al reconocimiento y conteo de personas:

- El método para el modelado del fondo de la escena es estático. Un método adaptativo haría que el sistema fuera más robusto, aunque un algoritmo de este tipo será más costoso que el que hemos utilizado.
- En ocasiones no es posible separar completamente dos o más personas que vayan agrupadas. Si por algún motivo varias personas van demasiado unidas, es posible que se las detecte como una sola, falsificando el conteo.
- La detección de personas no es perfecta. Hemos utilizado una serie de medidas que permiten una buena detección, pero también hemos encontrado casos que inicialmente no esperábamos. Aunque ciertos problemas de este tipo se han logrado solventar, el sistema de detección es mejorable.
- Si el número de imágenes por segundo de la secuencia es bajo, tendremos fallos en el conteo por no poder realizar el seguimiento de cada persona de manera correcta; no encontraremos solapes entre las diferentes posiciones de la trayectoria a seguir.

También hemos realizado un estudio sobre la detección de sexo utilizando características de las personas detectadas extraídas con este sistema. Estos datos consisten en la altura de la persona, el tamaño de la cabeza de la misma, su circularidad y nuestra medida de las normales. Para ello hemos extraído estos datos de las secuencias de imágenes utilizadas, los hemos etiquetado con el sexo de la persona y hemos utilizado Weka para entrenar diferentes clasificadores. Haciendo uso de los diferentes conjuntos de datos obtenidos de cada una de las secuencias y empleándolos para entrenar y validar un clasificador de sexo, obtenemos una media de 83 % en la tasa de acierto, lo cual nos indica que, aunque el clasificador no es perfecto, permite la detección del sexo con un margen de error relativamente pequeño. Podemos decir que los objetivos principales del proyecto se han cumplido.

Desde un punto de vista académico, este proyecto me ha hecho adquirir diferentes tipos de experiencia. Anteriormente no había tratado con bibliotecas como OpenCV o PCL, conocimiento del que ahora dispongo. Tampoco tenía experiencia con el tratamiento de imágenes o la visión por computador;

este último campo existía como una asignatura, pero debido a la extinción del plan de estudios no pude cursarla. Gracias a este proyecto ahora cuento con el bagaje necesario para enfrentarme a desafíos en el futuro que tengan que ver con este campo. Me ha dado también mucha experiencia en el campo de la programación, y me ha ayudado a crecer como persona; ha sido una experiencia intensa y gratificante.

Además, aunque muchos proyectos queden archivados una vez son expuestos, este estará disponible a través de *BitBucket* en el siguiente repositorio `https://bitbucket.org/Edv/pfc` para todo aquel que lo necesite, con lo cual será útil no sólo dentro de la Universidad.

6.2. Futuras líneas

Este proyecto puede ampliarse y modificarse para dar lugar a un producto más completo y con un mejor funcionamiento. A continuación se exponen algunas de dichas mejoras.

- *Adaptar el sistema para su funcionamiento con Kinect 2.0.* Como se comentó en la sección 2.1, durante el desarollo del proyecto se lanzó una nueva versión del sensor Kinect, que ofrece unas prestaciones superiores que pueden ayudar a mejorar este sistema.

- *Cálculo dinámico del modelo de fondo.* Hemos utilizado un modelado de fondo estático, pero podría mejorarse para incluir objetos en él que se hayan introducido *a posteriori*. Podríamos considerar recalcular el modelo de fondo después de un determinado número de imágenes tratadas, o adoptar algún otro algoritmo de sustracción de fondo dinámico conocido.

- *Implementación de un algoritmo alternativo para la segmentación de personas en la imagen.* Una variante de un algoritmo como *Watershed* o *Mean-shift* podría utilizarse a la hora de detectar personas dentro de cada imagen, para intentar mejorar este proceso.

- *Ampliación de las funcionalidades del sistema.* Se podría añadir un método de re-identificación de personas utilizando los datos biométricos extraídos anteriormente, o buscando otras características. Cabe destacar el proyecto fin de carrera (Mireles-Suárez, 2015) *Re-identificación*

de personas en redes de sensores RGB-D, de Alberto Mireles Suárez, el cual utiliza redes de sensores Kinect para la reidentificación de personas. Dicho proyecto y el presente fueron desarrollados de manera paralela, de ahí que surja esta línea futura de trabajo.

Bibliografía

Alemán-Flores, M. (2011). Modelos de ciclo de vida del software.

Bouckaert, R. R. (2008). Bayesian network classifiers in weka for version 3-5-7. Available at `http://www.cs.waikato.ac.nz/~remco/weka.bn.pdf`.

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, Inc.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Castrillón-Santana, M., Lorenzo-Navarro, J., and Hernández-Sosa, D. (2014a). Conteo de personas con un sensor RGB-D comercial. *Revista Iberoamericana de Autmática e Informática industrial*, (11):348–357.

Castrillón-Santana, M., Lorenzo-Navarro, J., and Hernández-Sosa, D. (2014b). Depthvisdoor database.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, pages 273–297.

EL-Manzalawy, Y. (2005). WLSVM. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm/`.

Fernández-Pereira, I. (2013). Sistema automático de conteo de personas basado en sensor de profundidad. Master's thesis, Escola de Enxeñaría de Telecomunicación de la Universidade de Vigo.

Freund, Y. and E. Schapire, R. (1996). Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning, San Francisco*, pages 148–156.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, (11:1).

Horcajada-González, R. (2011). *Apuntes generales de anatomía morfológica aplicada: cánones y proporciones*. Departamento de Dibujo I, Facultad de Bellas Artes de San Fernando, Universidad Complutense de Madrid.

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, (8:2):179–187.

Itseez (2015). Página web oficial de OpenCV.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifier. In *Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo*, pages 338–345.

Kohavi, R. (1995). *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. Department of Computer Science, Stanford University.

Lorenzo-Navarro, J., Castrillón-Santana, M., and Hernández-Sosa, D. (2013). On the use of simple geometric descriptors provided by RGB-D sensors for re-identification. *Sensors*, (13):8222–8238.

Machine Learning Group at the University of Waikato (2015). Página web oficial de Weka.

Mathé, L., Sambán, D., and Gómez, G. (2012). Estudio del funcionamiento del sensor kinect y aplicaciones para bioingeniería. In *Argencon 2012*.

Microsoft Corporation (2015). Página web oficial de Xbox.

Mireles-Suárez, A. (2015). Re-identificación de personas utilizando redes de sensores RGB-D. Master's thesis, Escuela de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria.

Open Perception Foundation (2015). Página web oficial de PCL.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Sonka, M., Hlavac, V., and Boyle, R. (2008). *Image Processing, Analysis, and Machine Vision*. Thomson Learning.

Structure Sensor Team (2015). Página web oficial de OpenNI.

Zhang, X., Yan, J., Feng, S., Lei, Z., Yi, D., and Li, S. Z. (2012). Water filling: Unsupervised people counting via vertical kinect sensor. In *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*.

Glosario

ARFF Attribute-Relation File Format.

batch Conjunto de pruebas ejecutadas a la vez.

blob Región de una imagen.

BSD Berkeley Software Distribution.

frame Cada una de las imágenes de una secuencia.

GNU-GPL GNU General Public License.

PCL Point Cloud Library.

RGB Red, Green, Blue.

RGB-D Red, Green, Blue and Depth.

ROI Region Of Interest.

SDK Software Development Kit.

Weka Waikato Environment for Knowledge Analysis.