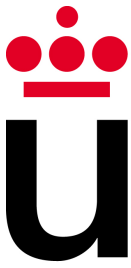


Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería
de Telecomunicación

Trabajo fin de grado

Contador de personas con sensor RGBD

Autor: Rebeca Sáez Díaz-Guerra
Tutor: Prof. José María Cañas

Doble Grado en Ingeniería en Sistemas de Telecomunicación y Administración y Dirección de Empresas

Curso académico 2015/2016

A mi padre.

Agradecimientos

En primer lugar agradecer a mi padre todo lo que ha luchado para que pueda estar donde estoy ahora, porque sin él no habría podido empezar y no habría sabido terminar. Gracias por entenderme y estar siempre ahí. También darle las gracias a mi madre, por su cariño y especial comprensión no sólo estos últimos años, sino desde que me alcanza la memoria. Gracias por tu ánimo y apoyo.

Como no acordarme de mi hermana Elena, por toda la energía que me ha transmitido y por su capacidad de hacerme olvidar todos los problemas. Por las risas y por los fantásticos momentos que hemos compartido juntas.

Agradecer al resto de mi familia por acompañarme siempre, por estar a mi lado. A mi abuela por presumir de nieta, a mis tíos por acogerme en casa cuando lo he necesitado, a mi tía por sus whatsapp a cualquier hora para saber cómo me iba, a mis padrastros por escucharme y entenderme incluso cuando yo no me entedía. En definitiva, gracias, y sólo espero poder devoleros el favor algún día.

También quisiera agradecer a mi tutor José María por la paciencia y dedicación que ha mostrado siempre conmigo, por su esfuerzo y por la curiosidad que ha despertado en mí por un mundo que hasta hace no tan poco no conocía.

Finalmente, no puedo olvidar agradecer la paciencia que han tenido todos mis amigos por estar tan ausente últimamente, pero sobre todo, por seguir pendientes de mí y mostrándome a cada momento su preocupación y apoyo.

¡Gracias!

Resumen

El sector de los videojuegos se ha visto obligado a innovar para seguir atrayendo a un público cada vez más exigente, esto les ha llevado a desarrollar nuevos interfaces que hagan la experiencia e interactuación de los usuarios más real a través de sensores RGBD como Kinect para Xbox o Xtion de Asus. Sin embargo, estos revolucionarios sensores no se limitan únicamente al ocio y a los videojuegos, sino que su papel en campos como la robótica y la visión computacional cada vez es más importante. Sin lugar a dudas, la aportación más relevante de estos sensores a la róbotica es la riqueza de información aportada, ya que proporcionan datos tridimensionales. Esto unido al potencial de la visión computacional y al asequible precio de estos sensores (unos 150 euros, mucho más baratos que los láseres), facilita el tratamiento de imágenes para reconocer situaciones u objetos.

El objetivo de este trabajo fin de grado es presentar un prototipo capaz de detectar y contar personas mediante el procesamiento de las imágenes de vídeo obtenidas a partir de sensores RGBD. Este prototipo consta de tres bloques funcionales. El primer bloque se encarga de la detección del fondo de la imagen de profundidad, por ejemplo usando mezcla de gaussianas. El segundo bloque identifica los objetos existentes en la imagen segmentando la imagen de primer plano en grupos de píxeles cercanos y conectados pertenecientes al mismo objeto real (*blobs*). El tercer bloque realiza un seguimiento a lo largo del tiempo haciendo corresponder los *blobs* observados con las personas estimadas hasta la iteración anterior del algoritmo.

Para el desarrollo del sistema hemos utilizado el lenguaje de programación C++, y nos hemos apoyado en la plataforma de software libre JdeRobot 5 y en varias librerías también libres (como OpenCV o PCL), sobre Ubuntu 14.04 de 64 bits.

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde la aparición de la computación se ha perseguido la automatización de procesos complicados, largos y con varias fases que puedan provocar que el error operacional aumente. El objetivo es automatizar al máximo estos procesos para simplificar los sistemas y disminuir los posibles errores. En el procesamiento de datos, es importante mencionar el análisis del entorno que nos rodea a partir de la información obtenida por sensores. Un sistema puede reaccionar de una forma u otra dependiendo del estímulo que le llegue con una mínima o nula intervención del usuario en el proceso.

La emulación de la percepción humana abre todo un abanico de posibilidades en la creación de programas y sistemas informáticos. Lograr que un algoritmo, por ejemplo, reconozca formas, contornos y colores y sea capaz de señalarlos, da pie a realizar cosas cada vez más complejas, como la identificación de personas o reconocimento de situaciones en una imagen o en un vídeo. El número de aplicaciones que se pueden diseñar es inmenso.

Este trabajo fin de grado se centra en la visión computacional y su aplicación en la detección y conteo de personas a partir de la información proporcionada por sensores RGBD. En este capítulo vamos a introducir los conceptos de la visión por computador y los usos de los sensores RGBD con la intención de servir de base para el resto de capítulos.

1.1. Visión por computador

La visión por computador, o visión artificial, es un subcampo de la inteligencia artificial como ilustra la figura 1.1. El propósito de la visión por computador es programar un computador para que entienda una escena o las características de una imagen. Por tanto, podemos decir que la visión por computador comprende todas aquellas herramientas y métodos utilizados para extraer, analizar y procesar la información visual de un entorno real. De esta manera, se pretende dotar a una máquina o sistema de una percepción y comprensión de su entorno empleando sistemas visuales.

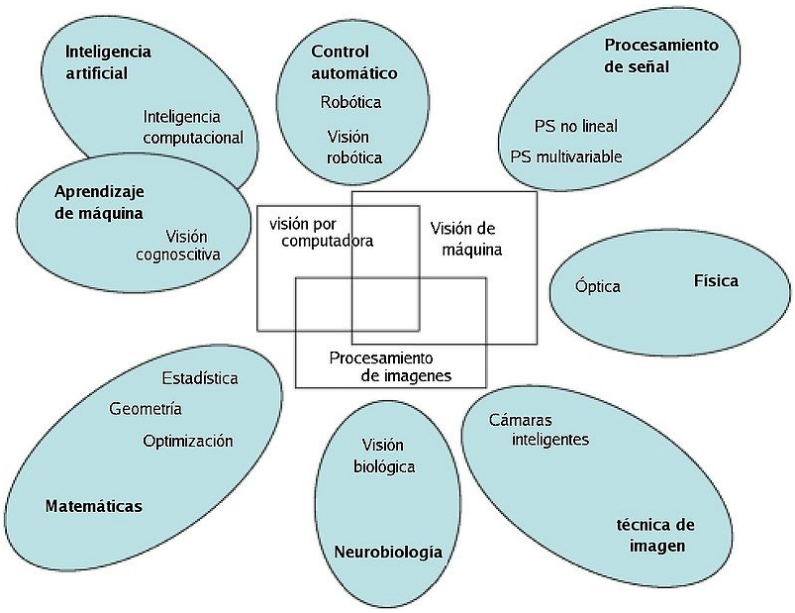


Figura 1.1: Esquema de las relaciones entre la visión por computadora y otras áreas afines.

La visión artificial surgió a raíz de un estudio publicado por Larry Roberts, uno de los ideólogos de ARPANET. Roberts en 1961 creó un programa que podía "ver" una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva, demostrando así a los espectadores que esa información visual que había sido mandada al ordenador por una cámara, había sido procesada adecuadamente por éste. A raíz de esto, muchos grupos de investigación se centraron en este nuevo campo.

Más tarde gracias a los avances de las cámaras se pudo incorporar al ordenador, que junto con el aumento de la potencia de los procesadores, posibilitó el diseño de algoritmos más potentes, pudiendo usarlos en tiempo real tanto con imágenes como con vídeos. Éstos últimos tienen una dificultad añadida: el sistema que se desarrolle tiene que dar continuidad y coherencia a los datos que se obtengan entre los distintos fotogramas de un vídeo.

Si analizamos la situación actual, podemos comprobar que hoy en día las cámaras están implantadas en todo nuestro entorno: en dispositivos móviles y ordenadores, sistemas de vigilancia, domótica, vehículos, robots industriales... A causa del aumento de la utilización de estas técnicas se ha desarrollado un entorno ideal para el crecimiento que está experimentando este campo.

Los objetivos típicos de la visión artificial incluyen:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (por ejemplo caras humanas).

- Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.

- Estimación de la postura de un ser humano.
- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la escena; este modelo podría ser usado por un robot para navegar por la escena.
- Estimación de la posición 3D de humanos y objetos.
- Búsqueda de imágenes digitales por su contenido.
- Procesado automático de imágenes para mejorar su calidad (reducción de ruido, mejora del contraste, realzado...)

Sin duda, parte de este avance ha sido posible a librerías libres como por ejemplo OpenCV[1] (ver figura 1.2), una librería de visión artificial desarrollada por Intel. Desde que apareció su primera versión en 1999 se ha utilizado en infinidad de aplicaciones, como sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esta librería contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estérea y visión robótica. El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Una de las aplicaciones más conocidas en las que se ha utilizado OpenCV es el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford, el ganador en el año 2005 del Gran desafío DARPA.

Figura 1.2: Librería OpenCV.

La visión artificial ha dado muy buenos resultados en todas esas tareas mediante técnicas muy diversas, y por ello está llegando prácticamente a todos los ámbitos de la sociedad como el industrial, videovigilancia, banca y administración, televisión y realidad aumentada, deportivo, biometría, medicina, tráfico o videojuegos.

[1] Opencv: `http://opencv.org/`

1.1.1. Aplicaciones de la visión por computador

El amplio abanico de aplicaciones cubierto por la visión por computador se debe a que permite extraer y analizar información espectral, espacial y temporal de los distintos objetos.

La información espectral incluye frecuencia (color) e intensidad (tonos de gris). La información espacial se refiere a aspectos como forma y posición (tres dimensiones). La información temporal comprende aspectos estacionarios y dependientes del tiempo (eventos, movimientos y procesos).

La mayoría de las aplicaciones de la visión artificial pueden ser clasificadas por el tipo de tarea que llevan a cabo:

- **Medición o calibración:** hace referencia a la correlación cuantitativa con datos del diseño, asegurando que las mediciones cumplan con las especificaciones del diseño.
- **Detección de fallas:** es un análisis cualitativo que involucra la detección de defectos o artefactos no deseados, con forma desconocida y/o en una posición desconocida.
- **Verificación:** es el chequeo cualitativo de que una operación de ensamblaje ha sido llevada a cabo correctamente.
- **Reconocimiento:** involucra la identificación de un objeto con base en descriptores asociados al objetos (ver figura 1.3a y 1.3b.
- **Identificación:** es el proceso de identificar un objeto por el uso de símbolos en el mismo.
- **Análisis de localización:** evaluación de la posición de un objeto.

(a)

(b)

Figura 1.3: (a) Detección de cuerpo humano. (b) Segmentación mano.

Un ejemplo de una aplicación que utiliza algunas de estas técnicas es *CamScanner*, se trata de una aplicación para móviles y tabletas que es capaz de escanear y digitalizar documentos o imágenes a partir de la imagen obtenida de la cámara del dispositivo

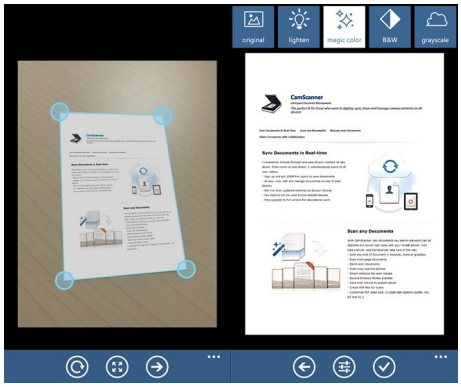
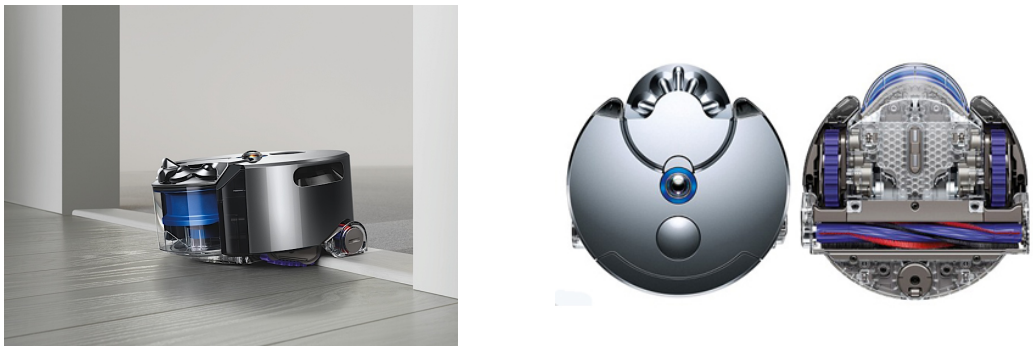


Figura 1.4: Aplicación móvil CamScanner.

móvil. La aplicación es capaz de detectar los bordes de la hoja y filtrar la imagen obtenida para limpiarla (ver figura 1.4).

En el campo de la investigación, el sistema de visión artificial de los robots utiliza cámaras de vídeo como receptor de estímulos del exterior. Es esencial para que el robot pueda decidir cómo actuar en cada caso. Un ejemplo de esto es la aspiradora robótica Dyson. La revolución del sistema de Dyson 360 Eye (ver figura 1.5) es su cámara de 360 grados que captura 30 fotos por segundo con una inclinación de 45 grados para realizar un mapa de la estancia. Posteriormente, el robot comienza su recorrido sistemático localizando su propia ubicación con respecto al centro de la habitación y se desplaza hasta él para comenzar el proceso de limpieza. A partir de ahí aspira en espiral desde ese punto, analizando en todo momento el espacio por si se le ha quedado algo atrás. Al terminar el proceso regresa por sí mismo a la base.



(a) (b)

Figura 1.5: Aspirador robótico Dyson 360 Eye.

Uno de los campos de actuación más propicios para los drones en el ámbito civil es el de la agricultura. Y es que los drones ofrecen múltiples posibilidades para la agricultura. Pueden sobrevolar los campos de una forma rápida y captar información diversa gracias a sus sensores. Esto permite que aquellos que gestionan los cultivos tengan a su disposición una herramienta para controlar e incrementar la productividad. Un solo dron puede monitorizar cientos de hectáreas de forma precisa, evaluando las condiciones del terreno, con el fin de recoger información sobre la hidratación, la temperatura o

el ritmo de crecimiento de los cultivos. Una de las funciones más importantes que se atribuyen a estos dispositivos es la localización prematura de enfermedades y el uso selectivo de herbicidas sólo en las zonas afectadas. De esta forma se pueden evitar plagas que arruinen parte de la cosecha. Los dispositivos pueden controlar cómo funciona el riego y son capaces de enviar fotografías e incluso vídeo en tiempo real a un centro donde se observe el estado de los cultivos. Este tipo de operaciones ya se han puesto en práctica en algunos lugares. Uno de los países más avanzados en este sentido es Japón.

Otro ejemplo de las aplicaciones que ofrece la visión artificial es el reconocimiento automático de matrículas. Este método de vigilancia en masa utiliza el reconocimiento óptico de caracteres en imágenes para leer las matrículas de los vechículos. Estos sistemas pueden escanear matrículas con una frecuencia aproximada de una por segundo en vehículos con velocidades de hasta 160 km/h. Son utilizadas por las diversas fuerzas de policía y como método de recaudación electrónica de peaje en las autopistas de pago, y para vigilar la actividad del tránsito como una luz roja en una intersección.

La conocida aplicación de Google Street view también utiliza algunas de estas técnicas. Google ha repartido coches equipados con cámaras de 360º en el techo y una unidad GPS por las calles y carreteras de medio mundo, de tal modo que toda imagen captada está georeferenciada inequivocamente en un punto del globo terráqueo. Obviamente, Street View ha sido objeto de duras críticas por suponer una amenaza a la privacidad de las personas que aparecen en las fotografías tomadas por la empresa. Para evitar situaciones embarazosas, Google emplea un algoritmo que automáticamente emborrona las caras de las personas y las matrículas de los coches.

Existen muchos más campos donde las técnicas de visión por computador son realmente esenciales. Esto es sólo una muestra de lo integradas que están ya estas técnicas en nuestro día a día. Es evidente que un futuro muy interesante se avecina mientras se desarrollan cada vez mejores sistemas basados en la visión artificial.

1.1.2. Reconocimiento de objetos

Un área interesante de visión artificial es el reconocimiento de objetos que se entiende como la tarea para encontrar e identificar objetos en una imagen o secuencia de vídeo. Los humanos reconocemos multitud de objetos en imágenes con poco esfuerzo, a pesar del hecho de que la imagen del objeto puede variar un poco en diferentes puntos de vista, en diferentes tamaños o escala e incluso cuando están trasladados o rotados. No obstante esta tarea es un desafío para los sistemas de visión artificial. Para este problema se han implementado muchos métodos durante las últimas décadas.

Métodos basados en apariencia

Utiliza imágenes de ejemplo que se denominan plantillas o prototipos de los objetos a analizar. Sin embargo, hay que tener en cuenta que los objetos pueden verse diferentes bajo una variedad de condiciones como cambios en la iluminación o color, en la dirección de observación o en el tamaño y forma.

Por estos motivos, es muy improbable lograr un éxito absoluto con un solo prototipo de imagen. Sin embargo, sí es posible representar todas las apariencias de un objeto para lograr tener una base de datos lo más completa posible.

Métodos basados en características

Este método se basa en la búsqueda de emparejamientos factibles entre las características del objeto y las caracerísticas de la imagen analizada. La principal restricción de este método es que la única posición del objeto debe contar para todos los emparejamientos posibles. Los métodos utilizados para extraer características de un objeto para ser reconocido son parches de superficie, esquinas y bordes lineales.

Una característica se describe mediante una serie de propiedades que van a permitir establecer las correspondencias por similitud entre los valores de las propiedades de una característica en una imagen modelo y características en la imagen a analizada. Cada una de las propiedades tenidas en cuenta posee un peso específico concreto, de forma que su importancia relativa queda fijada por el valor de dicho peso a la hora de establecer la correspondencia.

1.1.3. Seguimiento de objetos

Las técnicas de detección de objetos aplicadas a imágenes pueden utilizarse también para analizar vídeo, ya que un vídeo se compone de un conjunto de imágenes, ordenadas de tal manera, que al mirar sucesivamente 24 de estas imágenes en un segundo, se percibe la sensación de movimiento. De este modo, existe una correlación entre sí en este conjunto de imágenes, aportando coherencia entre ellas.

Cuando se quieren detectar elementos de interés en un vídeo es recomendable hacer un seguimiento de estos objetos a lo largo de todos los fotogramas. El seguimiento visual de objetos pretende desarrollar técnicas para identificar objetos de interés en un vídeo y relacionarlos entre sus distintos fotogramas.

Uno de los sectores que más partido saca de las ventajas que brinda el seguimiento visual de objetos es el de la seguridad. En la videovigilancia, el seguimiento visual de objetos puede lograr que un programa informático haga un seguimiento de las personas que capte la cámara, e incluso tener la capacidad de detectar ciertas situaciones y avisar de ellas en tiempo real (ver figura 1.6a y 1.6b).

1.2. Sensores RGBD

Como ya hemos dicho, el sector de los videojuegos es uno de los sectores que más hincapié ha hecho en la incorporación de interfaces basados en la visión artificial para la interacción del usuario con la consola. Esta revolución comenzó en 2006 con la salida al mercado del *Wiimote* (ver figura 1.7a). Desde entonces todas las empresas del

(a) (b)

Figura 1.6: Videovigilancia. (a) Detección de personas (b) Detección de vehículos.

sector han invertido grandes fondos para lograr interfaces adaptados a sus consolas que utilizasen la visión por computador.

Sony también quiso lanzar un dispositivo basado en visión artificial adaptado a sus consolas, por este motivo lanzó *EyeToy* (ver figura 1.7b). Este dispositivo está compuesto básicamente por una cámara y un micrófono. A partir de las imágenes obtenidas por este periférico, los juegos de Sony eran capaces de reconocer una serie de gestos con los que el usuario interactuaba con la consola.

Finalmente, fue en 2010 cuando Microsoft lanzó al mercado un sensor para su consola Xbox 360 que revolucionó el mercado (ver figura 1.7c). Este periférico está formado por una cámara RGB, un sensor de profundidad y un micrófono matricial. La novedad que aportaba Kinect era que además de poder reconocer gestos preestablecidos, permite calcular la posición de cada una de las articulaciones del usuario lo que aumenta las opciones de interactuación con la consola. La auténtica revolución del sensor Kinect de Microsoft llegó en el sector de la investigación, principalmente en el campo de la róbotica e inteligencia artificial basada en la visión ya que facilitó el acceso a la comunidad de investigadores, proporcionando un enorme crecimiento en las líneas de la reconstrucción y la localización 3D.

(a) (b) (c)

Figura 1.7: (a) Wiimote de Nintendo. (b) Eyetoy de Sony. (c) Kinect de Microsoft.

1.2.1. Mapeado a partir de sensores RGBD

La generación de mapas desde visión es un campo de la visión por computador que ha suscitado el interés de muchos investigadores en los últimos años. Consiste en construir una representación del entorno que rodea al sensor únicamente a partir de la información visual recibida.

En el campo de la robótica y la visión artificial existe un gran interés en desarrollar sistemas que sean capaces de generar información 3D para su uso en multitud de aplicaciones, especialmente en la localización y en el seguimietno. Existen diferentes técnicas para la reconstrucción de objetos y/o escenas como por ejemplo la triangulación, el uso de técnicas de tiempo de vuelo, sistemas de proyección de luz estrucuturada, aplicaciones basadas en detección de silueta y sombreado y o técnicas de flujo óptico. Estas técnicas cada vez están más extendidas.

Tanto Xtion (ver figura 1.8a) como Kinect-1 (ver figura 1.8b) son sensores que utilizan proyección de luz estructurada. Los escáneres de luz estructurada basan su funcionamiento en la emisión de un patrón de luz en la escena, captan la deformación del patrón con un receptor y analizan estas deformaciones para calcular la distancia a la que se encuentran los objetos, está técnica es relamente rápida.

Gracias a su bajo coste y a sus buenas prestaciones, este tipo de sensores han tenido un gran aceptación por parte del público del sector de la investigación. Los componentes de este completo dispositivo son: emisor de infrarrojos, receptor de infrarrojos, cámara de color, micrófono matricial y motor.

El funcionamiento de estos sensores consiste en que el proyector de infrarrojos emite un patrón de luz con una estructura determinada, la cámara VGA infrarroja obtiene una imagen con la intensidad con la que dicho patrón rebota en los objetos de la escena y analizando las deformaciones sufridas por el patrón se obtiene la distancia correspondiente a cada píxel de la cámara. El inconveniente de estos sensores es que debido a la luz infrarroja que utilizan, estos periféricos sólo pueden utilizarse sin presencia de luz solar, ya que la luz solar produce serias interferencias al recibirse junto con los rebotes del patrón emitido, lo que impide la correcta estimación de las distancias.

(a) (b)

Figura 1.8: (a) Xtion (b) Kinect-1

En julio de 2014 junto con el lanzamiento de la nueva Xbox One, apareció el kinect-

2 que mejora las prestaciones de su primera versión. El kinect-2 es un sensor basado en técnicas de tiempo de vuelo (TOF). Los escáneres TOF basados en tiempo de vuelo determinan la distancia del objeto a la cámara cronometrando el tiempo que tarda un pulso de luz láser en ir del emisor al objeto y volver al receptor. Estos sensores sólo miden la distancia directa de un punto por lo que es necesario variar la dirección del haz tras cada medida. Son sistemas de muy alto muestreo, de alta precisión (el error medio es inferior a un milímetro) y gran alcance. La principal pega de estos sensores es que no son capaces de funcionar contra superficies reflejantes como espejos, ni contra superficies transparentes como el cristal.

Los sensores RGBD son muy importantes y útiles como herramienta para la resolución práctica de la reconstrucción 3D. A lo largo de este trabajo serán el soporte para extraer información y detalles acerca del entorno presente para después, tras el procesamiento de la imagen llegar a una solución para la detección y conteo de personas.

Algunos ejemplos de aplicaciones que utilizan este tipo de sensores es por ejemplo Google Tango[2]. La clave de Google Tango es que es un proyecto colaborativo que une el conocimiento de más de unas pocas compañías trabajando en tecnología punta. Esto incluye un chip de procesamiento de visión de baja potencia llamado Myriad 1 que es capaz de soportar los complejos algoritmos necesarios para visión computarizada sin arrasar con la batería del *smartphone*. El Google Tango que existe actualmente es un prototipo de teléfono Android de cinco pulgadas, con hardware y software a medida capaz de registrar el "movimiento 3D completo del dispositivo mientras crea un mapa del entorno de manera simultánea", dice Google. Las aplicaciones sugeridas van desde lo más mundano, como medir las dimensiones de tu casa antes de comprar muebles simplemente moviendo tu teléfono alrededor de la habitación, hasta lo más útil, como ayudar a las personas con discapacidades visuales en el interior de edificios desconocidos, pasando por lo frívolo, como convertir un pasillo en el espacio de un juego de realidad virtual.

Tal es la importancia de esta tecnología que incluso la conocida multinacional Apple compró en noviembre de 2013 la compañía PrimeSense[3]. PrimeSense es popularmente conocida por ser la desarrolladora del sensor de movimiento Kinect de la Xbox de Microsoft. Como ya hemos dicho anteriormente, este sensor es capaz de capturar mediante cámaras los movimientos de los jugadores para incorporarlos a los juegos. Además, PrimeSense ha expandido en los últimos años su línea de productos de manera que ha incluido más hardware al crear nuevos sensores para dispositivos más compactos. Por ejemplo, el modelo Capri de la compañía, diseñado para el mercado de dispositivos móviles.

Finalmente, encontramos algoritmos que reconstruyen mapas 3D en tiempo real con estos sonsores. Encontramos muchos avances por parte de la comunidad científica en los últimos años. Por ejemplo, encontramos *KinectFusion* que permite a un usuario sostener y mover una cámara estandar Kinect para crear con rapidez detalladas reconstrucciones en 3D de un escenario interior (ver figura 1.9). Para la reconstrucción geométrica

[2]Web Google Tango: `https://www.google.com/atap/project-tango/`
[3]`https://www.crunchbase.com/organization/primesense`

de modelos precisos en 3D de la escena física real, Kinect sólo utiliza los datos de profundidad para seguir la postura del sensor 3D. En Youtube encontramos multitud de ejemplos de este software[4]

Figura 1.9: Escena reconstruida gracias a KinectFusion.

1.3. Seguimiento de personas y sensores RGBD en el Grupo de Robótica de la URJC

En el Grupo de Robótica de la URJC, existen una serie de trabajos previos que estudian temas relacionados con el seguimiento visual de personas, donde los algoritmos que se utilizan han ayudado al aprendizaje y comprensión de diversas técnicas.

1.3.1. ElderCare

ElderCare[5] nace dentro del Grupo de Robótica de la Universidad Rey Juan Carlos con el fin de complementar, a través de información tridimensional, a los sitemas convencionales de teleasistencia. Haciendo uso de datos 3D, *ElderCare* es capaz de detectar situaciones de riesgo para personas como pueden ser caídas.

Esta tecnología se empieza a orientar hacia el campo de la teleasistencia con el trabajo fin de carrera *Seguimiento 3D de múltiples personas utilizando un algoritmo evolutivo multimodal* [2]. En este trabajo se presentan mejoras en las técnicas de seguimiento con el fin de localizar a múltiples sujetos y donde el propio sistema es capaz de aprender automáticamente el color de cada uno de ellos. La misma autora realizó dos evoluciones más de ElderCare [3] [4], donde se introdujeron grandes novedades:

- Introducción de primitivas 3D volumétricas (ver figura 1.10).
- Mejoras de robustez en el aprendizaje del color utilizado.

[4]Vídeo demostración KinectFusion: `https://www.youtube.com/watch?v=zzb_RQWrt6I`
[5]Web: `http://jderobot.org/ElderCare`

- Desarrollo del soporte en Android para visualizar el entorno monotirzado desde el exterior utilizando dispositivos móviles.
- Pruebas piloto en entornos reales (ver figura 1.11).

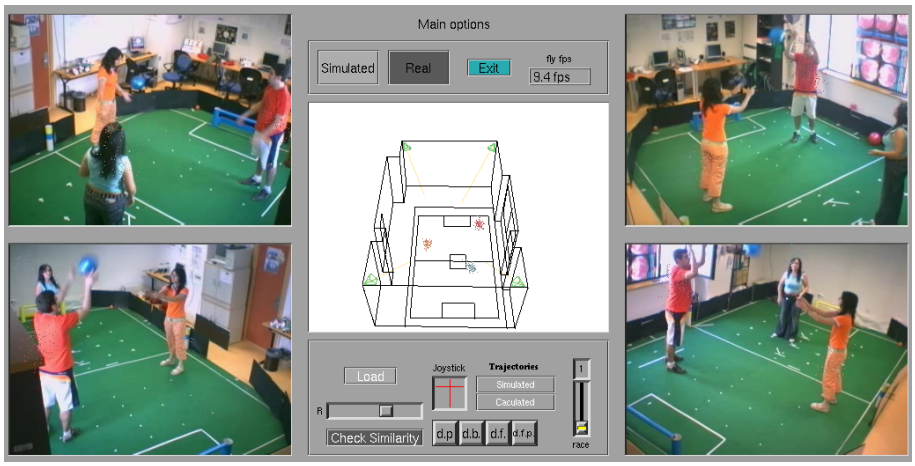


Figura 1.10: Aplicación ElderCare.

Figura 1.11: Aplicación ElderCare en entornos reales.

Posteriormente, en el trabajo *Seguimiento de personas en 3D basado en sensores RGBD* [7] supone una evolución más de *ElderCare*, el cual conlleva un cambio completo del sistema, teniendo en cuenta que el sensor base, que hasta la fecha habían sido cámaras de color convencionales, será reemplezado por sensores de profundidad tipo Kinect. Sin cambiar el enfoque teleasistencial de *ElderCare*, el objetivo principal del proyecto era desarrollar un sistema robusto de seguimiento tridimensional de personas utilizando sensores RGBD capaz de funcionar las 24 horas del día. *ElderCare* ha seguido desarrollándose en varios trabajos como el Trabajo Fin de Máster *Algoritmo evolutivo para detección y seguimiento de personas en 3D con sensores RGBD* [6].

Este programa, de forma totalmente autónoma, realiza un análisis simultáneo de las imágenes captadas desde diversas cámaras de una habitación concreta. De esta forma, se puede realizar un seguimiento en 3 dimensiones a una o varias personas a través de una habitación. El sistema también puede detectar si una persona cae al suelo y en ese caso, si se mantiene inmóvil durante un tiempo determinado, salta una alarma.

ElderCare se basa en un algoritmo de seguimiento evolutivo, tanto en 2D como en 3D, utilizando técnicas de cálculo de puntos característicos, filtrado de imágenes por color en el modelo HSV y más tipos de filtros, junto con técnicas basadas en cálculos geométricos para el seguimiento 3D, con el fin de aportar robustez al proceso de seguimiento y obtener resultados fiables.

1.3.2. Mapas 3D desde sensores RGBD

En el Grupo de Robótica de la URJC existen más trabajos relacionados con el seguimiento visual de objetos y sensores RGBD por ejemplo el proyecto fin de carrera de Juan Navarro [5]. En su proyecto *Construcción de mapas 3D compactos desde sensores RGBD* presenta un sistema capaz de generar en tiempo real una representación 3D compacta del entorno en forma de parches plano desde la información cruda de sensores RGBD que se pueden mover (ver figura 1.12). Para este fin, utiliza técnicas como RANSAC, SVD y PCA.

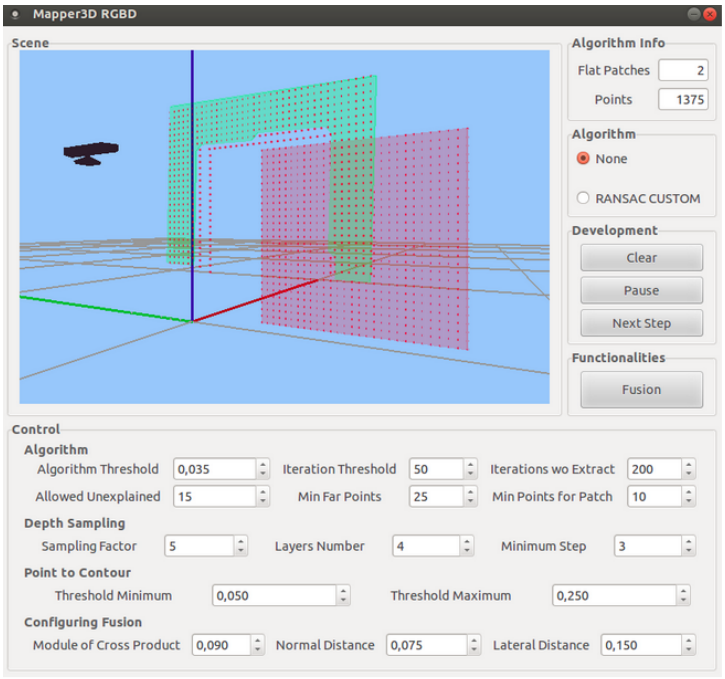


Figura 1.12: Interfaz gráfica mapeador desde datos RGBD.

En cada iteración del algoritmo se capta primero la pose3D, con la posición y la orientación del sensor, y a continuación se recibe la imagen de profunidad del sensor RGBD. En la figura 1.13 aparece un esquema del funcionamiento del componente.

Este componente utiliza los datos provenientes de la imagen de profundidad del sensor RGBD en lugar de la nube de puntos, porque la nube de puntos es demasiado voluminosa. Si se quiere muestrear se debe hacer un muestreo dependiente de la profundidad, debido a que la densidad de puntos cercanos al sensor es mayor que la densidad de puntos lejanos. Es decir, se debe filtrar en mayor proporción los puntos cercanos e ir disminuyendo la cantidad de puntos eliminados según nos alejamos del sensor. Esto es más sencillo partiendo de la imagen de profundidad. Para ecualizar la densidad espacial de puntos 3D en distintas distancias y obtener una nube de puntos con la que trabajar partiendo de la imagen de profundidad, separa en distintas capas los puntos de la imagen según su distancia hasta el sensor.

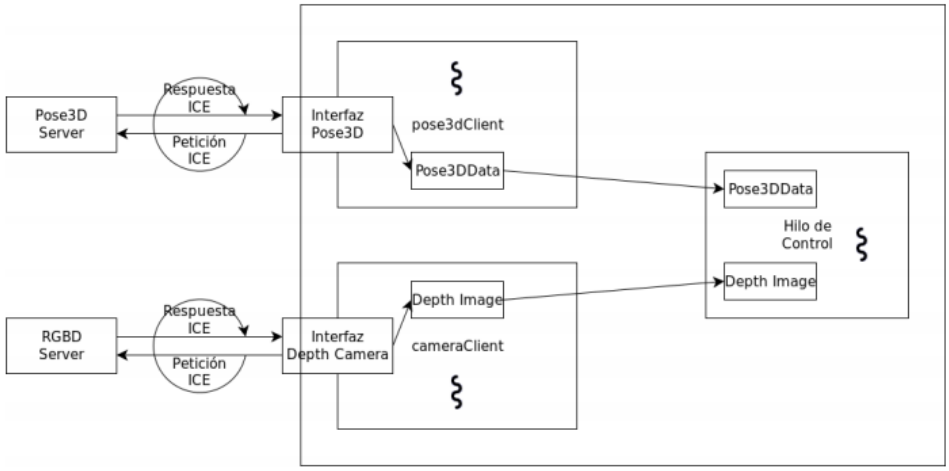


Figura 1.13: Diagrama de funcionamiento del mapeador desde datos RGBD.

A continuación, a los puntos de la imagen de profundidad que pasan el filtrado, se les calcula sus coordenadas relativas al sensor empleando la distancia a este, y con la pose3D obtenida de la iteración actual se convierten a coordenadas absolutas. Todos estos puntos con sus coordenadas absolutas forman la nube de puntos que se emplea en el resto del algoritmo. Gracias a esto el algoritmo podría segmentar los planos provenientes de varios sensores RGBD si todos se expresan en el mismo marco absoluto de referencia.

Antes de extraer parches planos los puntos se clasifican según la información que puedan aportar. Para ello se distinguen tres categorías según su relación con los parches planos extraídos hata el momento:

- *Perteneciente a alguno de los parches planos:* Estos puntos no aportan información nueva. Sólo confirman a algún parche ya existente.
- *Cercano a alguno de los parches planos:* Estos puntos son útiles para extender y redefinir los parches planos a los que están asociados.
- *No explicado por ningún parche existente:* Con estos puntos se estudiará la posibilidad de extraer nuevos parches planos que aún no se hayan detectado.

Para realizar está clasifiación, a cada punto de la nube se le calcula su distancia con respescto a todos los planos ya localizados. Un punto que pueda clasificarse en distintas categorías respecto a varios parches planos es clasificado finalmente en base a la condición más estricta y al parche plano cuyo error sea menor.

Los próximos capítulos explican de forma más detallada los objetivos principales del proyecto y la infraestructura utilizada. Después expondremos los diversos algoritmos probados hasta llegar a los empleados en la solución propuesta. Finalmente se plantearán las conclusiones obtenidas y las líneas de desarrollo futuro que deja abiertas este proyecto.

Capítulo 2

Objetivos

En este capítulo vamos a definir el problema que abordaremos dividido en subobjetivos. También vamos a presentar los requisitos necesarios tanto para el desarrollo como para la solución de este proyecto, la metodología seguida, y el plan de trabajo empleado para lograr la solución presentada.

2.1. Descripción del problema

El objetivo principal de este Trabajo Fin de Grado es desarrollar e implementar en un componente software un algoritmo capaz de detectar y contar personas a partir de la información obtenida de sensores RGBD. El programa recogerá imágenes de vídeo de una sola cámara, siendo ésta la única fuente de datos. Se quiere tener un prototipo funcional que sirva de prueba de concepto. Dicho objetivo se ha articulado en subobjetivos más fácilmente abordables:

1. *Detección de primer plano:* El primer paso consiste en ser capaces de diferenciar el primer plano del fondo de la imagen de distancias obtenida del sensor RGBD. Para ello se estudiarán técnicas de visión artificial como la mezcla de gaussianas o la sustracción de imágenes y se integrarán en el sistema.

2. *Detección de blobs:* El segundo subobjetivo consiste en programar un módulo de detección de blobs. Este detector utiliza como punto de partida la imagen umbralizada, proporcionada por el primer bloque, que distingue fondo del primer plano.

3. *Seguimiento de personas:* Cuando hayamos diferenciado los objetos de la imagen, el último paso consistirá en lograr un seguimiento continuo de las personas.

2.2. Requisitos

El desarrollo y la solución final de este proyecto deben satisfacer además los siguientes requisitos:

- *Implementación del sistema basado en la plataforma JdeRobot 5.2:* Este requisito facilita y reduce el tiempo necesario para el desarrollo de todos los elementos del proyecto. Esta plataforma permite abstraer el problema de adquisición de información de los sensores, facilitar un desarrollo modular interoperable y aprovechar componentes ya desarrollados en la plataforma.
- *Funcionamiento del sistema en tiempo real en hardware convencional:* Debe presentar un comportamiento vivaz y ágil que permita que otros componentes puedan utilizar la información aportada para distintos propósitos.
- *Modularización de todos los bloques del sistema:* Para facilitar el uso de la implementación de este prototipo y permitir su reutilización tanto en las distintas fases de implementación como en futuros proyectos que puedan hacer uso del mismo, interesa que la solución sea lo más modular posible.
- *Sistema operativo y lenguaje:* El sistema operativo sobre el que transcurrirá el desarrollo del proyecto será GNU/Linux, dado que es lo utilizado en el laboratorio de Robótica. Por otro lado, el lenguage de programación que se utilizará para el desarrollo del proyecto será C++.

2.3. Metodología

A la hora de realizar o trabajar en un proyecto de cierta envergadura es necesario establecer una metodología para mantener un cierto orden durante todos los pasos y fases del desarrollo del mismo. Si se encuentra la metodología idónea, se puede obtener mayor rendimiento del tiempo dedicado a la creación de un software, pudiendo incluso a llegar a optimizar hipotéticos costes en el desarrollo de un proyecto.

La metodología utilizada en este trabajo fin de grado está basada en el modelo incremental. Este modelo consiste en agrupar una serie de tareas en distintas etapas en las que se busca consguir un pequeño hito dentro del desarrollo del sistema. La figura 2.1 ayuda a comprobar que la estructura básica de cada etapa repite a la anterior.

La principal ventaja de este modelo es que permite añadir cada vez más funcionalidad a un programa en desarrollo, a partir de la versión más reciente de éste, cumpliendo poco a poco con los objetivos que se hayan establecido. También posibilita añadir nuevas funcionalidades al sistema una vez acabado. Otro de los beneficios que ofrece este modelo es que permite probar el software desde etapas muy tempranas, detectar errores y corregirlos en alguna de las siguientes etapas. Esto permite a los desarrolladores adaptarse a la situación que se presenta con cada etapa, pudiendo incluso rectificar la planificación del proyecto sin muchos inconvenientes si fuera preciso.

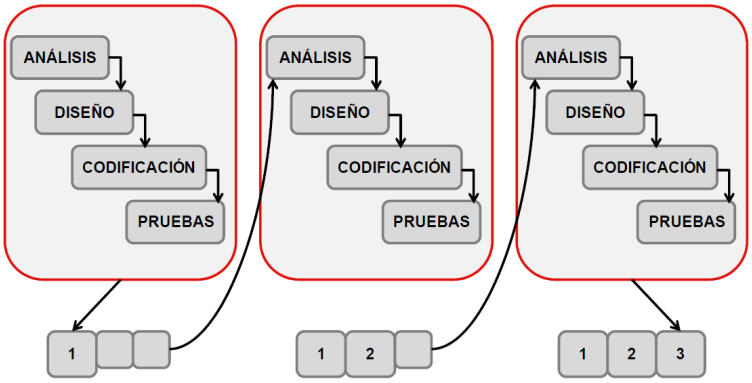


Figura 2.1: Esquema del desarrollo incremental

Dada la flexibilidad que proporciona este modelo, es lo más adecuado para un proyecto de este tipo, ya que permite la inclusión de requisitos que en un principio no estaban incluidos cuando se establecieron la lista de objetivos y requerimientos que debía tener este sistema.

Durante el desarrollo del proyecto se mantendrá un registro en la página web del grupo de Robótica en formato *wiki*[1], donde se explicará cada avance conseguido e integrado en el sistema, se publicarán vídeos e imágenes de cada funcionalidad nueva implementada y se mostrarán los resultados. Por otro lado, se subirán todos los avances a un repositorio de código fuente *subversion*[2] públicamente accesibles. También se llevarán a cabo reuniones periódicas de seguimiento con el tutor, las cuales servirán para llevar a buen puerto la consecución de los objetivos establecidos para este proyecto.

2.4. Plan de trabajo

A lo largo del desarrollo de este proyecto, se han propuesto una serie de etapas siendo asesorada, y en parte fijado, por el supervisor del proyecto en las diferentes tutorías que han tenido lugar. Las principales tareas a descatar son:

1. **Aprendizaje de la infraestructura de JdeRobot 5:** Para empezar a abordar este proyecto, se instalará JdeRobot y se empezará a estudiar su arquitectura y el funcionamiento de sus módulos más significativos. Se instalarán y ejecutarán los componentes *CameraServer* y *CameraView* para comenzar a entender su funcionamiento y la interacción entre módulos del entorno. También se probará el componente *Introrob* en el simulador de entorno *Gazebo* dedicado al control telemático o programado de robots virutales en escenarios pregenerados.

2. **Aprendizaje de bibliotecas de tratamiento de imágenes:** Se instalarán manualmente las librerías de OpenCV. Se probarán el componente *OpenCVDemo*

[1]Mediawiki: `http://jderobot.org/rsaezd`
[2]Repositorio SVN: `http://svn.jderobot.org/users/rsaezd/tfg/`

de JdeRobot. Se leerá el código de *OpenCVDemo* para comprender el funcionamiento de todos los métodos de OpenCV que utilizará este componente. Se programará un pequeño componente para experimentar con la funciones de la biblioteca OpenCV. También se estudiarán diversas técnicas de detección de fondo y segmentación de objetos, utilizando mezcla de Gaussianas y diferencia de imágenes entre otras, o técnicas como *watershed*. Finalmente, se programará un componente para cononcer las funciones de la visión de nube de puntos que ofrece la biblioteca PCL.

3. **Estudio de las herramientas de desarrollo:** Se analizará el funcionamiento interno del componente OpenniServer, para adaptar nuestro componente a este componente de JdeRobot. En este punto, se comprobará el funcionamiento de *ICE* al tener que comunicarse nuestra aplicación con *OpenniServer*. También se aprenderá a utilizar la herramienta *Make* y a configurar la compilación de código mediante los *makefile*.

4. **Aprendizaje sobre la utilización de herramientas gráficas:** Se aprenderá a utilizar la librería gráfica para C++ Qt. También se aprenderá cómo gestionar las señales que se lanzan desde el GUI usando métodos de Qt.

5. **Implementación de los algoritmos de extracción de primer plano:** Se desarrollarán algoritmos de detección de fondo utilizando diferentes métodos como detección por gaussianas o diferencia de imágenes, apoyándose en funciones de la librería OpenCV.

6. **Implementación de algoritmo de segmentación:** Se utilizarán los métodos de las librerías OpenCV y cvBlob para realizar la detección de personas, aunque antes se tendrán que adaptar dichos métodos para poder utilizarlos en C++.

7. **Implementación de algoritmo de seguimiento:**

Capítulo 3

Infraestructura

Este Trabajo Fin de Grado se ha basado en software libre de terceros para su desarrollo. El sistema operativo sobre el que se ha trabajado ha sido una distribución Ubuntu de 64 bits, concretamente Ubuntu 14.04 LTS por su estabilidad y por la disponibilidad de todas las herramientas necesarias para este proyecto. El lenguaje que se ha utilizado para crear este programa ha sido C++ ya que permite adaptarse con más facilidad al lenguaje en el que está implemetanda la plataforma software en la que el componente se apoyará.

En este capítulo se describirán tanto los sensores utilizados como la plataforma base software y las librerías empleadas.

3.1. Sensores RGBD

Con Kinect apareció un sensor que ofrece de forma simultánea una imagen en color y el mapa de profunidad correspondiente. Es capaz de funcionar a unos 30 fotogramas por segundo y su coste es relativamente bajo. El rango de este sensor está entre los 0.8 y 3.5m según las especificaciones del fabricante, aunque realmente puede llegar hasta los 8-10 metros. Microsoft lo acotó a ese rango porque es donde tiene mayor precisión su algoritmo de reconocimiento gestual, y por lo tanto, es donde desempeña correctamente su función de interfaz con la consola.

Debido a que se empezaron a popularizar este tipo de sensores en el campo de la visión por computador, la empresa taiwanesa Asus comenzó a comercializar un sensor RGBD llamado Xtion. Este sensor está basado en el mismo chip que utiliza Microsoft en Kinect. Xtion se comercializó como un sensor susceptible de ser incorporado en aplicaciones de visión artificial, facilitando su conexión a través de USB 2.0 y alimentando e incorporando todas la herramientas necesarias para acceder a él.

Nuestro componente contador de personas funciona indistintamente con ambos sensores RGBD, ya sea el Xtion de Asus o el Kinect-1 de Microsoft.

3.2. JdeRobot

JdeRobot[1] es una plataforma de software libre para el desarrollo de aplicaciones en robótica, visión por computadora, domótica y escenarios con sensores, actuadores y software inteligente. Esta plataforma es desarrollada dentro del grupo de robótica de la Universidad Rey Juan Carlos. El proyecto hace uso de la versión 5.2.

En JdeRobot las librerías y componentes principales están programados en C y C++, aunque por su sistema de interconexión permite el desarrollo de componentes en otros lenguajes de programación y su distribución sobre distintos sistemas operativos. Esta interconexión se basa en el *middleware* ICE[2], desarrollado por la empresa ZeroC y disponible bajo licencias GNU GPL o comercial. Está enfocado a facilitar el desarrollo de aplicaciones distribuidas cubriendo sus necesidades de comunicación entre módulos. Mediante el lenguaje *SLICE (Specication Language for ICE)* se definen las distintas interfaces de la plataforma, siendo después traducidas al lenguaje o lenguajes de programación de los componentes.

La plataforma JdeRobot se basa en distintos componentes. Un *componente* es la unidad modular funcional e independiente más básica con una funcionalidad concreta. Los componentes pueden trabajar de forma distribuida e interoperan entre sí a través de una serie de interfaces estandarizados. De esta forma, los módulos encargados de obtener información de sensores o enviar una orden a los actuadores abstraen su funcionamiento interno y pueden ofrecer interfaces similares a aquellos que hacen uso de estos elementos.

Los *drivers* están destinados a interactuar con dispositivos finales, sean reales o simulados. Entran en esta categoría componentes que actúan como fuentes de información sensorial, recogiendo y tratando los datos de los sensores para poder enviar su información a través de una o más interfaces. También pertenecen a este tipo sumideros que reciben información por una o más interfaces y la traducen en órdenes para los actuadores. Un ejemplo de los *drivers* genéricos que ofrece la plataforma es `CameraServer`, que sirve información de un sensor de tipo cámara.

En este proyecto, se ha utilizado un *driver* específico para poder al sensor RGBD:

OpenNIServer Es un servidor que utiliza OpenNi para dar soporte a diversos sensores RGBD. Este driver es capaz de proporcionar las imágenes de la cámara de color, el mapa de profundidad y también ofrece la posibilidad de combinar las dos anteriores en forma de nube de puntos. Como se puede observar en la figura 3.1, OpenNI provee a las aplicaciones una interfaz de comunicación con los diferentes sensores hardware así como también, con los diferentes componentes middleware que se registren en el framework. Un punto a destacar es que provee únicamente las interfaces, relegando la lógica que implementa cada funcionalidad a los diferentes componentes, sean sensores o middleware; algunas operaciones de la API estarán implementadas por los dispositivos hardware y otras por los

[1] Página web oficial del proyecto JdeRobot: `http://jderobot.org`

[2] `http://www.zeroc.com/ice.html`

componentes middleware. Así, por ejemplo, OpenNI provee operaciones para obtener el mapa de profundidad (mapa en el cual cada punto representa la distancia desde el sensor) de la escena o los datos crudos de imagen RGB desde los sensores, cuya lógica estará implementada en el propio sensor. Además, se pueden tener registrados varios componentes que implementen la misma funcionalidad y elegir cuál de ellos utilizar al momento de solicitar los datos. Esta API común provista por OpenNI lo hace extremadamente flexible para la incorporación de nuevos componentes middleware así como también para el desarrollo y ejecución de aplicaciones, que pueden utilizar la API de forma trasparente e independiente del dispositivo hardware (sensores) que genere los datos.

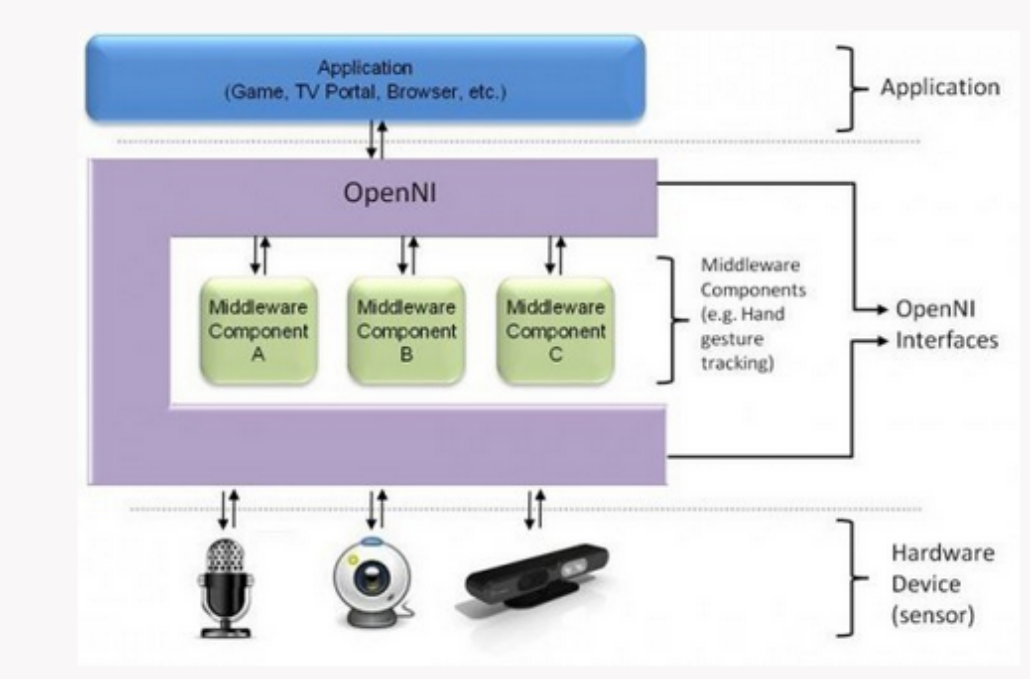


Figura 3.1: Arquitectura en capas de OpenNi.

Las *herramientas* están pensadas para conectarse a otros componentes de JdeRobot y extender su funcionalidad o permitir la explotación de la misma. Al contrario que los anteriores, no suelen interactuar directamente con el dispositivo final. Entran en esta categoría componentes de tipo visor, controladores manuales, generadores y registradores de datos derivados de los sensoriales, y algoritmos para detección o actuación en general.

Por otro lado, las *interfaces* son los elementos de intercambio de información entre componentes de JdeRobot.

Además de interfaces y componentes básicos, la plataforma ofrece una serie de librerías que simplifican distintas formas de intercomunicación entre componentes u operaciones relacionadas con el ámbito de aplicación de JdeRobot.

3.3. Point Cloud Library

Point Cloud Library (PCL[3]) es una librería escrita en C++ para el tratamiento de imágenes 2D, mapas de profundidad y nubes de puntos. Está publicada bajo una licencia BSD de software libre, lo que permite su uso tanto para aplicaciones comerciales como de investigación. Ha sido desarrollada por un gran consorcio de ingenieros e investigadores de todo el mundo, Willow Garage[4], siendo presentada su primera versión en mayo de 2011. La financiación y el soporte de esta librería se presenta gracias a grandes empresas tales como, *Nvidia, Google, Toyota, Trimble, Urban Robotics, Honda Research Institute* y *Sandia Intelligent Systems and Robotics*. Por otra parte, PCL funciona sobre varias plataformas como Windows, Linux, MacOS, y Android. Estas herramientas ofrecen el potencial necesario para procesar y reconstruir una escena tridimensional de una manera sencilla.

Esta librería independiente incluye numerosos algoritmos sobre tratamiento de nube de puntos n-dimensionales y procesamiento geométrico en 3D. Las aplicaciones principales de esta librería consisten en percepción en robots, filtrado de datos obtenidos con sensores ruidosos, combinación de nubes de puntos en 3D, segmentación de partes relevantes de escenas, extracción de puntos de interés o cálculo de descriptores para crear reconocedores de objetos basados en su geometría. También incluye numerosas funciones para el mallado y reconstrucción de superficies, así como herramientas básicas de visualización. En este proyecto se ha empleado la versión 1.7 de esta librería, y se han utilizado los tipos de datos disponibles en PCL para la representación de datos como nube de puntos.

3.4. OpenCV

OpenCV[5] es una biblioteca libre de visión artificial originalmente desarrollada por Intel en enero de 1999. Al igual que PCL, en la actualidad está siendo mantenida por Willow Garage y su versión más actual es la 3.0, siendo ésta la empleada para la realización de este proyecto.

Se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente tanto para propósitos comerciales como de investigación.

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, calibración, visión estérea y visión robótica. Ha sido desarrollada principalmente en C y C++ primando su optimización,

[3] `http://pointclouds.org/`
[4] `http://www.willowgarage.com/`
[5] `http://opencv.org/`

aprovechando al máximo la capacidad de los procesadores multinúcleo. Otro de los puntos fuertes de esta librería es su documentación, tiene una amplia bibliografía y una gran comunidad de desarrolladores que la utiliza y actualiza.

OpenCV tiene una estrucutura modular, lo que significa que cada librería está clasificada en las siguientes categorías:

- **Core:** Aquí se definen los tipos de datos y las infraestructuras básicas de OpenCV en cuanto a estructuras de datos y funciones primarias.
- **Imgproc:** Todas las librerías ubicadas en esta categoría están relacionadas con el tratamiento de imágenes como las operaciones de filtrado, transformaciones geométricas, conversiones de sistemas de color, etc...
- **Vídeo:** Procedimientos relacionados con el análisis de vídeo como la estimación de movimiento, extracción del fondo y sistemas de seguimiento de objetos.
- **Calib3d:** Algoritmos relacionados con la calibración de cámaras, estimación de posición de objetos y cálculos geométricos básicos en vista múltiple.
- **Objdetect:** Detección de objetos y su reconocimiento y clasificación en clases predefinidas.
- **Highgui:** Proporciona un método sencillo para captura de vídeo, códecs para imagen y vídeo y procedimientos básicos apra desarrollo de interfaces.

En el presente proyecto se han utilizado los tipos de datos ofrecidos por esta librería para el manejo de las imágenes de profundidad y de celdas utilizadas en todo el sistema como el tipo de dato cv::Mat, y los algoritmos para realizar operaciones con matrices y el procesamiento de imágenes.

3.5. Biblioteca cvBlob

La librería cvBlob[6] fue diseñada y desarrollada en 2008 y está dedicada a detectar objetos de interés en una imagen en forma de *blobs* y gestionarlos de distintas maneras apoyándose en OpenCV. Entre las funciones que se pueden encontrar en esta librería está por ejemplo el mostrar por pantalla los *blobs* que se encuentran en cada fotograma o hacer un seguimiento de los *blobs* que aparecen por pantalla.

El algoritmo que detecta *blobs* en el procedimiento cvLabel incluido en esta librería está basado en el trabajo de Fu Chang [1]. Chun-Jen Lu en el que en una sola pasada de imagen pueden recoger tanto los *blobs* de una imagen como sus contornos. De hecho, detectando los contornos de una imagen el algoritmo va conformando los *blobs* en un algoritmo de cuatro pasos y haciendo una sola pasada:

[6]Web Librería cvBlob: `https://code.google.com/p/cvblob/wiki/HowToInstall`

1. Una vez obtenemos los contornos de una imagen, la función recorre la imagen binaria de arriba a abajo y de izquierda a derecha. Cuando encuentra un punto perteneciente a uno de los contornos sin etiqueta, le asigna una nueva y única a todos los puntos que lo conforman.

2. Si en los siguientes barridos nos encontramos con un punto B que forma parte de un contorno que ya ha sido etiquetado, todos los puntos sin etiqueta que nos encontremos dentro de él, como el punto C, se le asignará la misma que la del punto B.

3. Una vez etiquetado el punto C, se recorrerán todos los puntos que conforman este nuevo contorno para asignarles la misma etiqueta que la del punto C.

4. Cuando se encuentre un punto D ya etiquetado de un contorno contenido dentro de otro, el algoritmo recorrerá horizontalmente hacia la derecha la imagen y asiganará a los puntos en negro que se encuentre la misma etiqueta D hasta llegar al contorno exterior. De esta forma, aunque la figura conste de varios contornos internos, el algoritmo lo identificará como un solo componente o *blob*.

El algoritmo de detección de cvBlob ha servido de mucha ayuda en la etapa de segmentación para el desarrollo de un sistema de detección y conteo propio.

3.6. Librería Qt para la interfaz gráfica

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. En este caso, nostros hemos utilizado la librería Qt para desarrollar la interfaz gráfica de nuestro componente.

Qt ha sido desarrollada como un software libre y de código abierto a través de Qt Project, donde partcipa la comunidad de desarrolladores de Nokia, Digia y otras empresas. Qt es utilizada en KDE, entorno de escritorio para sistemas GNU/Linux o FreeBSD, entre otros. Qt utiliza el lenguaje de programación C++ de forma nativa, aunque puede ser utilizado en otros lenguajes de programación. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.

Funciona en todas las principales plataformas y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de funciones para el manejo de ficheros, además de estrucuturas adicionales.

A continuación un ejemplo de código Qt (ver figura 3.2):

```
/******** Hola.cpp **********/
```

```
#include <QtGui/QApplication>
#include <QtGui/QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel label("Hello World!");
    label.show();
    return app.exec();
}
```

Figura 3.2: Hola mundo en código Qt.

Los elementos que componen la ventana de un GUI se denominan *widgets*. Internamente, es posible implementar los efectos que tendrá cada uno de clicks en cada uno de los *widgets* gestionando los eventos (*signals*) que lanzan. Cada uno de los *widgets* hereda los métodos de un tipo de datos primigenio y después va especilizándose, dependiendo del tipo de *widget* que sea (botón, barra deslizante..), adquiriendo los métodos de cada tipo de datos del que procede. (Figura 3.3)

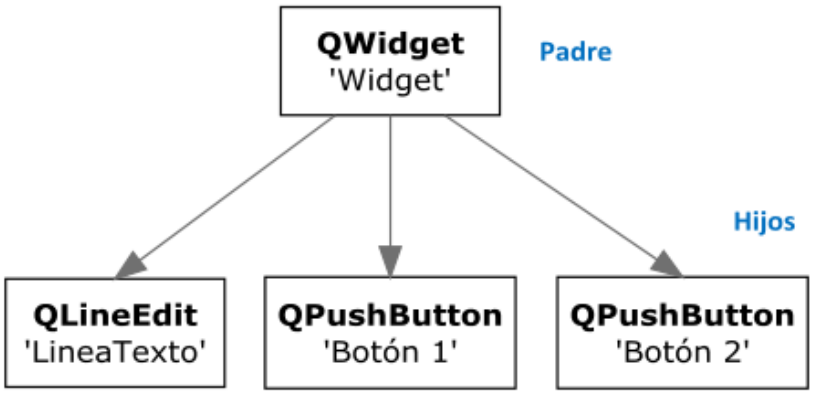


Figura 3.3: Jerarquía QWidget.

Los eventos que puede activar cada uno de los *widgets* se denominan *signals*. Cuando el usuario realiza alguna acción sobre un *widget* determinado tales como pulsar en el área que ocupa dicho elemento, arrastrar con el ratón el cursor, activar o desactivar un botón... se pueden recoger en el código y establer las acciones que se desse que se realice el componente.

De este modo podemos hacer que el software reaccione como se quiera, dependiendo de las acciones que realice el usuario a través de la interfaz construida y mostrar los resultados por pantalla a través del GUI, mediante el *widget* correspondiente, únicamente utilizando las herramientas que ofrece Qt, facilitando la tarea de desarrollo de esta parte del componente.

Para poder seleccionar entre las distintas opciones que el componente ofrece, se necesita implementar un medio con el que el usuario interactúe. Por ello, se ha desarrollado un GUI (*Graphical User Interface*) con el que poder seleccionar distintas opciones de visualización y modificar parámetros dentro de cada una en tiempo real. Para ello se ha utilizado la librería Qt compatible con C++, simplificando la tarea al poder usar el mismo lenguaje que utiliza el componente.

3.7. Herramienta Make

Es una unidad GNU que determina automáticamente qué ficheros de código fuente necesitan ser compilados y lanza los comandos necesarios para hacerlo, con ayuda de un fichero de texto llamado *makefile* donde se definen los parámetros de compilación que el usuario requiera. *Make* puede ejecutarse siempre y cuando el compilador del lenguaje con el que estemos trabajando pueda correr en *shell*. Fue implementada por Richard Stallman y Rolan McGrath.

En el fichero *makefile* se fijan todos los parámetros que se necesitan para compilar el código, incluido los enlaces a las librerías dinámicas y nombres de los ficheros resultantes de la compilación. *Make* también admite otras facilidades como poder interpretar nociones de bash en el fichero: uso de variables, ejecuciones de comandos en segundo plano... lo que lo convierte en una herramienta muy potente.

Gracias a toda la automatización que aporta la herramienta, simplifica y aclara mucho el proceso de compilación para el programador. Por este motivo hemos utilizado esta herramienta para compilar nuestro componente.

3.8. Subversion

Apache Subversion[7] (abreviado como SVN) es la herramienta de control de versiones que hemos utilizado para mantener los avances de nuestro componente. Esta herramienta está basada en un respositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.

Utiliza el concepto de revisión para guardar los cambios producidos en el respositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones, optimizando así al máximo el uso de espacio en disco. SVN permite al usuario crear,

[7] `https://subversion.apache.org/`

copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en un disco local. Dada esta flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones generadas.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras. A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada.

SVN posee el mismo sistema de ficheros básico para todos los proyectos. Una vez que se establece la ubicación del root del proyecto, se crea una estructura básica de directorios con tres carpetas:

- **Trunk:** Este es el directorio para la versión de desarrolo en curso y válida.
- **Tags:** Alberga las distintas versiones desarrolladas y cerradas del proyecto.
- **Branches:** Donde se ubican los desarrollos paralelos a lo que se está realizando en *trunk*.

Este tipo de programas se han hecho muy populares en el ámbito del desarrollo de software libre ya que permite compartir código *online* de una forma cómoda y organizada, controlando el número de versión cada vez que un miembro del equipo desarrolle algo nuevo y poniéndolo a disposición de quien posea permsisos para acceder a los ficheros publicados.

Capítulo 4

Desarrollo

En este capítulo se expondrá con detalle la solución propuesta al problema que se ha planteado anteriormente. Contar y seguir personas a partir de la información proporcionada por sensores RGBD y mediante técnicas de Visión Artificial.

En este capítulo se detallarán los módulos que componen el componente diseñado y desarrollado para ello. También se expondrá la mécanica de su funcionamiento: los parámetros que recibe cada uno de los bloques en que se estructura el componente y qué es lo que dibuja y muestra en la imagen de salida y en las de control. Finalmente se describirán los métodos que se han utilizado para las diferentes opciones que el componente posee para mostrar los resultados.

4.1. Diseño

El componente ha sido pensado para recoger datos de vídeo de un sensor RGBD y, fotograma a fotograma, procesar los datos del vídeo según haya sido la elección del usuario, devolviendo el resultado en el vídeo de salida integrado en la interfaz gráfica. Este modelo de caja negra del componente se detalla en la figura 4.1:

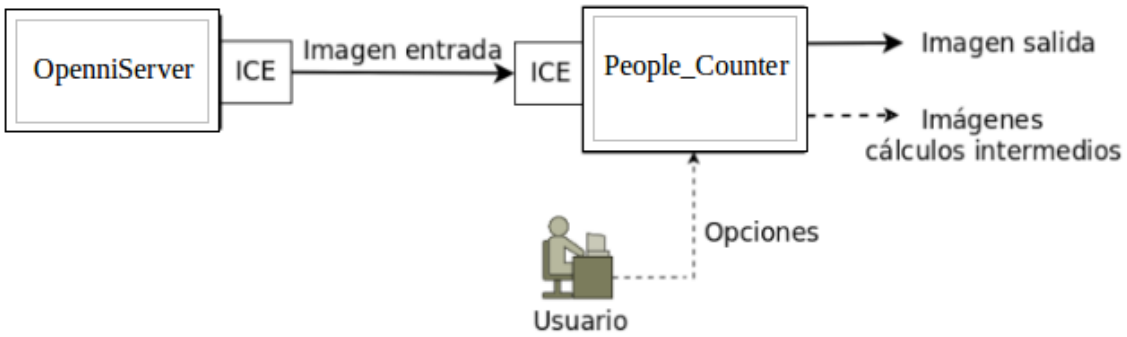


Figura 4.1: Diagrama de entradas y salidas del componente.

El componente se ha desarrollado modularmente y esto ha posibilitado que el usuario mediante la interfaz gráfica, pueda seleccionar y comprobar en el vídeo de salida y en el visor 3D cuál es el resultado de los procesamientos intermedios por los que pasa cada fotograma.

Se ha desarrollado un algoritmo que procesa la imagen en 4 etapas:

1. Etapa de captura de información en la que en cada instante *(t)* se recibe a desde un sensor RGBD la imagen de color y la imagen de profundidad actual. A partir de estas dos imágenes, componemos una nube de puntos para lograr una representación tridimensional de la escena.
2. Extracción de primer plano utilizando diferentes técnicas. En esta etapa nos quedamos con el primer plano de la imagen de distancias, ya que es ahí donde se encuentran las personas que aparecen en la escena.
3. Detección de los *blobs* de interés en la imagen de primer plano obtenida anteriormente. Se deben analizar los diferentes *blobs* del primer plano para identificar cuáles son personas y cuáles son falsos positivos.
4. Seguimiento de los *blobs* y actualización del número de personas observadas en la imagen.

Para la implementación y estudio del funcionamiento del algoritmo se ha programado un componente de JdeRobot escrito en C++. Se ha utilizado una estructura de dos hebras (ver figura 4.2) y para hacer posible la gestión de distintos hilos de ejecución se ha hecho uso de la librería *Qthread*. Ésta provee distintas utilidades para crear y gestionar varios hilos de ejecución. Las dos hebras que componen nuestro componente son:

- Una hebra para gestionar la interfaz gráfico de usuario (GUI) para mostrar de manera clara los resultados parciales o finales del control, y permite controlar el funcionamiento del componente.
- Otra hebra para administrar la API, cuyo cometido es ejecutar las funciones necesarias según la selección del usuario en la interfaz, pasando los resultados de estas funciones de vuelta al GUI para mostrarlo por pantalla.

Para el paso de datos entre la hebra GUI y la hebra API se utilizan variables compartidas y se controla el acceso a las mismas mediante variables de control *mutex* para evitar condiciones de carrera.

4.2. Extracción de primer plano

En cuestiones de tratamiento de imágenes, es esencial reducir cuanto se pueda el coste computacional para ejecutar un componente puesto que interesa presentar los

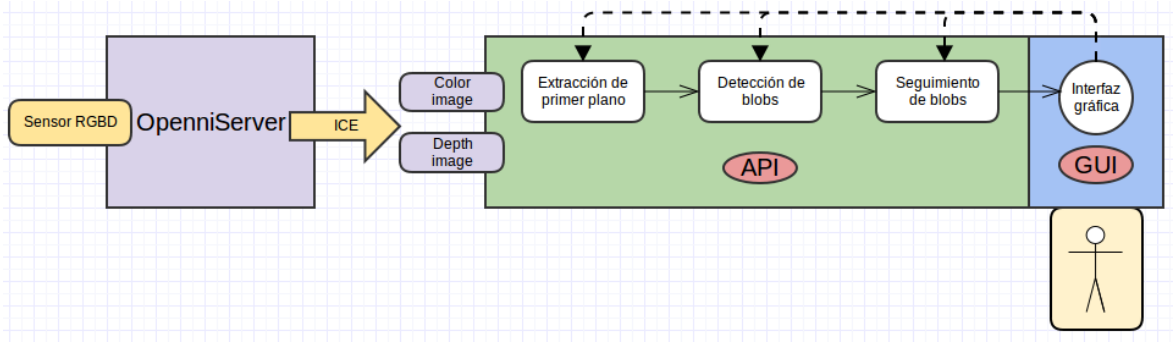


Figura 4.2: Diagrama de bloques de funcionamiento del componente.

resultados que da el algoritmo en tiempo real. También interesa facilitar la tarea a funciones como detectores de *blobs* o de puntos de interés para afinar y obtener mejores datos del proceso. Es por ello que se realiza antes de nada un filtrado del fondo de la imagen para descartar las zonas que no tienen interés, ya que en el primer plano se encuentran las personas que es lo que nos interesa identificar y contar.

Todas las técnicas de detección de primer plano se basan en modelar primeramente el fondo de la imagen. Es decir, definir el fondo y luego ver qué cambios se producen en éste. Definirlo puede ser una tarea difícil cuando contiene formas, sombras y objetos que se mueven. A la hora de definirlo, se asume que consiste en objetos estacionarios que pueden tener variaciones de color e intensidad en función del tiempo.

Los escenarios donde se aplican estas técnicas suelen ser muy diversos. Pueden ser secuencias muy variables, imágenes con una iluminación muy diferente, interiores, exteriores, más calidad o menos, etc. Se necesita un sistema que, aparte de poder procesar en tiempo real, sea capaz de adaptarse a todas estas variaciones.

Un buen sistema de detección de primer plano debe ser capaz de:

- Obtener el fondo (estimación) tanto si es variable como si es estático.
- Ser robusto ante cambios de iluminación, movimientos repetitivos en el fondo o cambios a largo plazo.

En nuestro componente se han incluido dos tipos de aprendizaje de fondo, lo cual sirve para comprobar su funcionamiento, y también se utilizarán para realizar la detección y seguimiento posterior de personas; la mezcla de guassianas y la incorporación de un fondo fijo.

4.2.1. Mezcla de Gaussianas

La librería OpenCV aporta varios procedimientos para el aprendizaje del fondo, entre los cuales se encuentra el aprendizaje mediante mezcla de gaussianas. Basado en el algoritmo desarrollado por Zoran Zivkovic [8] [9], esta función permite la detección

de objetos en primer plano de vídeos así como las extracción del fondo. También puede incluir en la detección las sombras pertenecientes a dichos objetos. En nuestro componente el aprendizaje de fondo se va alimentando continuamente con las nuevas imágenes que envía el sensor, actualizando en tiempo real su modelo de fondo y recalculando el primer plano en cada instante con esa nueva información.

Esta técnica aprovecha la idea de modelar el fondo con funciones gaussianas, pero en lugar de utilizar sólo una por cada píxel y variarla, considera que cada píxel puede tener diferentes estados en función del tiempo. El método de mezcla de guassianas proporciona para cada píxel una serie de gaussianas que corresponden a los estados posibles.

Para cada píxel:

- Diferentes gaussianas modelan diferentes estados de la escena.
- Cada gaussiana puede representar un modelo de fondo o de primer plano.

El algoritmo tiene diferentes pasos:

1. **Distribución de cada píxel (K gaussianas).** La distribución de cada píxel es una imagen viene definida por la expresión:

   $P(I_t) = \sum_{k=1}^{n} \omega_{i,t} \cdot \eta(I_t, \mu_{i,t}, \sum i, t)$

   Donde k es el número de guassianas utilizadas para modelar cada píxel (normalmente entre 3 y 5), $\omega_{i,t}$ es el peso de cada gaussiana y $\eta(I_t, \mu_{i,t}, \sum i, t)$ es la definición de la gaussiana con media ($\mu_{i,t}$) y matriz de covariancia ($\sum i, t$).

2. **Elección del modelo de fondo (B gaussianas).** El fondo para un píxel se modela con las B gaussianas con más peso y con menos varianza:

   $B = argmin_b(\sum_{k=1}^{b} \omega_{i,t} > T)$

   Donde T es un umbral de decisión (normalmente 0.6) y B el número mínimo de gaussianas para cada píxel. Para elegir las B funciones del fondo, las k gaussianas de cada píxel se ordenan por factor de peso $\omega_{i,t}$ y se eligen las B primas. El peso en una gaussiana representa las vecess que aparece, para cada píxel, ese estado. Cuanto más aparezca un estado, la gaussiana irá aumentando el peso.

3. **Decisión de fondo o primer plano.** Una vez tenemos todos los píxeles modelados podemos elegir si el píxel a analizar corresponde al fondo o no.

   - Un píxel será fondo si su valor corresponde con alguna de las B gaussianas.
   - Un píxel será de primer plano si su valor no corresponde con ninguna de las B gaussianas.

   Si un píxel no pertenece a ninguna de las K gaussianas del modelo se creará una nueva. Normalmente, un píxel así será considerado a primera vista primer plano, pero, si esta gaussiana nueva creada va aumentando su peso significa que

se convertirá en una de las gaussianas del fondo y este se habrá convertido en un objeto estático. El número de gaussianas del fondo siempre es constante. Si esta nueva gaussiana creada se conviertiera en fondo debido a un aumento de su peso, entonces otra gaussiana sería eliminada del modelo de fondo.

4. **Actualización del fondo.** Finalmente, se hará una actualización del fondo cuando uno de los píxeles corresponda con una de las K gaussianas del modelo probabilístico.

Este método tiene en cuenta los cambios de iluminación de la escena. A medida que se va reproduciendo el vídeo, el algoritmo de OpenCV va clasificando la información de imagen según los detecte como objetos de primer plano o como fondo.

En nuestro algoritmo aprendemos el fondo sobre la imagen de profundidad obtenida del sensor. A continuación detallamos el código utilizado en nuestro componente para extraer el primer plano por mezcla de Gaussianas utilizando para ello la función que proporciona OpenCV.

```
cv::Mat Control::getImageMOG(){
    mutex.lock();

    //Imagen depth del sensor RGBD
    cv::Mat result= image_d.clone();
    cv::vector<cv::Mat> rgb;
    split(result, rgb);

    //Umbralizamos la imagen de distancia
    threshold(rgb[0], result, 0, 255, CV_THRESH_BINARY|CV_THRESH_OTSU);

    //Extraccion de primer plano por mezcla de gaussianas
    pMOG= new cv::BackgroundSubtractorMOG2();
    blur(result, result, cv::Size(4,4));
    pMOG->operator()(result, result);

    //Erosionamos la imagen para eliminar el ruido
    cv::Mat kernel= cv::getStructuringElement(cv::MORPH_RECT,
                    cv::Size(7,7), cv::Point(3,3));
    morphologyEx(result, result, CV_MOP_CLOSE, kernel);
    threshold(result, result, 0, 255, CV_THRESH_BINARY|CV_THRESH_OTSU);

    cvtColor(result, result, CV_GRAY2RGB);

    mutex.unlock();
    return result;
    ~result;
    ~kernel;
}
```

(a) Imagen original (b) Imagen distancias (c) Primer plano

Figura 4.3: Función MOG de OpenCV.

Utilizamos la imagen de distancia que obtenemos del sensor RGBD, ya que utilizando esta imagen se reduce el error en el aprendizaje de fondo evitando confundir con el primer plano elementos como por ejemplo sombras.

En el experimento de la figura 4.3 hemos extraído el primer plano de una escena con varios elementos a parte de la persona que nos interesa identificar. Como el aprendizaje de fondo por mezcla de Gaussianas se va alimentando continuamente con las nuevas imágenes obtenidas del sensor, este algoritmo es capaz de etiquetar los objetos estáticos como fondo y de este modo no considerarlos primer plano, devolviendo una imagen más sencilla de tratar por el siguiente bloque del algoritmo (segmentación). Esta técnica proporciona resultados muy buenos en diferentes entornos debido a su carácter adaptativo.

4.2.2. Diferencia de imágenes sobre fondo fijo

Las imágenes con las que se va a trabajar se obtienen a través de una cámara fija. Por este motivo, los valores de los píxeles del fondo serán principalmente estáticos, o variarán muy poco. Con esta premisa se puede observar que restando los valores de los píxeles de cada fotograma y del fondo aprendido inicialmente, se pueden extraer los objetos móviles en primer plano, eliminado de este modo el fondo.

Para ello, es fundamental aprender al comienzo del algoritmo la imagen de fondo sin objetos de primer plano. Una vez que se obtiene el fondo aprendido, se puede calcular lo que se llama imagen de diferencias para obtener los objetos en primer plano. Para ello, se resta el valor de cada píxel de la imagen del fondo aprendido al de cada píxel correspondiente de la imagen de fondo. El resultado, si es menor que cierto umbral, se le asigna el valor 0. Si es mayor, se le asigna el valor diferencia. La figura 4.4 muestra un esquema del algoritmo utilizando este método:

Al obtener la imagen diferencias, se aplica una operación de filtrado de imagen *erode*, ubicado en la librería de OpenCV, para eliminar el ruido de la imagen y evitar de este modo "falsos positivos".

A continuación detallamos el código utilizado en nuestro componente para extraer el primer plano mediante diferencia de imágenes, apoyándonos para ello en funciones

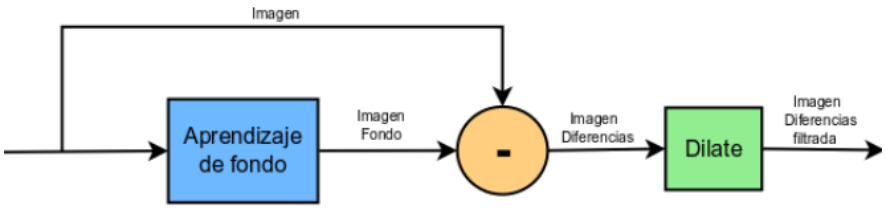


Figura 4.4: Extracción de primer plano por diferencias.

de la librería OpenCV.

```
cv::Mat Control::getImageDiff(){
    mutex.lock();
    cv::Mat first= first_image.clone();
    cv::Mat depth= image_d.clone();
    cv::Mat result(depth.size(), CV_8UC1);

    //Cogemos la imagen de distancia proporcionada por el sensor RGBD
    //y la imagen de fondo aprendido inicialmente
    cv::vector<cv::Mat> rgb_first;
    cv::vector<cv::Mat> rgb_depth;
    split(first, rgb_first);
    split(depth, rgb_depth);

    //Restamos ambas imagenes
    absdiff(rgb_first[0], rgb_depth[0], result);

    //Umbralizamos la imagen y erosionamos para eliminar el ruido
    threshold(result, result, 0, 255, CV_THRESH_BINARY|CV_THRESH_OTSU);
    cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE,
                    cv::Size(4,4));
    morphologyEx(result, result, cv::MORPH_CLOSE, kernel);
    cvtColor(result, result, CV_GRAY2RGB);

    mutex.unlock();
    return result;
    ~result;
    ~first;
    ~depth;
}
```

En el experimento de la figura 4.5 hemos aprendido la imagen de fondo al inicio de la prueba. Para ser capaces de extraer el primer plano de cada imagen obtenida del sensor RGBD, restamos píxel a píxel cada imagen con el fondo aprendido. De este modo, de

(a) Imagen original (b) Fondo aprendido (c) Primer plano

Figura 4.5: Extracción de primer plano por diferencias

una manera rápida podemos diferenciar el primer plano del fondo. Esta técnica no es adaptativa como la mezcla de gaussianas, ya que para obtener la diferencia utilizamos una imagen fija ya aprendida, sin embargo, cabe la posibilidad de volver a aprender el fondo si este hubiese cambiado demasiado y nos interesase hacerlo.

Sin embargo, debido al ruido producido por el sensor en los bordes de los objetos, la diferencia de imágenes sobre fondo fijo devuelve a veces un primer plano algo distorsionado por este ruido. Para solucionar este fenómeno, erosionamos la imagen para eliminar estos píxeles que habían sido marcados pcomo primer plano por error. Por tanto, con esta opción obtenemos unos resultados muy parecidos a los logrados con el método anterior.

4.2.3. Otras técnicas de extracción de primer plano

La detección de primer plano es un problema muy estudiado desde hace tiempo en el campo de la visión por computador. Por este motivo, a parte de las técnicas detalladas anteriormente y utilizadas en nuestro componente, existen otras técnicas basadas en la dualidad de fondo estacionario y dinámico a la vez.

- **Filtro de mediana temporal.** Este sistema estima el modelo de fondo a partir de la mediana de todos los píxeles de un cierto número de imágenes anteriores.

  Para modelar el fondo, el sistema estudia todas las imágenes de un periodo de tiempo determinado llamado tiempo de entrenamiento. En este tiempo sólo se visualizarán imágenes del fondo y se calculará la mediana, píxel por píxel, de todas las tramas de fondo de este tiempo.

  Después del periodo de entrenamiento, para cada trama nueva, cada valor de píxel de entrada se compara con su valor de fondo calculado previamente. Si el píxel de entrada está dentro de un umbral establecido, se considera que el píxel coincide con el fondo modelo y su valor se incluye en la memoria intermedia de píxeles. En caso contrario, si el valor de píxel está fuera de este umbral, se clasifica como de primer plano, y no se incluirá en la memoria intermedia.

No se puede considerar este método muy eficiente ya que no presenta una base estadística rigurosa y precisa de una memoria intermedia que tiene un coste computacionalmente elevado.

- **Promedio de Gaussianas.** La idea principal de este método es modelar el fondo de una manera más probabilística basada en una única distribución gaussiana. Cada píxel cuenta con su distribución de probabilidad gaussiana caracterizada por una media y una varianza. Como para cada trama el fondo puede ir cambiando, la media y la varianza se tienen que ir actualizando.

  Aparte de actualizar en cada píxel la distribución gaussiana, se clasificará el píxel a analizar: fondo o primer plano. Para ello, tenemos que comprobar si el píxel analizado pertenece a la gaussiana definida para el modelo de fondo de aquel valor de píxel.

  Este método, además de estimar un modelo de fondo más parecido a la realidad, precisa de menos memoria que el anterior ya que sólo tienen que ir variando la media y la varianza.

- **Eigenbackgrounds.** La idea de este método es construir un fondo mediante los autovectores de éste, es decir, crearemos una imagen de fondo modélica. Dado que una imagen contiene su información de una manera matricial, en este método se utilizan los autovectores y los autovalores que se van obteniendo. Este entorno es conocido como un autoespacio. Se obtiene, imagen a imagen, aquella matriz donde se delimita un autoespacio. Cabe señalar que la mayor parte del peso de una imagen está contenida en pocos autovalores, de modo que los de menor peso serán desechables.

  Este método, al contrario que los métodos anteriores que se basan en datos estadísticos del píxel, se basa en datos estadísticos de la escala de grises sobre el tiempo. Es una técnica más sencilla y rápida pero pierde precisión y es dificultoso trabajar con datos en tiempo real dado el alto coste computacional del algoritmo.

4.3. Segmentación de imágenes

La segmentación en la visión artificial es el proceso de dividir una imagen digital en varias partes u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. La segmentación se utiliza tanto para localizar objetos y/o personas como para encontrar los límites de éstos dentro de una imagen. Más precisamente, la segmentación de la imagen es el proceso de asignación de una etiqueta a cada píxel de la imagen de forma que los píxeles que compartan la misma etiqueta forman parte de un mismo segmento.

El resultado de la segmentación de una imagen es un conjunto de segmentos que cubren en conjunto a toda la imagen. Cada uno de los píxeles de un segmento son similares en alguna característica, como el color, la intensidad o la textura. Regiones adyacentes son significativamente diferentes con respecto a las mismas características.

Los algoritmos de segmentación se basan en los siguientes principios:

- **Discontinuidades del nivel de gris.** Consisten en segmentar la imagen a partir de los cambios grandes en los niveles de gris entre los píxeles. Las técnicas que utilizan las discontinuidades como base son la detección de líneas, de bordes y de puntos aislados.
- **Similitud de niveles de gris.** Es lo contrario al método anterior, las divisiones de la imagen se hacen agrupando los píxeles que tienen unas características similares. Algunas técnicas que usan esto son la umbralización y el crecimiento de regiones.

En nuestro componente se han incluido dos tipos de segmentación de objetos, lo cual sirve para comprobar su funcionamiento y también para realizar la detección y posterior seguimiento de personas.

4.3.1. Segmentación mediante librería cvBlob

Una vez conseguidos los resultados por la extracción del primer plano, se pasan a la función cvLabel de la librería cvBlob para la detección de *blobs*. Esta función se basa en un sistema de detección y seguimiento de contornos del objeto que se detecta en la imagen e identifica y etiqueta el área interna de cada componente detectado en forma de *blobs*. La descripción de la cabecera de la función corresponde a la siguiente:

```
unsigned int cvLabel(IplImage const *img, IplImage *imgOut, cvBlobs &blobs);
```

- **img** es la imagen de entrada que se pasará al procedimiento.
- **imgOut** será la imagen que devuelve la función
- **blobs** es la variable donde se devuelven todos los *blobs* encontrados por el procedimiento. Esta variable está basada en el tipo *std::map* y cada elemento de esta estructura es donde se encuentran los datos de cada *blob*.
- La función devuelve un entero correspondiente al número de píxeles que ha etiquetado en *blobs*.

Una vez se obtiene la variable *blobs*, si no está vacía, se pueden reprocesar áreas de interés en la imagen a partir de estos datos o dibujar en la imagen lo devuelto por la función cvLabel.

La función cvLabel trabaja con tipos de datos IplImage, y nuestro algoritmo trabaja con cv::Mat, por este motivo, es necesario una conversión de datos. A continuación, detallamos el código de nuestro componente.

```
cv::Mat Control::getImageBlob(){
    mutex.lock();
    // obtenemos la imagen de distanciad el sensor RGBD
    cv::Mat depth_gray= image_d.clone();
    cv::Mat sub_depth_gray(depth_gray, cv::Rect(10, 5, depth_gray.cols-30,
           depth_gray.rows-20));
    cv::vector<cv::Mat> rgb;
    split(sub_depth_gray, rgb);
    cvtColor(rgb[0], sub_depth_gray, CV_GRAY2RGB);

    // Convertimos de cv::Mat a IplImage
    IplImage img = sub_depth_gray;

    // Convertimos a escala de grises
    IplImage *gray = cvCreateImage(cvGetSize(&img), IPL_DEPTH_8U, 1);
    cvCvtColor( &img, gray, CV_BGR2GRAY );

    // Obtenemos una imagen binaria
    cvThreshold( gray, gray, 0, 255, CV_THRESH_BINARY_INV|CV_THRESH_OTSU);

    // Obtenemos los blobs
    IplImage *labelImg = cvCreateImage(cvGetSize(gray), IPL_DEPTH_LABEL, 1);
    CvBlobs blobs;
    unsigned int result = cvLabel(gray, labelImg, blobs);

    // Representamos los blobs en la imagen original
    cvRenderBlobs(labelImg, blobs, &img, &img);
    nblobs = blobs.size();

    // Convertimos de nuevo a cv::Mat
    cv::Mat out(&img);
    mutex.unlock();

    return out;
    ~depth_gray;
    ~sub_depth_gray;
    ~out;
    cvReleaseImage(&labelImg);
    cvReleaseImage(&gray);
}
```

En el experimento mostrado en la figura 4.6 hemos utilizado la función cvLabel de la biblioteca cvBlobs para segmentar los *blobs* de la imagen. Esta función devuelve resultados muy satisfactorios en diferentes entornos y además de segmentar los objetos de primer plano identificados, los señala en la imagen de salida marcando su centro.

Sin embargo, este método depende bastante de la extracción de primer plano realizada previamente, ya que segmenta todos los objetos. Por tanto, si la imagen

(a) Imagen RGB (b) Imagen Depth (c) Segmentación cvLabel

Figura 4.6: Segmentación mediante cvLabel

de primer plano que utiliza tiene ruido, la segmentación devuelve falsos positivos, sobresegmentando la imagen. Por eso, en entornos con fondos poco uniformes, es fundamental lograr una buena extracción de primer plano para evitar errores en esta segunda fase. Aún así, utilizando cvBlob para la segmentación de personas en nuestro componente hemos obtenido muy buenos resultados.

4.3.2. Segmentación mediante Watershed

La técnica *watershed* es una técnica bien conocida en el área de la topografía. La técnica *watershed* considera la imagen como un gráfico 3D, en el que se representa la altitud (nivel de gris) frente a las coordenadas espaciales de dicho punto.

Los mínimos locales de dicho gráfico representarían cuencas donde el agua se iría acumulando, simulando así el comportamiento del terreno en la realidad. Llegado el punto en el que si seguimos llenando una cuenca ésta se desbordaría, se construye un dique para evitar la fusión de dos o más de estas cuencas. Estos diques serían los contornos de las imágenes a reconocer (ver figura 4.7). La idea básica consiste en considerar que los contornos de los objetos se corresponden con las línea donde el nivel de gris varía más rápidamente.

Sin embargo, la técnica de *watershed* suele llevar a una sobresegmentación. Para evitar este fenómeno indeseado, existen dos posibilidades.

- Eliminar los contornos irrelevantes una vez realizado el proceso de *watershed*.
- Modificar la imagen gradiente de tal forma que las regiones de depresión o valles se correspondan únicamente con los objetos que se desea segmentar.

En nuestro componente hemos implementado la segunda opción, ya que pasamos a nuestro algoritmo de segmentación por *watershed* una imagen binaria donde se distingue fondo de primer plano. A continuación, mostramos el código utilizado en nuestro componente para segmentar la imagen obtenida del sensor mediante *watershed*. Para ello, utilizamos funciones de la librería OpenCV:

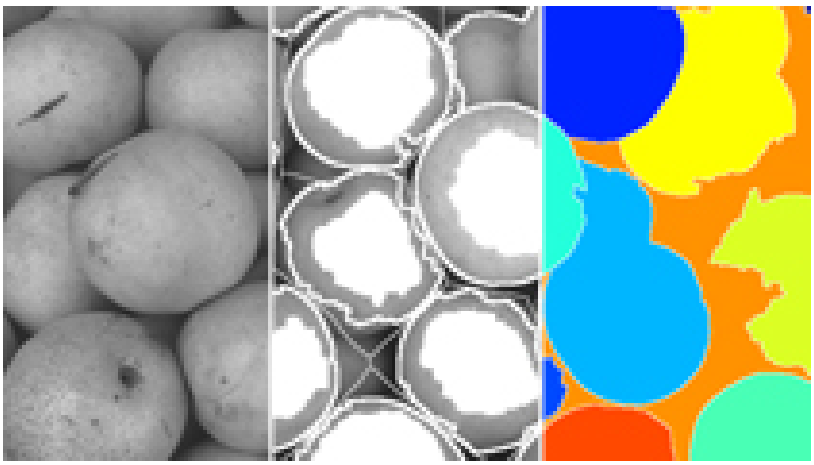


Figura 4.7: Segmentación por watershed.

```
cv::Mat Control::getImageSeg2(){
    mutex.lock();
    //Obtenemos la imagen de distancia del sensor RGBD
    cv::Mat depth_gray= image_d.clone();
    cv::Mat sub_depth_gray(depth_gray, cv::Rect(2, 5, depth_gray.cols-30,
                    depth_gray.rows-10));
    cv::vector<cv::Mat> rgb;
    split(sub_depth_gray, rgb);
    cvtColor(rgb[0], sub_depth_gray, CV_GRAY2RGB);

    // Obtenemos la imagen binaria
    threshold(rgb[0], sub_depth_gray, 0, 255,
              CV_THRESH_BINARY_INV|CV_THRESH_OTSU);

    // DistanceTransform
    cv::Mat dist;
    distanceTransform(sub_depth_gray, dist, CV_DIST_L2, 3);
    normalize(dist, dist, 0, 1., cv::NORM_MINMAX);
    threshold(dist, dist, .5, 1., CV_THRESH_BINARY);
    cv::Mat dist_8u;
    dist.convertTo(dist_8u, CV_8U);

    // Encontramos las semillas
    std::vector<std::vector<cv::Point> > contours;
    findContours(dist_8u, contours, CV_RETR_EXTERNAL,
              CV_CHAIN_APPROX_SIMPLE);

    // Identificamos y contamos los objetos de la imagen
    ncomp = contours.size();
    cv::Mat markers = cv::Mat::zeros(dist.size(), CV_32SC1);
    for (int i = 0; i < ncomp; i++)
        drawContours(markers, contours, i, cv::Scalar::all(i+1), -1);
```

```
    circle(markers, cv::Point(5,5), 3, CV_RGB(255,255,255), -1);
    cvtColor(rgb[0], sub_depth_gray, CV_GRAY2RGB);
    watershed(sub_depth_gray, markers);

    // Generamos color aleatorios para las semillas
    std::vector<cv::Vec3b> colors;
    for (int i = 0; i < ncomp; i++){
        int b = cv::theRNG().uniform(0, 255);
        int g = cv::theRNG().uniform(0, 255);
        int r = cv::theRNG().uniform(0, 255);
        colors.push_back(cv::Vec3b((uchar)b, (uchar)g, (uchar)r));
    }

    // Representamos las semillas en la imagen original
    cv::Mat result = cv::Mat::zeros(markers.size(), CV_8UC3);
    for (int i = 0; i < markers.rows; i++){
        for (int j = 0; j < markers.cols; j++){
            int index = markers.at<int>(i,j);
            if (index > 0 && index <= ncomp)
                result.at<cv::Vec3b>(i,j) = colors[index-1];
            else
                result.at<cv::Vec3b>(i,j) = cv::Vec3b(0,0,0);
        }
    }

    mutex.unlock();
    return result;
    ~result;
    ~dist;
    ~dist_8u;;
    ~depth_gray;
    ~sub_depth_gray;
    ~markers;
}
```

(a) Imagen original (b) Imagen distancias (c) Imagen segmentada

Figura 4.8: Segmentación mediante *watershed*.

En el experimento de la figura 4.8 hemos utilizado la técnica de *watershed* para la segmentación de la imagen. Para ello, utilizamos la imagen binaria de primer plano que previamente hemos obtenido en el paso anterior del algoritmo. Este método de segmentación es mucho más sensible a las variaciones de nivel de gris que la función cvLabel de cvBlob, por este motivo en ocasiones se produce el fenómeno de la sobresegmentación obteniendo resultados erróneos. Este fenómeno se produce sobre todo cuando tenemos varios blobs muy próximos entre sí. Sin embargo, es un método con un menor coste computacional y que proporciona resultados muy buenos en la mayoría de los entornos, pudiendo solucionar la mayoría de los casos de sobresegmentación con técnicas sencillas y rápidas como la erosión y dilatación.

4.3.3. Otras técnicas de segmentación de objetos

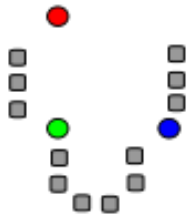
Aunque para componente hemos utilizado las técnicas explicadas anteriormente, existen mutlitud de técnicas para la segmentación de objetos. A continuación expondremos algunas de las técnicas más utilizadas.

- **Algoritmo K-medias.** Es una técnica iterativa que se utiliza para dividir una imagen en K segmentos. El algoritmo básico es (ver figura 4.9):

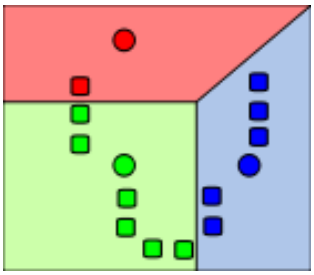
  1. Escoger K centros de segmentos, ya sea de forma aleatoria o basándose en algún método heurístico.
  2. Asignar a cada píxel de la imagen el segmento que minimiza la varianza entre el pixel y el centro del cluster.
  3. Recalcular los centros de los segmentos haciendo la media de todos los píxeles del cluster.
  4. Repetir los pasos 2 y 3 hasta que se consiga la convergencia.

  En este caso la varianza es la diferencia absoluta entre un píxel y el centro del cluster. La diferencia se basa típicamente en el color, la intensidad, la textura y la localización del píxel, o una combinación ponderada de todos estos factores. Este algorítmo garantiza la convergencia, pero puede devolver una solución que no sea óptima. La calidad de la solución depende de la serie inicial de segmentos y del valor de k.

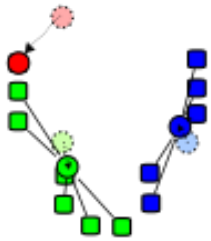
- **Métodos basados en el histograma.** Los métodos basados en el histograma son muy eficientes en comparación con otros métodos de segmentación de la imagen, ya que normalmente requieren sólo una pasada por los píxeles. En esta técnica, un histograma se calcula a partir de todos los píxeles de la imagen, y los picos y valles en el histograma se utilizan para localizar los grupos en la imagen, se puede utilizar el color o la intensidad como medida (ver imagen 4.10. Una desventaja del método de búsqueda de histograma es que puede ser difícil de identificar los picos y valles importantes en la imagen.



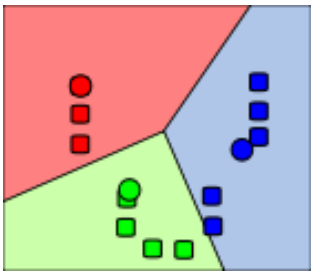
(a) Se seleccionan k puntos de la imagen, 3 en este caso.



(b) Se crean k clusters a partir de los puntos anteriores.



(c) El centro de gravedad de cada grupo k se convierte en el nuevo medio.



(d) Los pasos b y c se repiten hasta que se alcance la convergencia

Figura 4.9: Algoritmo K-medias

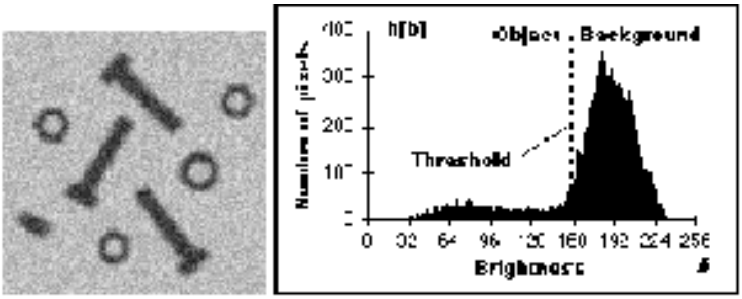


Figura 4.10: Segmentación por histograma.

- **Método de detección de bordes.** Los límites de regiones y los bordes están estrechamente relacionados, ya que a menudo hay un fuerte ajuste en la intensidad en los límites de las regiones. Las técnicas de detección de bordes pueden ser usadas como otra técnica de segmentación más. Los bordes identificados por la detección de bordes en ocasiones están desconectados. Para segmentar un objeto a partir de una imagen sin embargo, es necesario que los bordes formen figuras cerradas (ver figura 4.11.

Figura 4.11: Detección de bordes.

- **Método de crecimiento de semillas.** Este método toma un conjunto de semillas como entrada junto con la imagen. Las semillas marcan cada uno de los objetos que tienen que ser segmentados. Las regiones crecen iterativamente mediante la comparación de todos los píxeles vecinos no asignados a ninguna región. La diferencia entre el valor de la intensidad de un píxel y el de la media de la región se utiliza como una medida de similitud. Cada píxel se asigna a la región con la que su diferencia con la media es menor, de esta forma todos los píxeles se asignan a sus respectivas regiones.

  Este proceso de crecimiento de regiones por semillas requiere semillas como entrada adicional. Los resultados de la segmentación dependen de la elección de las semillas. El ruido en la imagen puede hacer que las semillas queden mal colocadas.

- **Método basado en modelos.** La hipótesis central de este enfoque es que las estructuras de interés tienen una forma geométrica repetitiva. Por lo tanto, se puede buscar un modelo probabilístico para explicar la variación de la forma de la estructura y luego cuando se segmenta una imagen se imponen limitaciones para tomar la imagen como el modelo elegido a priori. Esta tarea implica:

  - La selección de los ejemplos de entrenamiento (ejemplos que se usan para probar los modelos).
  - La representación probabilística de la variación de los ejemplos seleccionados.
  - La inferencia estadística entre el modelo y la imagen.

4.4. Seguimiento y cuenta de objetos

Una vez que tenemos el resultado final de la segmentación de primer plano de un fotograma, se pueden hallar los objetos de interés en pantalla, y realizar un seguimiento de los mismos, identificando cada *blob* en el transcurso de la ejecución de las diferentes iteraciones del algoritmo con el objeto con el que se corresponde y que aparece en pantalla. El resultado de dicho proceso de seguimiento se irá almacenando en una estructura de datos denominada *track*.

Para lograr este objetivo, hemos utilizado el algoritmo de seguimiento que proporciona la librería cvBlob. Este algoritmo trabaja con los *blobs* que se han detectado previamente. El planteamiento de este algoritmo es realizar el seguimiento relacionando un mismo *blob* en distintos fotogramas, utilizando nada más que la distancia recorrida entre un fotograma y otro. Este procedimiento trabaja con una matriz de distancias entre los *blobs* detectados en el fotograma actual, y los *tracks*, la cual se denomina matriz de adyacencias. Esta matriz se utiliza para cribar los resultados obtenidos optimizando el consumo de recursos del sistema, pudiendo así distinguir entre elementos de interés reconocidos de anteriores fotogramas y nuevos elementos detectados que aparecen por primera vez.

La función de la librería cvBlob tiene el siguiente aspecto:

```
void cvUpdateTracks(cvBlobs const &b, cvTracks &t,
                    const doube thDistance,
                    const unsigned int thInactive,
                    const unsigned int thActive=0);
```

- **b -** Variable donde se encuentran los *blobs* detectados
- **t -** Variable donde se encuentran los *tracks*
- **thDistance -** Máxima ditancia de separación entre un blob y un track para ser considerados en el mismo elemento. Si este valor se rebasa, se considera que ese *blob* y ese *track* no son el mismo.
- **thInactive -** Número máximo de fotogramas en los que un *track* puede quedarse inactivo.
- **thActive -** Si un *track* pasa a inactivo, pero ha estado activo menos cantidad de fotogramas que el indicado en este argumento, se elimina.

El algoritmo hace uso de acumuladores para cada *blob* y cada *track*, relacionados con sus identificadores. Cada acumulador se incrementa cada vez que se detecta que una distancia entre un *blob* instantáneo y un *track* en la matriz es menor que la distancia umbral establecida en los parámetros de la función. De esta forma, cada acumulador sirve como un indicador que muestra cuántas veces se ha cumplido la condición de la distancia para un *blob* o *track* concreto. De este modo, el conteo se realiza sobre los

tracks, que son una estimación más fiable que los *blobs* instantáneos y más fiable y robusta al ruido de los sensores.

En la figura 4.12 se puede observar la estructura de la matriz de adyacencias. Tras las líneas, se sitúan los acumuladores para cada *track* y *blob* detectado en el fotograma actual. En la parte superior y en el extremo izquierdo en negrita, pueden verse los distintos identificadores para cada *blob* y *track* activos en ese momento.

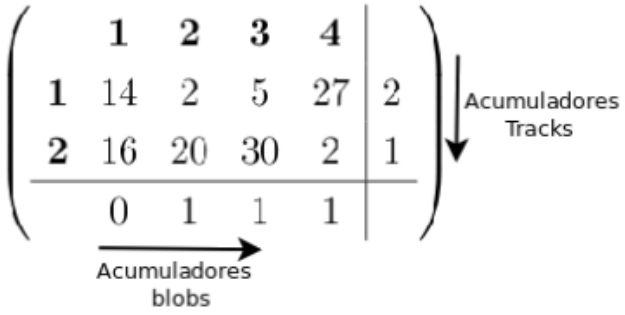


Figura 4.12: Representación de matriz de adyacencias utilizada por cvBlob.

Una vez que tenemos construida dicha matriz, ésta se analiza elemento a elemento y se crea una lista de *blobs* y otra de *tracks*. Dependiendo de si su acumulador es mayor que 0 o no, el *blob* o el *track* pasa a formar parte de la lista correspondiente de posibles candidatos aptos para ser emparejados. El algoritmo busca entonces en la lista de *tracks* el elemento con mayor área y lo empareja con el *blob* de mayor área que encuentra en la otra lista.

4.5. Interfaz gráfico

Para que el usuario pueda interaccionar con la aplicación y ver los resultados en tiempo real es preciso diseñar y construir una interfaz gráfica fácil de manejar y que ayude a abstraer al usuario lo máximo posible de los pormenores del funcionamiento interno del programa. Para ello se ha implementado una interfaz usando Qt para el componente. La figura 4.13 muestra la interfaz gráfica de nuestro componente.

Como se puede ver en la figura 4.13 nuestro componente está dividido en tres zonas diferenciadas: imágenes 2D, panel de opciones y visor 3D. Para mostrar las imágenes en la interfaz gráfica hemos utilizado *canvas* de Qt. Las imágenes de tipo cv::Mat se muestran en el GUI convirtiéndolas a QImage y colocándolas en QLabel como muestra el siguiente código:

```
imageColor= this->control->getImageRGB();
QImage imageQt1= QImage((uchar*) imageColor.data, imageColor.cols,
              imageColor.rows, imageColor.step, QImage::Format_RGB888);
img1label->setPixmap(QPixmap::fromImage(imageQt1));
```

Por otro lado, también se ha incluido dentro del código del GUI un visor 3D para

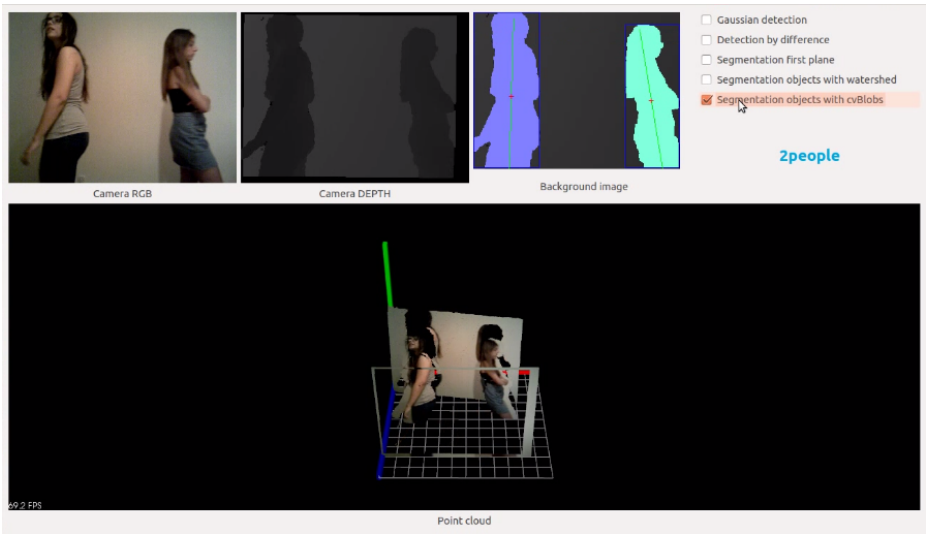
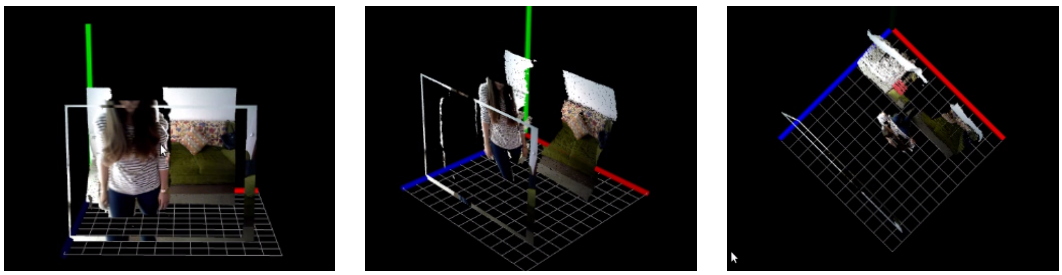


Figura 4.13: Interfaz gráfica de People Counter.

visualizar en tres dimensiones la escena captada por en el sensor RGBD. Esta nube de puntos se construye apoyándonos en la librería PCL[1], utilizando para ello los valores RGB obtenidos de la imagen de color del sensor y la posición en el espacio calculada a partir de los valores de la imagen de distancia del sensor. Para mostrar el visor 3D hemos utilizado qvtkWidget que es capaz de contener nubes de puntos. Este tipo de *widget* permite además de mostrar ejes y rejilla, la posibilidad de ver la nube de puntos desde cualquier ángulo, ya que es dinámico y el usuario puede cambiar la perspectiva con el ratón (ver figura 4.14).



(a) (b) (c)

Figura 4.14: Visor 3D

La interfaz da la posibilidad de mostrar al usuario cómo funcionan los cálculos intermedios utilizados para calcular los *blobs* de la imagen, tales como la extracción de primer plano, y el uso de varios métodos de realizarlo si procede. Estas diferentes opciones pueden seleccionarse a través de lo botones de la interfaz gráfica. Para estos botones hemos utilizado el *widget* de Qt QCheckBox.

En la imagen de salida que se muestra en la parte derecha de la interfaz, puede

[1]PCL: `http://pointclouds.org/`

visualizarse desde imágenes que el algoritmo utiliza para cálculos intermedios hasta la imagen final con los *blobs* encontrados. El usuario puede seleccionar qué desea ver utilizando las opciones que se detallan a continuación:

- La opción *"Gaussian detection"* muestra una imagen binaria donde se diferencia el fondo del primer plano utilizando para ello la técnica mezcla de gaussianas explicada anteriormente.
- La opción *"Detection by difference"* muestra de nuevo una imagen binaria que distingue fondo de primer plano, pero en este caso, la técnica utilizada para calcular esta imagen es la diferencia de imágenes. Para conseguir está extracción de primer plano, se aprende inicialmente el fondo y se resta cada fotograma con el fondo aprendido.
- La opción *"Segmentation objects with watershed"* muestra la imagen segmentada utilizando la técnica de *watershed* explicada anteriormente. Esta opción identifica los diferentes blobs que hay en la imagen y los cuenta, mostrando este valor en el panel que se encuentra a la derecha de la imagen.
- La opción *"Segmentation objects with cvBlob"* muestra en la imagen de salida una segmentación de los objetos de primer plano utilizando para ello la librería cvBlob. Gracias a las funciones que esta librería ofrece podemos identificar y contar los blobs existentes en la imagen, mostrando este valor en el panel que se encuentra a la derecha de la imagen de salida.

4.6. Experimentos globales

En este último punto se explicarán las diferentes pruebas que se han hecho al programa desarrollado y se recogerá su resultado. Realizando estos experimentos, se busca que el programa reconozca las personas de la manera más exacta posible y las contabilice.

Este programa ha sido analizado y probado en un Toshiba Satellite L750-17M con un procesador de segunda generación de doble núcleo Intel Core a 2.20 Ghz, 2 GB de memoria RAM, una tarjeta gráfica integrada Intel HD Graphics 3000 y Ubuntu 14.04. También se ha utilizado el sensor RGBD Xtion PRO LIVE.

Las secuencias de vídeo fueron tomadas desde diferentes ángulos en interiores, para conseguir así distintas configuraciones y ejemplos, tanto ideales como no ideales.

A continuación detallamos diferentes experimentos que ilustran la ejecución típica del algoritmo y son representativos de su funcionamiento normal.

4.6.1. Experimento 1

En el siguiente experimento hemos probado nuestro algoritmo completo en un escenario real, utilizando las diferentes opciones que implemenadas para cada bloque

del algoritmo con idea de comparar los resultados obtenidos y determinar qué técnica es más recomendable.

En la figura 4.15 hemos utilizado la mezcla de gaussianas para la extracción de primer plano. Está técnica es bastante fluida ya que que analiza 19.85 fotogramas por segundo y además se obtienen muy buenos resultados. Cabe destacar, que para este bloque utilizamos la imagen de distancias obtenida por el sensor RGBD, ya que de este modo, la extracción de primer plano es más precisa que con imagen RGB porque eliminamos elementos que podrían dificultar este paso como por ejemplo las sombras. Como ya hemos dicho anteriormente, la fase de extracción del primer plano nos devuelve una imagen binaria donde el fondo tiene valor 0 y el primer plano 1. Con esta imagen, nuestro algoritmo de segmentación basado en cvLabel es capaz de detectar e identificar las diferentes personas que aparecen en la escena. De este modo, el bloque de seguimiento y conteo logra distinguir y contar a las dos personas que aparecen en la imagen.

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen Segmentada

Figura 4.15: Segmentación mediante cvLabel y mezcla de gaussianas

En la figura 4.16 hemos probado la segmentación mediante cvLabel utilizando previamente la diferencia de imágenes sobre fondo fijo para la extracción de primer plano. Esta técnica es más fluida que la mezcla de gaussianas, ya que analiza 20.13 imágenes por segundo. Por tanto, en escenarios como el de este experimento con fondos bastante estáticos, es muy recomendable utilizar está opción ya que ganamos algo de fluidez y como el fondo no cambia demasiado no necesitamos estimarlo continuamente. Una vez extraído el primer plano, la segmentación de la imagen con cvLabel devuelve un resultado similar al anterior.

Por otro lado, en la figura 4.16 y en la figura 4.18 hemos realizado las mismas pruebas que en el caso anterior (extracción de primer plano mediante mezcla de gaussianas y mediante diferencia de imágenes con fondo fijo), pero utilizando en este caso para la segmentación la técnica de *watershed*. Los resultados para ambos métodos de segmentación son muy similares y satisfactorios, además estas dos técnicas funcionan de manera muy fluida, ya que cvLabel analiza 19.55 fotogramas por segundo y *watershed* 19.73 fotogramas por segundo. Sin embargo, cvLabel aporta más información, ya que identifica el *blob* en la imagen de salida y señala su centro y además, es más robusto que *watershed*, ya que éste último es más sensible a niveles de gris y funciona peor cuando la persona está demasiado cerca del sensor.

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen Segmentada

Figura 4.16: Segmentación mediante cvLabel y diferencia de imágenes

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen segmentada

Figura 4.17: Segmentación mediante *watershed* y mezcla de gaussianas

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen segmentada

Figura 4.18: Segmentación mediante *watershed* y diferencia de imágenes

4.6.2. Experimento 2

En este segundo experimento hemos probado nuestro componente en un entorno con un fondo más hetereogéneo, para comparar los resultados de las diferente técnicas de extracción de primer plano y segmentación.

En la figura 4.19 podemos ver el funcionamiento de la extracción de primer plano utilizando mezcla de gaussianas. El resultado es muy bueno, ya que esta función estima el fondo continuamente, obviando los objetos que pertenecen a dicho fondo. Con lo cual, está técnica sigue devolviendo resultados óptimos en entornos más complejos.

(a) Imagen RGB (b) Imagen de profundidad (c) Primer plano

Figura 4.19: Extracción de primer plano por mezcla de gaussianas

En la figura 4.20 hemos utilizado la técnica de diferencia de imágenes con fondo fijo para extraer el primer plano. La imagen de profunidad que proporciona el sensor varía bastante en los bordes de los objetos entre los fotogramas *(t)* y *(t+1)*, por este motivo, como se puede ver en la imagen, esta técnica no devuelve un primer plano exacto, ya que tiene bastante ruido. Para solventar este problema hemos erosionado la imagen y de este modo eliminamos el ruido que aparece marcado como primer plano erróneamente, como se puede ver en la figura 4.21.

(a) Imagen RGB (b) Imagen de profundidad (c) Primer plano

Figura 4.20: Extracción de primer plano por diferencia de imágenes sin erosionar

Por otro lado, en lo referente al bloque de segmentación de objetos, en la figura 4.22 y 4.23 mostramos errores típicos que encontramos con la técnica de *watershed*. Como hemos dicho anteriormente, esta opción es más sensible a los niveles de grises y por este motivo nos encontramos sobresegmentación. Para solucionar este problema

(a) Imagen RGB (b) Imagen de profundidad (c) Primer plano

Figura 4.21: Extracción de primer plano por diferencia de imágenes erosionado

podemos tratar la imagen previamente para unificar el nivel de gris de los objetos y además distinguirlos mejor del fondo. Esto disminuiría algo la fluidez del componente, pero como con esta técnica estamos analizando unos 19 fotogramas por segundo, el aumento de tiempo de procesamiento no sería prácticamente perceptible.

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen segmentada

Figura 4.22: Segmentación mediante *watershed*

Finalmente, en la figura 4.24 podemos comprobar cómo el algoritmo basado en la librería cvBlob funciona satisfactoriamente incluso en entornos más complejos que los vistos anteriormente.

Por tanto, teniendo en cuenta los resultados obtenidos y el tiempo de procesamiento de cada método, llegamos a la conclusión de que la extracción por mezcla de gaussianas combinada con segmentación basada en cvBlob, es la opción con la que se obtienen mejores resultados.

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen segmentada

Figura 4.23: Segmentación mediante *watershed*

(a) Imagen RGB (b) Imagen de profundidad (c) Imagen segmentada

Figura 4.24: Segmentación mediante cvLabel

Capítulo 5

Conclusiones y trabajos futuros

Tras proponer y describir una solución a los problemas planteados al inicio de este Proyecto, a continuación se ponderará si se han cumplido los objetivos propuestos al principio de esta memoria. También se sugerirán futuras líneas de investigación relacionadas con este trabajo.

5.1. Conclusiones

Tal y como se describió en el capítulo 2, el objetivo principal de este Trabajo Fin de Grado es desarrollar un prototipo capaz de detectar y contabilizar las personas que pasan por delante del sensor. Vistos los resultados de los experimentos del capítulo anterior, se puede afirmar que los hemos cumplido en su mayor parte.

El objetivo estaba dividido en 3 subobjetivos: extracción de primer plano, segmentación de personas en la imagen y seguimiento y conteo de las mismas.

Para implementar el primer subobjetivo nos hemos apoyado en diversas funciones de la librería OpenCV, ofreciendo de este modo varias alternativas para el segmentador de fondo como diferencia directa de imágenes o mezcla de guassianas. Este subobjetivo se ha conseguido ya que hemos extraído con éxito en la gran mayoría de los casos el primer plano en la escena a analizar.

Para lograr el subobjetivo de la detección de *blobs*, se ha incluido un detector de *blobs* utilizando cvLabel de la librería cvBlob y también otro método basado en la técnica de *watershed* apoyándonos para ello funciones de la librería OpenCV. Sin embargo esta última técnica no ha proporcionado los resultados esperados, ya que la segmentación obtenida no se correspondía en muchos casos con el número de *blobs* reales en la imagen original, debido a problemas tales como fragmentación de *blobs* y la existencia de falsos positivos. Pero podemos decir que este subobjetivo se ha logrado ya que el método basado en cvBlob sí ha proporcionado en la mayoría de los casos un resultado apropiado.

En cuanto al tercer y último subobjetivo, se procedió a implementar el seguimiento visual de personas y su conteo. Para ello, se ha desarrollado un sistema de seguimiento de *blobs* utilizando la librería cvBlob. De este modo, se obtienen muy buenos resultados en el seguimiento de personas en las pruebas realizadas.

Respecto a los requisitos del programa, también puede decirse que se cumplen según lo fijado:

- El software es capaz de recoger datos de vídeo de una fuente RGBD cualquiera. Gracias al componente *OpenniServer*, el cual está captando datos de vídeo de la fuente que sea que se le configure y se lo transmite al sistema desarrollado, el origen de estos datos de vídeo es opaco para el programa.
- Durante el procesamiento de las imágenes en tiempo real el coste computacional ha sido muy razonable y en general no han existido ralentizaciones que pudieran afectar al sistema de conteo de personas.
- El lenguaje utilizado para implementar el componente ha sido C++.
- El componente se ha construido siguiendo la estructura del componente de JdeRobot *basic component*, el cual tiene dos hebras de procesamiento independientes que se encargan de una funcionalidad específica, controlando una la parte operacional y la otra el GUI. También el código se ha separado en distintas partes, dependiendo de su funcionalidad: API, control y GUI.
- La implementación software consta de distintos módulos y clases desarrollados para permitir la reutilización o sustitución de las diversas partes que componen el algoritmo.

Durante la realización de este Trabajo Fin de Grado se han adquirido conocimientos sobre técnicas de visión por computador empleando sensores RGBD. Además, se han adquirido conocimientos y profundizado respecto al software de infraestructura utilizado, entre ellos: C++, OpenCV, PCL, OpenGL, Latex, ICE, cvBlob, Qt y SVN.

5.2. Trabajos futuros

El trabajo en este Trabajo Fin de Grado abre las puertas a otras vías de investigación, pudiendo mejorar el sistema desarrollado y dando pie a aplicaciones más robustas apoyándose en este prototipo.

- Mejorar el sistema de detección de los *blobs*, basándose en sistemas tales como el flujo óptico o una mejora del detector de contornos, para aumentar la precisión de los resultados del programa.
- Un mejor sistema para detectar personas en entornos de escasa visibilidad o desde distintos ángulos.

- Un autoajuste de parámetros para la segmentación de fondo o un juego de parámetros válido para un conjunto amplio de vídeos de entrada con distintas condiciones de visibilidad. La idea es minimizar la intervención del usuario en el procesamiento de vídeos.
- Seguimiento directamente en 3D, aprovechando la información de distancia y aplicando filtros probabilísticos como los filtros de Kalman.

Bibliografía

[1] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding* (8 Sept. 2003).

[2] Marugán Alonso, S. Seguimiento 3d visual de múltiples personas utilizando un algoritmo evolutivo multimodal. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2007.

[3] Marugán Alonso, S. Seguimiento visual de personas mediante evolución de primitivas volumétricas. Trabajo fin de carrera, Universidad Rey Juan Carlos, 2010.

[4] Marugán Alonso, S. Sistema autónomo de detección de caídas con recepción de alarmas en un teléfono móvil. Trabajo fin de máster, Universidad Rey Juan Carlos, 2010.

[5] Navarro, J. Construcción de mapa 3d desde sensores rgb d. *Proyecto Fin de Carrera, URJC* (2015).

[6] Rivas Montero, F. M. Algoritmo evolutivo para detección y seguimiento de personas en 3d con sensores rgbd. Trabajo fin de grado, Universidad Rey Juan Carlos, 2014.

[7] Rivas Montero, F. M. Seguimiento de personas en 3d basado en sensores rgb-d. Trabajo fin de grado, Universidad Rey Juan Carlos, 2014.

[8] Zivkovic, Z. Improved adaptive gaussian mixture model for background subtraction. *International Conference Pattern Recognition* (2004).

[9] Zivkovic, Z. Pattern recognition letters. *Eficient Adaptative Density Estimation per Image Pixel for the Task of Background Subtraction* (2006).