uc3m | Universidad Carlos III de Madrid

Grado de Ingeniería en Tecnologías Industriales
2016/2017

Trabajo Fin de Grado

Sistema de detección de caídas de personas mayores por medio de visión artificial

---

Sergio González Díaz

Tutor
Enrique Pelayo Campillos

Octubre 2017

Dedicado a mi familia

Agradecimientos

En primer lugar, quería agradecer a mi familia, en especial a mi madre, por entenderme y apoyarme durante este largo año que ha supuesto realizar este proyecto. Les agradezco su ánimo, su tiempo y sobre todo, su paciencia, por saberme tratar en los momentos más difíciles. Con su ayuda todo ha resultado más fácil.

En segundo lugar, a mi tutor Enrique, por proponerme este interesante trabajo y guiarme durante todo el desarrollo.

También quería agradecer a mis amigos de toda la vida, por haber sabido aguantarme durante todos estos meses y mostrarme su paciencia y ayuda cuando lo necesitaba.

Por último, pero no menos importante, a mis compañeros de clase, los cuales ya considero más que compañeros. Gracias por ayudarme con el proyecto, pero especialmente gracias por sacarme sonrisas, por escucharme y por tantos buenos momentos durante estos cuatro difíciles, pero increíbles años.

Muchas gracias a todos.

Resumen

En torno a un 30 % de la población de edad avanzada sufre al menos una caída cada año, siendo éstas la segunda causa mundial de mortandad por lesiones accidentales o no intencionales. Muchas de estas caídas se producen en individuos que viven solos, en el interior de sus casas, donde no pueden ser socorridos. Además, debido a la crisis actual y a los recortes en sanidad pública, estas personas no pueden costearse el pago de residencias o clínicas privadas. Por otro lado, los avances y la investigación en el campo de la visión artificial y sus aplicaciones, hacen de esta ciencia una tecnología atractiva para ser usada en sistemas de todo tipo.

Por estas razones, este proyecto tiene como finalidad el desarrollo de un sistema de detección de caídas para personas mayores basado en la visión artificial, que mandará un aviso por correo electrónico cuando se detecte la caída. La aplicación está pensada para ser usada de forma doméstica, apareciendo a un precio asequible para el consumidor. Estará montada sobre una Raspberry Pi, y mediante el uso de la librería OpenCV se hará el procesamiento de las imágenes. En esta memoria se diseñará y explicará el desarrollo del sistema.

Palabras clave: Detección de caídas, Personas mayores, Visión Artificial, OpenCV, Raspberry Pi

Abstract

About 30 % of the elderly population suffers at least one fall each year, being these the second leading cause of death from accidental or unintentional injuries. Many of these falls occur in individuals living alone, inside their homes, where they can not be rescued. In addition, due to the current recession and cuts in public health, these people can not afford to pay for residences or private clinics. On the other hand, advances and research in the field of artificial vision and its applications make this science, an attractive technology to be used in systems of all kinds.

Therefore, this project aims to develop a fall detection system for older people based on artificial vision, which will send an alarm by email when the fall is detected. The device is intended to be used in a domestic environment, having an affordable price for the consumer. It will be mounted on a Raspberry Pi, and OpenCV library will make the image processing. In this report, the development of the system will be designed and explained.

Keywords: Fall detection, Elderly people, Computer vision, OpenCV, Raspberry Pi

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación del proyecto

Las personas con un alto grado de riesgo (edad avanzada, enfermedades, discapacidad) son las más propensas a tener accidentes por caídas. Según datos recogidos en el artículo "Lesiones por caídas y factores asociados en personas mayores de Cataluña, España"del *Repositorio Institucional de la OPS/OMS* [1], entre el 28 % y 34 % de las personas de 65 años sufren al menos una caída por año, porcentajes que aumentan con la edad.

Nuestra sociedad está experimentando un incremento de personas mayores, a la vez que se recortan presupuestos en seguridad social y atención médica. Ante esa situación, la tendencia actual hoy en día, pasa por el mantenimiento de estas personas en sus propias viviendas, con el fin de evitar el pago de centros especializados y clínicas privadas.

Este proyecto de fin de grado nace con la motivación de diseñar una aplicación que haga uso de la visión artificial, que sirva para ayudar a este segmento de la población. Los sistemas de visión artificial son una tecnología barata de implementar, fiable y fácil de usar, lo que supone una opción interesante a la hora de desarrollar cualquier sistema. Por esa razón, se ha buscado una aplicación que permita adaptar este tipo de tecnología en la vida de una persona, de forma que aumente su calidad de vida de forma sencilla.

1.2. Objetivos

El principal objetivo de este proyecto es el diseño y desarrollo de un sistema que sea capaz de detectar caídas, en particular de personas mayores. Así mismo, este dispositivo avisará a un familiar o conocido en caso de activación. El dispositivo se instalaría dentro de la vivienda de la persona, y tendría un uso comercial. Por ello, el sistema desarrollado deberá tener un coste asequible, acorde al capital del segmento de consumidores a los que va dirigido.

El código desarrollado se ejecutará en una Raspberry Pi 3, utilizando Pyt-

hon como lenguaje de programación y OpenCV como la librería principal para el procesamiento de las imágenes. Para capturar el vídeo se usará una cámara web convencional.

El código no se escribirá directamente en la Raspberry Pi, sino que se desarrollará en un ordenador personal. En las pruebas finales se implementará en la placa. De esta forma, será más fácil escribir el código y probarlo.

Además, el código desarrollado será de código abierto *"open source"*, por lo que cualquier persona podrá revisar el proyecto y mejorarlo.

1.3. Estructura del documento

La memoria del proyecto se estructurará de la siguiente manera:

- El segundo capítulo consistirá en el estudio de la literatura existente dentro del campo de las detecciones de caídas y de la visión artificial.
- En el capítulo 3 se describirán los componentes hardware y software del sistema.
- En el capítulo 4 se hablará sobre la elección de los vídeos utilizados para realizar las experimentaciones.
- En el capítulo 5, se detallará y explicará cada una de las etapas del algoritmo desarrollado.
- Los resultados y su análisis se encuentran el capítulo 6.
- En el capítulo 7, se explicarán los pasos necesarios para instalar el dispositivo, y las recomendaciones para su correcto funcionamiento.
- En el capítulo 8 se habla sobre la gestión del proyecto y su planificación.
- En el capítulo 9 se muestra el marco regulador para este proyecto, analizando la legislación aplicable al producto desarrollado.
- El capítulo 10 se basa en el entorno socio-económico, e incluye el presupuesto de realización del proyecto y el análisis de una posible comercialización frente a productos similares en el mercado.
- En el penúltimo capítulo se definen una serie de mejoras futuras que pueden ayudar a mejorar el sistema.
- Por último, en el capítulo 12, se darán las conclusiones finales del proyecto.
- Al final del documento, se ha incluido un anexo con los códigos del programa y la bibliografía utilizada.

Capítulo 2

Estado del arte

Las caídas son uno de los mayores riesgos a los que se enfrenta la población de avanzada edad. Según la OMS [2], las caídas son la segunda causa mundial de muerte por lesiones accidentales o no intencionales, siendo los mayores de 65 años quienes más sufren caídas mortales.

Cada año se producen 37,3 millones de caídas que, aunque no son todas mortales, requieren de atención médica y suponen la pérdida de más de 17 millones de años de vida ajustados en función de la discapacidad (AVAD). Estas caídas tienen un gran impacto físico y psicológico en la víctima, ya que conducen a lesiones que afectan negativamente la autonomía personal y la calidad de vida .

Según el artículo *Detector automático de caídas y monitorización de actividad para personas mayores* de la Revista española de geriatría y gerontología [3], la población de la tercera edad es el segmento de población que más crece, aumentando la tendencia los próximos años. Si continúa esta tendencia, en el año 2035, un 33 % de la población tendrá más de 65 años. Al mismo tiempo, debido a la crisis actual y a las gestiones de los gobiernos competentes, los servicios destinados a la salud pública han visto un decrecimiento importante de los presupuestos y una presión para reducir los gastos. Además, también existe una escasez de lugares de atención para personas de la tercera edad. Por estas razones, una de las soluciones pasa por implementar sistemas de mantenimiento en las propias viviendas de los afectados.

La organización de este capítulo es la siguiente. En el apartado 2.1 se hará un repaso de las diferentes tecnologías existentes en el ámbito de detección de caídas, y se discutirán sus ventajas e inconvenientes. En la sección 2.2 se verán las dificultades que conlleva crear un sistema de este tipo usando la visión artificial. Se hará un repaso de la definición de visión artificial y sus aplicaciones en el apartado 2.3. En el 2.4, se hará un estudio de las diferentes técnicas de substracción de fondo. Por último, en el apartado 2.5 se verán algunos de los criterios disponibles para determinar una caída.

2.1. Sistemas de detección de caídas

En la actualidad, existen diferentes productos que permiten la detección de caídas. En esta sección se mostrarán algunos de esos dispositivos y sus ventajas e inconvenientes respecto a un sistema basado en visión artificial.

Las caídas, como se explica en el punto anterior, suponen un gran riesgo para la población de edad avanzada. Para reducirlas, la mejor manera y la más obvia es la prevención. Sin embargo, este tipo de incidentes no se pueden predecir, y son muy difíciles de controlar. Ante esta situación, la alternativa es avisar cuando ya ha ocurrido la caída, algo más fácil de detectar, y que puede salvar muchas vidas.

Dentro del área de telemedicina han sido desarrollados diferentes sistemas que permiten avisar a un familiar o cuidador en caso de alguna emergencia. Uno de los dispositivos más usados es la telealarma, consistente en un collar o aparato con un botón de pánico, conectado al teléfono fijo del usuario. Este aparato resulta de gran utilidad, pero tiene bastantes limitaciones. La limitación principal ocurre cuando el usuario no puede accionar el botón. Esto puede deberse a diversos motivos, como que la persona se halle inconsciente o tenga alguna lesión que le impida pulsar. Otro inconveniente se halla en la propia dependencia del usuario con el aparato. Las personas con riesgo de caídas, y en especial las personas de avanzada edad, pueden olvidarse de llevar consigo el dispositivo, por lo que cuando se ha producido la caída, el usuario no puede acceder al botón. Por último, el sistema está limitado a un uso interior, por lo que deja indefenso al usuario frente a caídas fuera de la vivienda.

Figura 2.1: Sistemas de telealarma

Otro acercamiento a este problema podría ser la utilización de cámaras de vigilancia tradicional. Estos sistemas realizarían una grabación continua del usuario, por lo que en caso de caída, se podría observar el evento y actuar en consecuencia. Sin embargo, por lo general no realizan ningún tipo de análisis de vídeo, por lo que solo se podrían usar para vigilancia remota, y las caídas se detectarían con demasiado retraso. Además, este tipo de sistemas acarrearía varios problemas de privacidad.

Figura 2.2: Sistema de videovigilancia CleverLoop

Los sistemas de telealarma y videovigilancia sirven en múltiples contextos, no solo para la detección de caídas. Sin embargo, para el caso que se discute, no es una opción recomendable, aunque sí se pueden usar como método auxiliar. Se requiere de dispositivos con una serie de características específicas para poder realizar las detecciones con precisión y que no supongan una incomodidad al usuario.

A continuación, se muestran diferentes tipos de tecnologías destinadas a la detección de caídas, las cuales se pueden separar en cuatro grupos distintos, según el artículo "Detector automático de caídas y monitorización de actividad para personas mayores" de la Revista española de geriatría y gerontología [3]:

- **Dispositivos portátiles o "ponibles" con detección inmediata:** Detectores de tamaño reducido que lleva la persona, capaces de detectar de forma inmediata una caída y activar una alarma. Igual que con la con la telealarma, tienen la desventaja de depender del usuario para llevarlo encima, por lo que, si se olvida o no se pone correctamente, no se detecta la caída. Estos sistemas utilizan tecnologías de posición, choque, acelerómetros y sensores de inclinación para la detección.
- **Dispositivos portátiles o "ponibles" de comportamiento inusual:** Este grupo, al igual que el anterior, incluyen detectores portátiles, sin embargo, la detección no es inmediata. Se encargan de monitorizar la actividad del usuario y comparan su comportamiento con patrones de su rutina habitual. En el momento en el que se detecta alguna anomalía, el dispositivo activa la alarma. Por su naturaleza, este tipo de tecnología puede tardar bastante tiempo en mandar el aviso. Usan sensores de ritmo cardíaco, sudoración, posición, etc.
- **Monitorización ambiental con detección inmediata:** Detectores que son instalados en el entorno habitual del usuario. Estos sensores monitorizan

la actividad de la persona y detectan cuando se produce una caída. Tienen la ventaja de que una vez instalados no requieren del usuario para ser activados, por lo que pueden ser usados sin necesidad de intervención de la persona. Por otro lado, las principales desventajas son la necesidad de instalación en múltiples regiones de la vivienda, lo que puede resultar en un sistema intrusivo y caro. Los sensores pueden ser diversos: cámaras (análisis de imagen), sensores de choque en el suelo o mobiliario, micrófonos (análisis de sonido), etc.

- **Monitorización ambiental de comportamiento inusual:** Al igual que los dispositivos anteriores, son sensores que se instalan en el entorno para monitorizar el comportamiento del usuario. Se encargan de detectar comportamientos anómalos de la persona a partir de la información recogida por los sensores. Este tipo de sistemas son interesantes de cara al usuario, ya que pueden detectar diferentes tipos de anomalías, pero al igual que el segundo grupo necesitan bastante tiempo para asegurarse del problema. Además, también es un sistema que puede ser invasivo y costoso. Usan sensores infrarrojos, sensores de contacto en puertas y ventanas, análisis de imágenes, etc.

En resumen, los sistemas portátiles tienen la ventaja de poder ser usados fuera del domicilio de la persona, pero dependen del usuario para poder ser usados, además de resultar en ocasiones molestos. Por otro lado, la monitorización de la actividad es un sistema interesante para el conjunto de problemas que pueden ocurrir al usuario. Sin embargo, cuando se trata de detectar caídas no es una tecnología válida, ya que se requiere una reacción rápida por parte del cuidador.
A continuación, se muestra un esquema de las ventajas e inconvenientes de cada método:

	Detección inmediata	Detección de comportamiento inusual
Dispositivos portátiles	**Detección de caída inmediata** **Dependencia del usuario** **Interiores y exteriores**	**Detección de caída no inmediata** **Dependencia del usuario** **Interiores y exteriores**
Monitorización ambiental	**Detección de caída inmediata** **Sin dependencia del usuario** **Entorno habitual del usuario**	**Detección de caída no inmediata** **Sin dependencia del usuario** **Entorno habitual del usuario**

Tabla 2.1: Ventajas e inconvenientes de dispositivos de detección de caídas.

Como se ha visto, existe una gran variedad de sistemas de detección de caídas. La elección de cada sistema residirá en la situación del paciente y el enfoque de la detección de la caída.

En este proyecto se desarrollará un sistema perteneciente al tercer grupo. Los dispositivos portátiles no son los más indicados para ser usados por personas mayores, al ser fácilmente olvidados por estos o ser incómodos. El dispositivo creado se basará en un sistema de visión artificial que sea capaz detectar de forma

inmediata las caídas. De esta forma, se evitará el contacto físico del sistema con el individuo al no ser necesario ningún tipo de sensor pegado al usuario. Este sistema dispondrá de una o varias cámaras que serán las encargadas de recoger las imágenes del entorno, y una unidad de procesamiento que sea capaz de analizar esas imágenes y enviar la señal a la persona correspondiente. En la siguiente imagen, se muestra el esquema de un típico sistema de detección de caída por cámara.

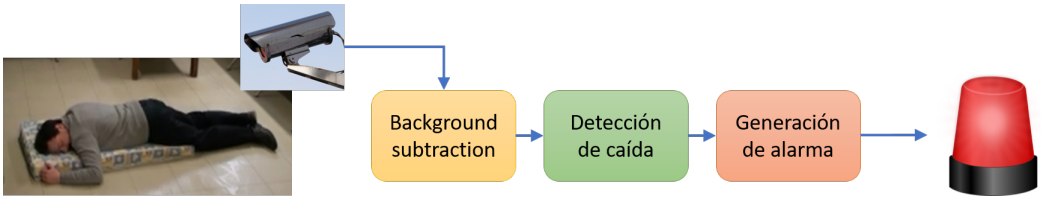

Figura 2.3: Esquema de sistema de detección de caída

Como se ve en la figura 2.3 la primera etapa del algoritmo consistirá en la sustracción de fondo para localizar los elementos en movimiento. Cuando se hayan detectado, se extraerán los datos para comprobar si se ha producido una caída. Una vez se haya confirmado la caída, la última etapa será la generación de la alarma.

En la siguiente sección se mostrarán las dificultades existentes a la hora de crear una aplicación de este tipo.

2.2. Dificultades para implementar un sistema de visión artificial

Como se explica en el punto anterior, cada método de detección de caídas tiene sus ventajas e inconvenientes. Aunque se ha decantado por usar un sistema basado en la utilización de cámaras, igualmente existen ciertos problemas que se explicarán a continuación de forma detallada.

2.2.1. Privacidad del usuario

En primer lugar, existe el problema de la privacidad del usuario [4]. La persona que usará este sistema, será grabada con cámaras con el fin de recoger los datos necesarios para detectar la caída. Sin embargo, esto puede ser un problema de cara a la privacidad de esa persona. El usuario debe ser consciente en todo momento de este hecho para tener permiso de instalar el sistema. Aparte, se debe informar a la persona de que las grabaciones no serán vistas por otras personas, y que no serán guardadas ni usadas para ningún otro propósito (salvo que sean necesarias para los periodos de pruebas, en cuyo caso será informado).

También cabe la posibilidad de mandar una imagen de la caída a la persona encargada de recibir la señal, como se explicará en el punto 5.10, con el fin de tener una mejor interpretación de la situación. Incluso, mandar el vídeo a partir

de este punto. Es por ello, que el usuario debe ser informado antes de aplicar estas mejoras en el sistema.

2.2.2. Dificultades al desarrollar el algoritmo

Como se explica en el artículo "Computer visión system for in-house video surveillance" de R. Cucchiara et al [5], existen diferentes complicaciones que aparecen al desarrollar un algoritmo que haga uso de sistemas de vigilancia visual. Los principales problemas de la vigilancia en entornos interiores se pueden resumir en 4 puntos.

a. La iluminación artificial de distintas fuentes afecta a la detección de movimiento, debido a la generación de sombras. Las sombras provocan la activación de falsos positivos, alterando la forma y tamaño de los objetos detectados. Por ello, será necesario aplicar un filtro de detección de sombras para solucionar este problema.

b. Las viviendas tienen muebles, los cuales son a menudos cambiados de posición. Una silla, o una mesa, por ejemplo, son elementos movidos con frecuencia, lo que puede alterar los resultados de la aplicación. Los sistemas que usan sustracción de fondo (usualmente sistemas de cámara fija) es importante que actualicen rápidamente la imagen del background, para poder reaccionar a estos cambios.

c. Los principales parámetros usados en los sistemas de seguimiento por visión se basan en el tamaño y forma de los objetos rastreados y sus cambios. Sin embargo, en una escena pueden aparecer oclusiones (cuando un objeto tapa parcialmente a otro) provocados por un objeto u otra persona.

d. Los entornos dentro de una casa son a menudo complejos. Es muy difícil recoger todos los movimientos de una persona con una única cámara. Dependiendo de la situación, es recomendable la utilización de varias cámaras.

2.2.3. Consideraciones del hardware

Jared Willems [4] explica los problemas principales que aparecen al trabajar con los elementos hardware del sistema:

- En primer lugar, los algoritmos suelen ser desarrollados en lugares donde se cuenta con amplia capacidad de procesamiento. Cuando el sistema se lleva al terreno comercial, es necesario abaratar costes, por lo que este poder de procesamiento se reduce drásticamente.

- Aparte de las limitaciones computacionales, el algoritmo puede necesitar de una gran calidad de imagen para funcionar. Si esto ocurre, el producto final necesitará hacer usos de cámaras de alta calidad, incrementando el coste total.

- La presencia de cámaras en las viviendas de las personas mayores, puede incomodar al usuario, invadiendo su sensación de privacidad. Es por ello que las cámaras deberán ser colocadas en lugares donde no llamen la atención.

- Por último, existen algoritmos que permiten trabajar con oclusiones, pero dado a las complicaciones que presentan es mejor evitarlas directamente. Es decir, habrá que colocar las cámaras en lugares donde se reduzcan las posibilidades de que aparezcan oclusiones en la escena.

2.3. Visión artificial

2.3.1. Introducción a la visión artificial

Antes de empezar a hablar de las diferenetes técnicas de la visión artificial y su implementación en la aplicación desarrollada, es necesario definir qué es realmente la visión artificial.

Según un artículo del Ministerio de Educación sobre aplicaciones prácticas de la visión artificial en el control de procesos industriales [6], la visión artificial se puede definir como:

> un campo de la "Inteligencia Artificial" que, mediante la utilización de las técnicas adecuadas, permite la obtención, procesamiento y análisis de cualquier tipo de información especial obtenida a través de imágenes digitales.

Como humanos, percibimos la estructura tridimensional del mundo que nos rodea con aparente facilidad. Somos capaces de percibir los tamaños y formas de los objetos, determinar sus colores y texturas, identificar el número y posición de éstos, y todo ello solamente con nuestros ojos. Sin embargo, esta capacidad es muy complicada de reproducir por un computador. Es en este punto donde se desarrolla el concepto de visión artificial.

La visión artificial o visión por computador, nace con el objetivo principal de entender la historia detrás de una imagen. Lo que para una persona resulta algo trivial, para un ordenador esta tarea resulta extremadamente complicada. Cada día se generan millones de imágenes, tendencia que va en aumento año tras año. La necesidad de controlar y analizar estas imágenes y vídeos, implica que la disciplina del tratamiento de imágenes sea un estudio en continua evolución.

Desde la perspectiva de los ordenadores, una imagen es simplemente un conjunto de números. Por sí solos estos valores no significan nada para un ordenador. El ordenador tiene que interpretarlos. En general, como explica Serge Belongie, investigador en Google especializado en visión artificial, hay cuatro estrategias para hacerlo [7]:

- **Reconocimiento (Recognition):** Esta técnica consiste en saber en una foto donde están los objetos y qué son. A nivel genérico resulta sencillo de realizar, pero a nivel específico es una tarea más complicada, en el sentido de poder reconocer por ejemplo un árbol, pero no el tipo.
- **Reconstrucción (Reconstruction):** La reconstrucción física es dar forma tridimensional a los elementos de una imagen. A partir de varias fotos, se

aplican algoritmos que comparan datos de esas imágenes para crear otras en tres dimensiones. Programas como Google Street View capturan datos bidimensionales a partir de imágenes panorámicas, que luego se transforman en mapas 3D.

- **Registro (Registration):** El registro es la detección o alineación de modelos. Se comparan los elementos de una imagen con patrones o modelos ya definidos, como pueden ser rostros, códigos de barras, etc., y se busca si existe una coincidencia. Un ejemplo de esto puede ser un coche automático que detecta peatones y señales de tráfico, o cámaras para tomar sefies.

- **Reorganización (Reorganization):** Por último, se encuentra la reorganización, conocido habitualmente como aprendizaje no supervisado. En el caso de los niños, ¿cómo son capaces de aprender cosas sobre el mundo sin que se les explique todo? Esta estrategia se basa en esta premisa. No se programa el ordenador para que haga una tarea específica, sino que se le enseña a base de prueba y error hasta que desarrolla su cometido con un alto porcentaje de aciertos. Los modelos de visión artificial siempre se han entrenado con cantidades enormes de datos. El objetivo del aprendizaje no supervisado es obtener los datos y organizarlos de forma que tengan sentido.

2.3.2. Aplicaciones de la visión artificial

Desde su introducción en la industria por la época de los años 80 hasta la actualidad, esta disciplina ha visto un espectacular desarrollo tecnológico, gracias a los beneficios que han supuesto los rápidos avances en las áreas de redes e informática. Como se explica en el libro Computer Vision: Algorithms and Applications [8], los investigadores en visión artificial han estado desarrollando y mejorando numerosas técnicas matemáticas para capturar las figuras tridimensionales y apariencia de los objetos a partir de imágenes. Gracias a ello tenemos actualmente métodos fiables que nos permiten generar objetos 3D a partir de imágenes bidimensionales, realizar seguimientos de personas (tracking) o incluso reconocimiento de personas.

Las ventajas que ofrece esta tecnología son numerosas. Su versatilidad y fácil implementación han provocado que su uso se haya extendido a muchos sectores. Sus aplicaciones más usuales se encuentran en la industria, la cual abarca la informática, la óptica, la ingeniería mecánica y la automatización industrial. También existen aplicaciones más cotidianas y comerciales, como la detección de rostros en cámaras, códigos de barras o videojuegos con realidad aumentada [7].

Figura 2.4: Vehículo Waymo con visión artificial

En resumen, son abundantes las áreas y campos de la ciencia donde la visión artificial esté implementada de una forma u otra. Por ello, sólo es posible dar una pequeña pincelada sobre los múltiples sectores en los que la visión artificial se ha aplicado hasta el momento. En el libro "Visión por Computador: Imágenes digitales y aplicaciones" [9], se muestran algunas de las muchas áreas donde está presente esta tecnología, además de algunos ejemplos de procesos de utilidad de las áreas en cuestión:

- **Computadores**: soluciones software y hardware para los diferentes métodos.
- **Medicina**: bombeo de sangre del corazón mediante observación del volumen en los movimientos de sístole y diástole.
- **Microscopía**: crecimiento de bacterias, cuenteo e identificación de partículas.
- **Robótica**: orientación espacial del robot, reconocimiento de objetos.
- **Astronomía**: cuenteo e identificación de estrellas.
- **Cartografía**: identificación de límites y áreas en superficies, reconstrucción física de escenas.
- **Control de Calidad**: en metalurgia, alimentación, textil, etc.
- **Seguridad**: detección de objetos en movimiento.
- **Otros**: reconocimiento de objetos, lectura de paneles o documentos, etc.

2.3.3. Etapas de un sistema de visión artificial

El principal objetivo de la visión por computador es la de obtener una descripción de una imagen. Sin embargo, como ya se ha explicado, para un ordenador es una tarea complicada. Dotar a un computador de la capacidad de ver no es sencillo, ya que a la hora de obtener una imagen intervienen factores como cambios en la iluminación, cambios de escala, movimiento, pérdida de información,

etc. La solución para manejar esta información pasará por descomponer la información obtenida en niveles de visión, reduciendo así la complejidad del problema. Estos niveles son los siguientes [10]:

1. **Nivel bajo**: Se trabaja directamente con los puntos de la imagen digitalizada (píxeles). En este primer nivel se extraen características como el gradiente, el color, la textura, la profundidad, etc.
2. **Nivel intermedio**: Con los datos obtenidos en el nivel inferior, se agrupa la información y se obtienen bordes, líneas, regiones, etc. para segmentar la imagen.
3. **Nivel alto**: Por último, en este nivel se interpretan los elementos obtenidos en los niveles anteriores utilizando modelos o conocimientos a priori del problema.

Aunque cada aplicación de visión artificial tiene su propia manera de funcionar y tratar los datos, la forma de obtenerlos y su procesamiento posterior es la misma. Los dos pilares de un sistema de visión artificial son el sistema de formación de las imágenes y el sistema de procesamiento.

El primer punto está constituido por el subsistema de iluminación, de captación de la imagen y de adquisición de la señal en el computador. La imagen analógica es capturada por una cámara, sensor o similar. Después es digitalizada para poder ser manipulada por un ordenador.

El segundo punto se refiere, como su nombre indica, al procesamiento de las imágenes obtenidas. Se trata de la parte software del sistema. Se encarga de manejar la información digitalizada recibida y le da un sentido lógico para el sistema para el que está desarrollado. Cuando la señal se introduce en el computador, ésta es procesada mediante los algoritmos para transformarla en información de alto nivel. Para ello, es necesario también usar métodos de preprocesado que mejoren la calidad de la imagen (ruidos, sombras, mala iluminación, etc.).

Estos dos puntos se pueden subdividir en 6 etapas [11]

1 **Sensado o adquisición de la imagen**: Primera etapa en la cual se construye el sistema de adquisición de las imágenes y se digitalizan.

2 **Preprocesado**: Mejora de la calidad informativa de la imagen (reducción de ruido y enriquecimiento de detalles).

3 **Segmentación**: División de la imagen en áreas de interés.

4 **Representación y descripción**: Extracción de características de los objetos de interés.

5 **Reconocimiento**: Proceso en el que se identifican esos objetos.

6 **Interpretación**: Asignación de un significado a un conjunto de objetos reconocidos.

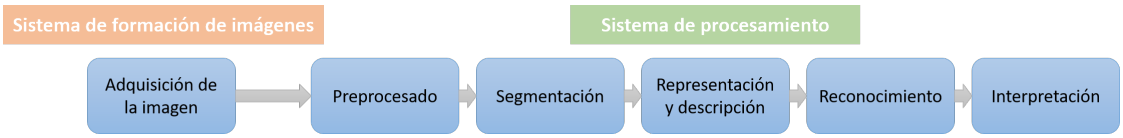

Figura 2.5: Etapas en una aplicación de visión artificial

Es necesario mencionar que los sistemas de tratamientos de imágenes requieren un elevado coste computacional. Se trabaja con grandes cantidades de datos, y los procesos requeridos suelen ser complejos y costosos. Es por ello que muchos analistas han desarrollado procesos para ser implementados en sistemas basados en arquitecturas paralelas, como el que desarrolla Hussain [12]. Por suerte, muchos de estos problemas están siendo actualmente superados gracias a los continuos avances tecnológicos.

En los siguientes apartados se hará un repaso de las diferentes técnicas de substracción de fondo y de detecciones de caídas. En el capítulo 5, se explicará en más detalle los pasos llevados a cabo para realizar el procesamiento de las imágenes.

2.4. Substracción de fondo

Con el objetivo de conseguir crear un sistema que detecte caídas mediante imágenes, el primer paso será localizar personas en la escena. Para ello es necesario que el software desarrollado sea capaz de detectar movimiento. Existen diferentes métodos, pero el más común y el que se usará será la substracción de fondo.

La substracción de fondo (o background subtraction en inglés) extrae los objetos en movimiento que hay en la escena. Consiste, en líneas generales, en la diferencia de dos frames de la escena (o un conjunto de ellos, como se verá más adelante), que devuelve una imagen del objeto u objetos que se han desplazado. Es decir, se toma como referencia un fondo o imagen sin movimiento y se le restan los sucesivos fotogramas que vamos obteniendo. La imagen sin movimiento se llama fondo o segundo plano (background en inglés), y el fotograma a analizar es el primer plano (foreground en inglés).

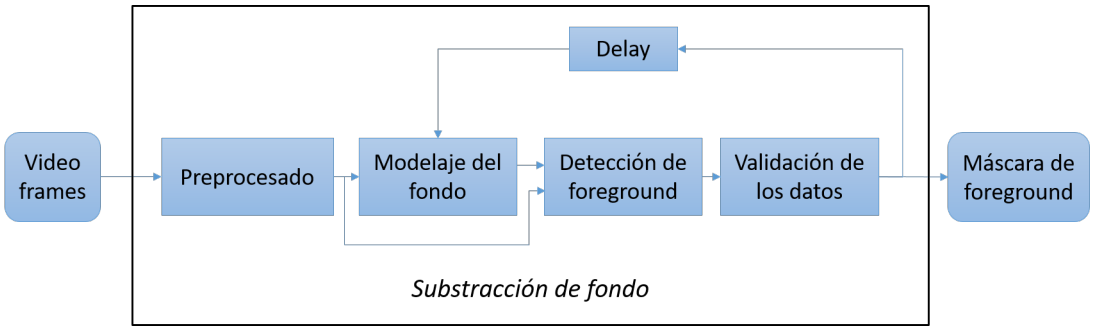

Figura 2.6: Etapas de la substracción de fondo

Esta técnica es bastante eficiente y no necesita de sensores externos para detectar movimiento, pero presenta algunas desventajas. La primera es que este método es muy sensible a cambios de iluminación de la escena, provocados por cambios en la luz natural o sombras. Otro inconveniente es que, si los objetos a detectar tienen un color parecido al del fondo, será difícil detectarlo. Aparte, está el problema de la propia cámara, la cual generará mucho ruido si tiene poca calidad de imagen.

Existen varias técnicas y algoritmos basados en la substracción de fondo que posibilitan la detección de movimiento. Cada caso es distinto y dependerá de la situación la elección de cada uno.

Para poder desarrollar una aplicación que detecte correctamente personas, será necesario estudiar los diferentes tipos de técnicas existentes. Dependiendo de la forma de obtener el fondo o segundo plano, existen dos modalidades dentro de la substracción de fondo.

2.4.1. Substracción con imagen de referencia

Este primer grupo consiste en capturar una imagen de la escena donde no haya ningún elemento en movimiento y usarla como imagen de referencia. A partir de ésta se aplica la resta a los sucesivos fotogramas para poder detectar el movimiento de los objetos. Normalmente se utiliza el primer fotograma de una secuencia de vídeo.

La parte negativa de este método es que es muy sensible a los movimientos de cámara. Un ligero golpe a la cámara podría desencuadrar las imágenes del foreground respecto a la imagen de referencia, provocando la aparición de falsos positivos. Por otro lado, los cambios en la iluminación también son un problema. Por ejemplo, una imagen de una escena, a una determinada hora del día con luz natural no tendrá las mismas condiciones lumínicas que la misma escena a otra hora.

Una aplicación práctica se encontraría al aplicarse en entornos con condiciones lumínicas controladas y cámara estática, en la que se detectaría con bastante precisión las siluetas de los objetos presentes, estén o no en movimiento.

2.4.2. Substracción con fotogramas anteriores

En este caso, la obtención del segundo plano se consigue a partir de los fotogramas anteriores. Es decir, el fondo se va actualizando continuamente a partir de los frames entrantes.

El principal inconveniente radica en el hecho de que, si la persona u objeto no se mueve dentro de la escena, el sistema no es capaz de detectarlo, ya que el segundo plano se actualizará con el objeto en él. Sin embargo, al contrario que la modalidad anterior, es bastante fiable respecto a los cambios en iluminación y movimientos de cámara.

Dentro de este grupo existen múltiples algoritmos y métodos como explica Jared Willems [4], que pueden dividirse en dos subgrupos: las técnicas recursivas y no recursivas. Se diferencian fundamentalmente en la manera de obtener la imagen de fondo.

Antes de explicar los diferentes métodos de modelaje de fondo, es necesario explicar cuál es la idea detrás de ellas.

Cada píxel de la imagen sigue una distribución probabilística, es decir, si juntamos todos los valores que obtiene un píxel a lo largo de una escena y los representamos en una gráfica, se puede observar que este píxel tenderá a obtener ciertos valores.

La gráfica que obtenemos, la mayoría de las veces, será una campana de Gauss, esto es, una función de probabilidad Gaussiana con una media y una desviación típica. Esta media hacia la que tienden los valores del píxel será interpretada como parte del background, por tanto, si se adquieren valores alejados de esa media, significa que hay un objeto en movimiento y se deduce que será foreground. A su vez, debido a ciertos factores como pueden ser la luminosidad de la escena, cambio de mobiliario, o similares, esta función cambiará con el tiempo, cambiando su media y desviación típica.

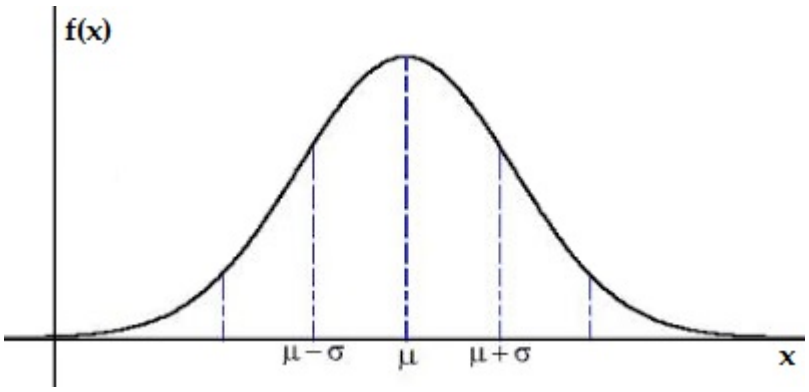

Figura 2.7: Campana de Gauss

Las técnicas siguientes tratan de adaptar este concepto, sin embargo, debido a la excesiva carga computacional y de memoria que resultaría hacer un histograma recursivo de cada píxel, cada una aborda este problema de forma diferente.

Para facilitar la compresión de las funciones que se verán a continuación, se definen ahora los elementos comunes a éstas:

I_t: Intensidad de luminosidad de un píxel a tiempo t.

B_t: Intensidad de luminosidad del background del píxel a tiempo t.

2.4.2.1. Técnicas no recursivas

Las técnicas no recursivas utilizan un número (N) de frames previos al actual, que son acumulados en un buffer y basándose en la variación temporal de cada píxel dentro del buffer realizan una estimación del segundo plano. Estos métodos son altamente adaptativos, ya que no dependen de los frames anteriores a los del buffer. Sin embargo, tienen un pequeño inconveniente. Si el tamaño del buffer es demasiado grande, necesitará mucha memoria para almacenar esos datos [13].

Diferencia de frames Es la técnica de substracción de fondo más simple. El background estimado es simplemente el frame anterior al actual. Se realiza la diferencia de estos dos frames y si los valores de los píxeles resultantes son mayores que un cierto umbral, el píxel se considera parte del fondo [14]:

$$|I_t - I_{t-1}| > T, \tag{2.1}$$

donde T es un valor fijado del umbral.

La umbralización que se aplica es un paso que se realiza en todas las técnicas de substracción de fondo, el cual se explicará con más detalle en la sección 5.4. Esta técnica es muy sensible al valor del umbral, ya que es el único factor que puede influenciar el resultado.

Este tipo de método no da resultados demasiado precisos y es muy sensible al ruido. Además, cuando el objeto o persona no se mueve durante más de un frame, el sistema no es capaz de detectarle, ya que pasa a formar parte del background. Sin embargo, tiene tres ventajas principales. Las primeras son su pequeña carga computacional y de memoria, ya que la única operación que se realiza es una resta de matrices y el buffer solo almacena un frame. La última es que el modelo de fondo es extremadamente adaptativo. Es decir, como el fondo se basa en un solo frame, se adapta a cambios en el entorno más rápido que cualquier otro método [4].

Filtrado con mediana (Median filtering) Es uno de los métodos de modelaje de fondo más usados. El valor del fondo estimado de cada píxel es calculado como la mediana en todos los valores del píxel en el buffer [13].

Esta técnica ofrece muy buenos resultados y, además, no requiere mucho coste computacional. La desventaja más notable aparece cuando el tamaño del buffer es muy grande, aumentando el requerimiento de memoria (N x tamaño del frame) y sobre todo, la carga de procesamiento.

El filtrado con mediana se puede extender a imágenes en color reemplazando la mediana por el "medoid" [13].

Filtro predictivo linear Esta técnica utiliza un filtro predictivo linear para calcular el segundo plano en los píxeles de los frames del buffer. Los coeficientes del filtro son estimados dependiendo de las covariancias de las muestras en cada frame, resultando difícil de aplicar en tiempo real [13].

Modelo no-paramétrico Como explican Elgammal et al. [15], este último método no recursivo, al contrario que las técnicas anteriores que usan un solo background para cada píxel, calcula el background mediante una función de densidad de píxel no paramétrica que usa todos los frames del buffer.

$$f(I_t) = \frac{1}{N} \sum_{i=1}^{N} K(I_t - I_i) \tag{2.2}$$

$K(\cdot)$ es el kernel estimador que sigue una distribución Gaussiana.

El píxel actual I_t es considerado parte del primer plano si $f(I_t) < T$, es decir, si es más pequeño que un umbral definido. La principal ventaja es la capacidad de manejar la distribución multimodal de fondo. Esto es, eventos como los píxeles de un árbol que se balancea, o bordes de alto contraste que "parpadean" ante ligeros movimientos de la cámara.

2.4.2.2. Técnicas recursivas

Las técnicas recursivas, al contrario de las no recursivas, no utilizan un buffer para el cálculo del fondo. En su lugar, actualizan de forma recursiva un único modelo de fondo basado en cada nuevo frame. De esta forma, frames pasados pueden tener un efecto sobre el fondo actual.

La ventaja principal es que solo se almacena un frame que actúa como background, que se actualiza cada vez que un nuevo frame es recibido, por lo que el gasto en memoria es mínimo. Sin embargo, si aparece algún error en el segundo plano, éste tarda mucho en desaparecer. esto significa que este tipo de modelaje de fondo es menos adaptativo que las técnicas no recursivas.

Media móvil (Running average o Running Gaussian average) Este método usa un algoritmo simple y rápido que no hace uso de grandes cantidades de memoria. Se basa en introducir una función de densidad de probabilidad gaussiana en los últimos valores de los píxeles:

$$B_t = \alpha I_{t-1} + (1 - \alpha) B_{t-1}, \tag{2.3}$$

donde α es el ratio de aprendizaje con valor típico de 0.05 y B_t es entendido como la media calculada de un píxel del background. Aparte, también es necesario calcular la desviación típica σ_t de B_t, que se calcula de forma similar [16]:

$$\sigma_t^2 = \alpha(I_t - B_t) + (1 - \alpha)\sigma_{t-1}^2, \tag{2.4}$$

Por último, se aplica una umbralización de valor $k\sigma_t$:

$$|I_t - B_t| > k\sigma_t \tag{2.5}$$

Si se supera ese valor, el píxel se considera primer plano. Si no, es parte del fondo.

Por tanto, al final solo se requieren dos parámetros por cada píxel (I_t, σ_t), ocupando menos espacio de memoria que los casos anteriores. Es una técnica que no da resultados muy precisos, pero depende de la aplicación y del valor de α puede ser usada con resultados aceptables [4].

Filtro de mediana aproximado (Approximated median filter) Es un tipo de modelado de fondo desarrollado en 1995 por McFarlane y Schofield para rastrear cerdos [17]. Funciona de la siguiente manera: si un píxel del frame actual tiene un valor mayor que su correspondiente en el segundo plano, este píxel es incrementado por 1. Si no, se decrementa en 1. De esta forma el background converge a valores donde la mitad de los píxeles de entrada son mayores que el background, y la otra mitad son menores, es decir, aproximadamente la mediana [14]. El tiempo de convergencia a estos valores dependerá del frame rate y la cantidad de movimiento en la escena.

$$B(x,y) = \begin{cases} B(x,y) + 1 & \text{si } I(x,y) > B(x,y) \\ B(x,y) - 1 & \text{si } I(x,y) > B(x,y) \end{cases} \tag{2.6}$$

Este método, a pesar de su sencillez, ofrece muy buenos resultados, además de necesitar de pocos requerimientos de memoria y ser una técnica robusta. La única desventaja es que tiene una baja adaptabilidad ante cambios en la escena.

Filtro Kalman El filtro Kalman es una técnica recursiva muy usada para rastrear sistemas lineales dinámicos con ruido gaussiano [13]. Esta técnica realiza una estimación del fondo mediante varios parámetros: la intensidad de luminosidad, su derivada temporal y/o sus derivadas espaciales. La versión más simple de este sistema usa solamente la intensidad luminosa.

Jared Willems muestra la más sencilla de estas variaciones, donde solo tiene en cuenta la intensidad de luminosidad [4] para calcular el fondo:

$$B_t = A \cdot B_{t-1} + K_t[I_t - H \cdot A \cdot B_{t-1}] \tag{2.7}$$

A es la matriz del sistema que describe la dinámica del segundo plano, H es la matriz de medida y K_t es la matriz de ganancia de Kalman. A y H son constantes, mientras que K_t es una variable que va alternando entre dos constantes de adaptación rápida o lenta, que cambian según si el píxel es un píxel de primer plano o de segundo plano.

Debido a su naturaleza, esta técnica presenta una alta adaptabilidad a cambios en el entorno. La parte negativa es que deja largos rastros detrás de objetos en movimiento.

Mezcla de Gaussianos (MoG) La mezcla de Gaussianos (Mixture of Gaussians en inglés), al contrario que el filtro Kalman que solo comprueba la evolución de un solo Gaussiano, trabaja con varios a la vez. Como se explica al principio de esta sección, cada píxel tiende a obtener una serie de valores que forman parte del fondo de la escena, que representados en un histograma formarán una distribución Gaussiana, en la mayoría de casos. Sin embargo, no siempre es así y los valores pueden oscilar entre varios valores distantes. Un ejemplo típico sería una escena en el exterior donde hay varios árboles enfrente de un edificio. Un mismo píxel oscilará entre diferentes valores: hojas de árbol, ramas de árbol, y el edificio. Otros ejemplos serían escenas con nieve o lluvia, u olas en una playa. En estos casos no sería posible usar un solo modelo de fondo [18].

Al contrario que las demás técnicas, el modelo de fondo creado no es una imagen de valores, sino que es paramétrico. Es decir, cada píxel es representado por un número determinado de funciones Gaussianas, que al sumarlas forman una función de distribución de probabilidad [14]. Esta fórmula es:

$$f(I_t) = \sum_{i=1}^{k} \omega_{i,t} \cdot \eta(\mu_{i,t}, \sigma_{i,t}), \tag{2.8}$$

donde k representa el número de Gaussianos utilizados (normalmente de 3 a 5), $\eta(\mu_i, \sigma_i)$ es el i-ésimo componente Gaussiano con media $\mu_{i,t}$ y desviación típica $\sigma_{i,t}$, y $\omega_{i,t}$ es el peso asociado a cada Gaussiano. El peso y la desviación estándar son medidas de confianza (más peso y menor desviación estándar significan mayor confianza).

Para comprobar si un píxel I_t forma parte del primer o segundo plano, primero hay que identificar el componente $\hat{i}$ cuya media se acerque más a este valor. Cuando se ha encontrado, el siguiente paso será obtener la diferencia entre el valor del píxel y la media del Gaussiano $\hat{i}$. Si el valor absoluto de este resultado es menor que la desviación estándar de $\hat{i}$ multiplicado por un

factor D, el píxel forma parte del background, si no será parte del primer plano:

$$|I_t - \mu_{\hat{i},t-1}| \leq D\sigma_{\hat{i},t-1} \tag{2.9}$$

Después de cada frame, las variables ω, μ, y σ deben ser actualizadas, mediante unas ecuaciones que explican detalladamente S.-C. Cheung y C. Kamath en [13]. Esta técnica ha ganado mucha popularidad, debido a su precisión, su capacidad de manejar escenarios multimodales y a sus pocos requerimientos de memoria. Sin embargo, es muy sensible a cambios repentinos en la iluminación y es considerado un método complejo [4].

A continuación, se muestra un esquema de los métodos de substracción de fondo estudiados.

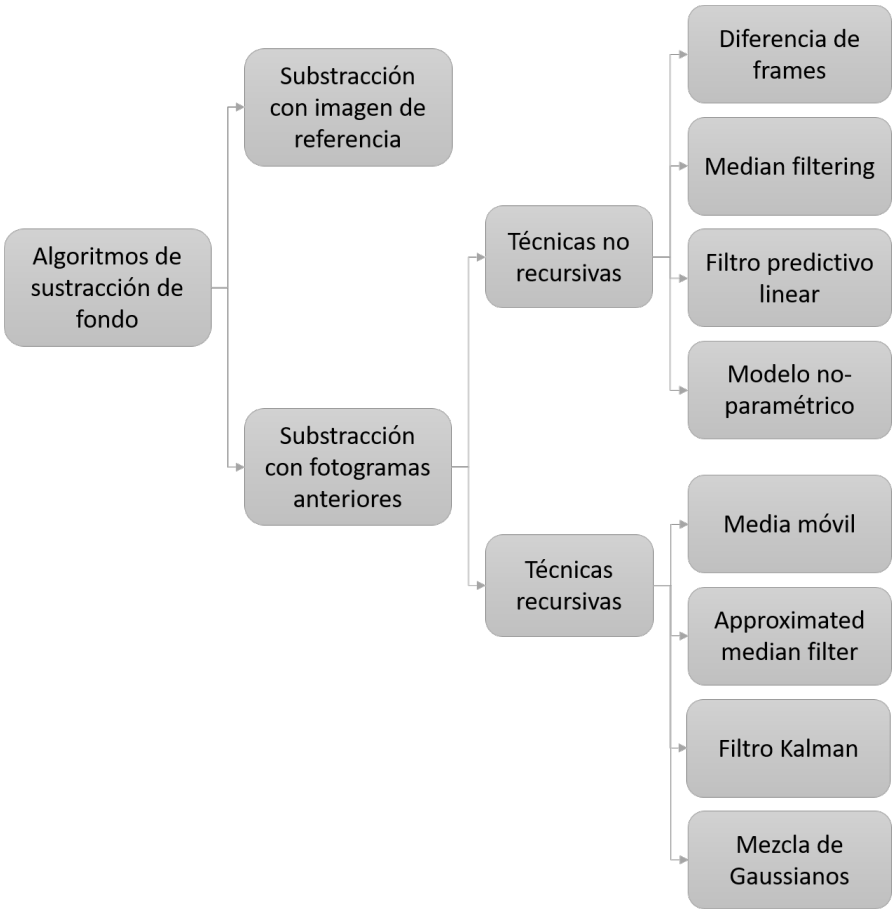

Figura 2.8: Esquema general de los algoritmos de substracción de fondo

Después de estudiar las diferentes técnicas existentes, en este proyecto se trabajará con el filtro con mediana y el filtro con mediana aproximado para el desarrollo del código, debido principalmente a su relativa baja complejidad, al bajo coste computacional que ofrecen, y a su fácil manejo.

2.5. Detección de caída

En esta sección se discutirá la etapa de detección de caída del sistema desarrollado.

Después de haber creado una máscara de detección de movimiento gracias a las técnicas de sustracción de fondo, y haber aplicado varios ajustes, como se explicará en el capítulo 5,el siguiente paso será manejar esa información para poder determinar cuándo se ha producido la caída. Existen diferentes acercamientos a este problema. Un primer acercamiento son los algoritmos basados en los parámetros visuales, es decir, aquellos que usan los datos directamente de las imágenes entrantes y máscaras creadas. Algunos ejemplos de estos parámetros pueden ser los cambios repentinos en las dimensiones de la persona o diferencias entre máscaras. Un segundo método serían los algoritmos de aprendizaje (o Machine Learning, en inglés), como los Modelos Ocultos de Márkov (HMMs) o las Máquinas de vectores de soporte (SVMs). Estos son programas que automatizan comportamientos a partir de un entrenamiento a base de ejemplos. Sin embargo, como explican Adam Williams et al. [19] tienen una desventaja importante. Aunque son eficaces, requieren de un entrenamiento computacionalmente intensivo y material con el que entrenar. El sistema con el que se trabajará no dispone de estos recursos y conseguir ese tipo de material es complicado. Por ello, en la memoria se centrará en estudiar la primera opción. A continuación, se mostrarán cinco métodos de detección de caídas de este tipo.

Una de las técnicas más simples y más usadas es el uso de la relación de aspecto (aspect ratio) [19], o lo que es lo mismo, el ancho de la persona dividido entre su altura. Es una buena forma de determinar su posición, es decir, si se encuentra de pie o en una posición horizontal. Si el aspect ratio del rectángulo que contiene a la persona es mayor que un cierto umbral $\alpha < 1$, la persona probablemente esté tumbada y puede que haya caído.

Una segunda opción consiste en calcular los gradientes verticales y horizontales del objeto [20]. Cuando la persona camina o está de pie, su gradiente horizontal es menor que su vertical, mientras que cuando se está cayendo, sucede al contrario.

Otra técnica se basa en el uso del ángulo de caída [4, 20]. El ángulo de caída es el ángulo del centroide del objeto con respecto al eje horizontal del rectángulo que lo contiene, dicho de otra forma, es el ángulo establecido entre el suelo y la persona desde donde la persona cae. El centroide se refiere al centro de masas de coordenadas de un objeto.

La cuarta alternativa de detección de caídas utiliza histogramas de proyección vertical [4]. El histograma de proyección vertical de una persona cambiará significativamente cuando ocurra una caída. Cogiendo los valores máximos de los histogramas y comparándolos se puede establecer si se ha producido la caída o no. Los histogramas de proyección vertical se calculan se esta manera:

$$H(x,y) = \begin{cases} 1 & \text{si (x,y) es un píxel del objeto} \\ 0 & \text{si no} \end{cases} \tag{2.10}$$

$$V(x) = \sum_{y} H(x,y) \tag{2.11}$$

El último método estudiado usa el centroide del objeto. Cuando se produce la caída, el centroide cambia de posición rápidamente [4]. Determinando un tiempo específico de caída, se puede saber si el cambio ha sido producido por una caída o por otra razón.

No existe una única forma de detectar una caída a través de vídeo. Lo que puede funcionar para un tipo de caídas, puede no servir para otras. Las opciones más eficientes deberán hacer uso de varias de estas técnicas para asegurar una buena detección.

La fase de detección de caída tiene dos etapas diferentes. La primera consiste en la misma detección de la caída, y la segunda consiste en la confirmación. El sistema puede detectar una actividad anómala que asociará a una caída, pero es necesario una segunda etapa que ayude a descartar posibles detecciones falsas. Como se verá en la sección 5.9, la aplicación usará el aspect ratio y el ángulo de caída como métodos de la primera etapa, utilizando histogramas de proyección vertical para la confirmación.

Capítulo 3

Arquitectura Hardware y Software

El desarrollo de un buen algoritmo de detección es la parte fundamental para que la aplicación funcione. El estudio de las diferentes técnicas nos servirán para crear una buena aplicación de detecciones, sin embargo, de nada servirá si no se elige un sistema óptimo que pueda ejecutarlo. Además, ya que la idea del proyecto es crear un producto destinado a un uso comercial, habrá que establecer un balance en la calidad y precio de los componentes.

A continuación, se puede ver un esquema de los componentes del sistema (figura 3.1) y una foto de éste (figura 3.2).

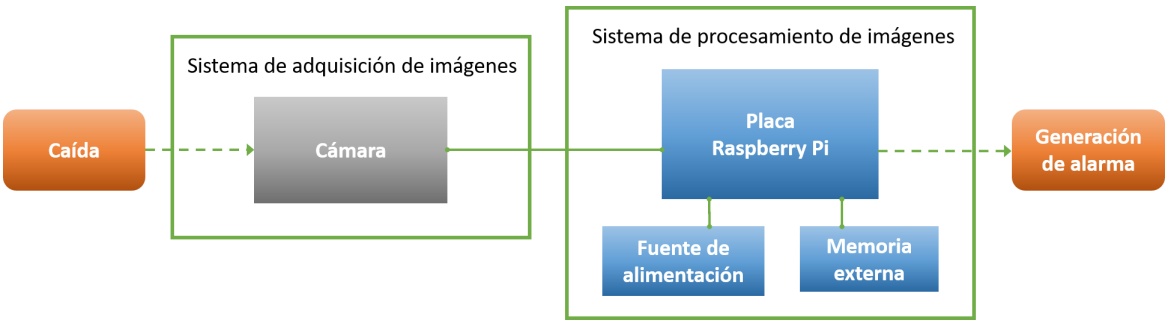

Figura 3.1: Esquema de los componentes hardware

Figura 3.2: Foto de los componentes del dispositivo

En esta sección se detallarán los elementos hardware y software utilizados para la realización del proyecto.

3.1. Software

3.1.1. Sistema operativo

La aplicación se desarrolla sobre el sistema operativo Raspbian, en concreto, la versión 2017-04-10-Raspbian-Jessie. Raspbian es una distribución del sistema operativo GNU/Linux, libre y gratuito, basado en Debian y optimizado para el hardware de la placa Raspberry Pi.

Raspbian es el sistema operativo principal que usa la Raspberry Pi. Incluye de serie una colección de 35000 paquetes precompilados entre los que se encuentran programas como Scratch, Java, Mathematica y Python [21]. La distribución incluye también un entorno de escritorio y navegador web. Debido a sus características y a las posibilidades que ofrece, la Raspberry Pi se convierte en una alternativa a las placas con microcontrolador clásicas, además de cubrir gran parte de las funcionalidades básicas de un ordenador personal.

El desarrollo de la aplicación y los experimentos iniciales, se probaron en un ordenador personal bajo el S.O. de Windows. El salto entre estos sistemas operativos no supone ningún inconveniente, ya que, salvo algunas diferencias obvias en el uso del Hardware y versiones, al utilizar Python no aparece ningún tipo de error.

3.1.2. Lenguaje de programación

El lenguaje de programación que se utilizará para escribir el código será Python. Este lenguaje tiene la ventaja respecto a otros en la sencillez y flexibilidad de su sintaxis, lo que lo hace por otra parte fácil de aprender. Otro punto a favor reside

en su portabilidad, es decir, es muy fácil portear código entre diferentes S.O. La versión que se utilizará en la placa será la 2.7.9.

3.1.3. Bibliotecas

3.1.3.1. OpenCV 3.1.0

Es la biblioteca principal que se usará para el tratamiento de las imágenes. OpenCV (Open Source Computer Vision Library) es una biblioteca libre destinada al procesamiento de imágenes y visión computarizada, desarrollada originalmente por Intel. Está escrita y optimizada en los lenguajes C y C++. Tiene interfaces de C, C++, Python y Java, y es compatible con Windows, Linux, Mac OS, iOS y Andorid [22].

Esta biblioteca permite visualizar datos de forma sencilla y extraer información de imágenes y vídeos gracias a las múltiples funciones que contiene. Ha sido diseñada para ser eficiente computacionalmente y con un enfoque a las aplicaciones en tiempo real. Uno de los objetivos principales es el de proveer una infraestructura de visión artificial de fácil manejo que permite realizar aplicaciones "sofisticadas" de una manera sencilla y rápida. Se usará la versión 3.1.0.

3.1.3.2. Otras librerías

Aparte de OpenCV, para poder manejar los datos obtenidos es necesario usar otras bibliotecas.

- **NumPy**: Es el paquete fundamental de computación científica para Python. Esta biblioteca le añade soporte para vectores y matrices multidimensionales y funciones de alto nivel matemático para trabajar con ellos.
- **Matplotlib**: Es la biblioteca de generación de gráficos para Python y su extensión Numpy.
- **Smtplib**: Paquete necesario para establecer la conexión con el servidor tipo SMTP, que permite enviar y leer correos electrónicos.
- **Email**: Sirve para crear mensajes tipo MIME complejos para el envío de correos electrónicos.

3.2. Hardware

3.2.1. Raspberry Pi 3 Model B

La idea principal detrás del proyecto es la de diseñar un sistema compacto y barato, es decir, crear un dispositivo que no necesite de grandes cantidades de potencia computacional, que pueda usarse de forma sencilla y a un precio asequible. Una de las mejores opciones que se pueden encontrar en la actualidad es la placa Raspberry Pi.

La Raspberry Pi es un computador de placa reducida (SBC). Es según la propia página web del producto [23], "un ordenador de tamaño de tarjeta de crédito que se conecta al televisor y teclado". Esta placa admite todo tipo de periféricos necesarios de un ordenador común, como ratón, cámara, pantalla, etc. La ventaja principal respecto a demás ordenadores convencionales es su reducido tamaño y precio. Otra ventaja importante es que es open source, por lo que permite instalar distintos S.O. y controlar aspectos hardware y software internos del dispositivo. Además, cuenta con varios pines genéricos (GPIO) controlables, lo que la hace idónea para aplicaciones de índole ingenieril, supliendo funciones de las placas microcontrolador clásicas.

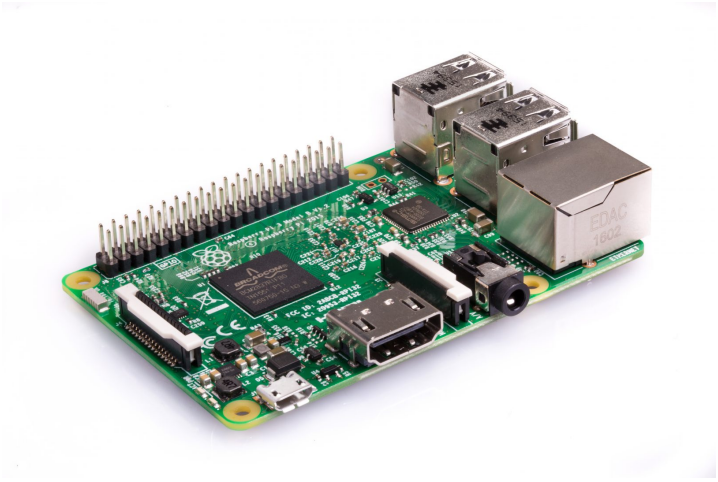

Figura 3.3: Placa Raspberry Pi 3 Model B

Las placas Raspberry Pi cuentan con diferentes modelos. Se ha optado por el modelo Raspberry Pi 3 Model B. El modelo B es más completo y tiene una memoria RAM superior que el A. En particular, el Raspberry Pi 3 Model B es la tercera generación del Raspberry Pi. Reemplazó al Raspberry Pi 2 Model B en febrero de 2016. Tiene las siguientes características [23]:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port

- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

Al precio inicial del dispositivo hay que añadirle una fuente de alimentación con conector micro-USB de 2.1A de corriente, y una tarjeta micro SD con capacidad suficiente para instalar el sistema operativo y demás programas. Además, es conveniente incluir una carcasa que la proteja de agentes externos.

3.2.2. Cámara Webcam Selecline 862050

El éxito final de un algoritmo de visión por computador depende en gran medida de la calidad de la imagen con la que se trabaja. Por ello es conveniente contar con el mejor equipo de grabación posible.

En este caso, debido a las limitaciones de presupuesto al querer hacer un sistema económico, no es posible adquirir la mejor cámara del mercado. Además, tampoco es necesario, ya que, como se verá más adelante, en el algoritmo se reducirá la resolución de las imágenes a favor de una carga de procesado menor.

La cámara escogida para el proyecto ha sido la cámara web Selecline 862050. Las ventajas principales son su calidad aceptable a un bajo precio y la característica Plug and Play, que permite su uso sin necesidad de instalación. Esta cámara ofrece las siguientes especificaciones:

- Resolución de vídeo: 640x480
- Fotos: 0.3 megapíxeles
- Conexión USB 2.0 (Plug and Play)
- Modo de vídeo VGA

Figura 3.4: Cámara Webcam Selecline 862050

Capítulo 4

Elección de escenas

Con el objetivo de experimentar con el código desarrollado, será necesario tener una colección de vídeos adecuados que permitan hacer las pruebas. La elección del material es crucial para el desarrollo de la aplicación, ya que una mala selección puede influir en los resultados, descartando código correcto o haciendo lo contrario. Además, es importante tener contenido variado, con el fin de probar el código en diferentes escenarios y situaciones.

En un principio, ante la falta de material, se grabaron algunas escenas, a modo de iniciación en la materia. Éstas ayudaron a comprender los fundamentos de la visión artificial y a experimentar con el código, pero eran escasos y no servían para hacer las pruebas. Finalmente se optó por buscar un dataset de vídeos de caídas en internet que sirvieran para ello.

4.1. Primera experimentación

Se encontró una colección de vídeos desarrollados en el "Interdisciplinary Centre for Computational Modelling University of Rzeszow" [24] que contenían una buena cantidad y variedad de vídeos de caídas. Este dataset contiene una colección de 70 vídeos siendo 30 vídeos de caídas y 40 de actividades cotidianas. Las caídas eran recogidas con cámaras Microsoft Kinect y sistemas de acelerómetros, pero para el proyecto servían solamente las imágenes grabadas por las cámaras. Éstas presentan una resolución de 320x240 píxeles y un frame rate de 30 frames/s, y recogen imágenes RGB y de profundidad. Están situadas en paralelo con el suelo, a unos 1,50 metros aproximadamente y recogen diferentes entornos de grabación y situaciones con diferentes personas y vestimentas. Esta colección se usó en la primera mitad de la memoria.

Esta colección servía para los experimentos, pero presentaba algunos problemas. Por un lado, los vídeos son demasiado cortos, por lo que es difícil observar cómo evoluciona el fondo a lo largo de estos. Por otro lado, estos vídeos sirven para la detección de caídas, pero están destinados al uso de otro tipo de técnicas de detección. Nuestra aplicación, como ya se ha explicado, usará la substracción de fondo para detectar el movimiento. Esto supone que es necesario construir un buen fondo inicial de la escena. Construir un background desde el primer frame

supondría considerar que el primer frame constituye el fondo, cuando no siempre es así. Hay ocasiones en las que el vídeo empieza con la persona ya en escena, haciendo imposible construir un background correcto. Desafortunadamente, la mayoría empiezan con la persona en el primer frame, por lo que la solución pasa por encontrar un frame donde no aparezca ningún objeto en escena para usarlo como fondo inicial, o construir uno a partir de dos o más frames, como muestra la figura 4.1.

Figura 4.1: Construcción manual del fondo inicial

Estas construcciones del background inicial a partir de varios frames se han hecho en los vídeos donde la persona no desaparece del escenario en ningún momento. Sirven para su propósito, pero presentan varios fallos, ya que son construidos manualmente. Además, muchos han sido creados a partir de frames de diferentes vídeos del mismo escenario, por lo que tendrán zonas con distinta iluminación y sombra, dando errores en los experimentos y alterando los resultados. Por último, aunque los vídeos contienen diferentes escenarios y actividades, los vídeos donde se recogen las caídas suceden en un único escenario. Aparte, este entorno de grabación es exclusivo para las caídas, y no aparece en los demás vídeos de actividades cotidianas, haciendo complicada la comparación de resultados.

4.2. Dataset final

Ante esta situación, se optó en una segunda etapa de experimentación por buscar otro dataset de videos, que solucionaran todos estos problemas. El escogido fue el proporcionado por el laboratorio "LE2I Laboratoire d'Electronique, Informatique et Image" [25]. Este dataset contiene 221 vídeos con distintos escenarios y actividades más variadas que la primera colección. La resolución de estos es de 320x240, con un frame rate de 25 frames/s. Para poder realizar los experimentos adecuadamente se ha hecho una clasificación en la que se detallan las actividades y tipos de caídas realizadas, separando cada vídeo por tipo de escenario, iluminación, tipo de vestimenta, si hay o no oclusiones, y si empieza o no con la escena vacía. Este último punto es importante, ya que, como se ha comentado antes, los que tengan el frame inicial ocupado darán problemas en los resultados.

En todos los vídeos de la colección, la cámara se encuentra a una distancia aproximada de 2.50 metros del nivel del suelo, de forma que la habitación es capturada totalmente desde una posición elevada. Existen 8 vídeos donde la sala es grabada desde una altura de unos 1.50 metros aproximados en paralelo al suelo, sin embargo, graban caídas después de bajar escaleras, con oclusiones y el fondo inicial

ocupado, por lo que serán descartados. El emplazamiento del material de grabación influirá a la hora de determinar los parámetros de detección de caída, ya que la silueta de los objetos variará dependiendo desde donde se grabe. De esta forma la información obtenida deberá ser tratada de forma diferente en cada ocasión. Lo ideal sería poder trabajar con distintas ubicaciones de la cámara para ayudar a encontrar la opción más óptima de cara a la detección de caídas, pero debido a la falta de material, el código se desarrollará teniendo en mente la colocación de cámaras en esta posición.

Figura 4.2: Escenarios del dataset final

Para conseguir una buena detección, el sistema más adecuado consistiría en la colocación más de una cámara en la habitación en diferentes ubicaciones para obtener una detección más eficiente [19], pero esta memoria se centrará en la detección usando una sola.

A continuación, se muestran tablas donde se recoge de forma resumida estas clasificaciones. La tabla 4.1 muestra la cantidad de vídeos en los que el frame inicial muestra la escena vacía u ocupada. Las tablas siguientes muestran datos más específicos separándolos según la característica anterior. La tabla 4.5 contiene los diferentes tipos de caídas recogidas.

Estado frame inicial	**nº videos**
Vacío	83
Ocupado	138

Tabla 4.1: Estado del frame inicial

	Frame inicial vacío	Frame inicial ocupado	Total
Luz natural	57	77	134
Luz artificial	25	2	27
Luz natural y artificial	0	54	54
Luz natural y artificial (baja iluminación)	1	5	6

Tabla 4.2: Tipos de iluminación

	Frame inicial vacío	Frame inicial ocupado	Total
Oclusiones	38	64	102
Sin oclusiones	45	74	119

Tabla 4.3: Vídeos con oclusiones

	Frame inicial vacío	Frame inicial ocupado	Total
Caídas	31	99	130
Sin caídas	52	39	91

Tabla 4.4: Vídeos con caídas y actividades cotidianas

Tipos de caídas y movimientos:

CL – Caída lateral

CF - Caída frontal

CA - Caída hacia atrás

LS – Se levanta de silla/sofá

SS – Se sienta en silla/sofá

G – Giro

AS – Se apoya en silla

BE – Baja escaleras

MP – Mueve puerta

MO – Mueve objeto

	Frame inicial vacío	**Frame inicial ocupado**	**Total**
CL	8	12	20
CF	10	39	49
CA	1	10	13
LS + CL	0	16	16
LS + CF	2	9	11
SS + CA	0	1	1
G + CF	0	2	2
AS + CF	1	0	1
BE + CF	0	8	8
MP + CF	2	0	2
MP + CA	1	0	1
MO + CA	2	0	2
MO + CF	3	0	3
MP + LS + CF	1	0	1

Tabla 4.5: Tipos de caídas

La colección de vídeo escogida recoge un buen número de situaciones y entornos. Estas grabaciones, sin embargo, no representan caídas reales de personas mayores. Los resultados obtenidos servirán para tener una buena base de cara a las detecciones de caídas, pero es conveniente realizar una segunda experimentación usando un dataset más realista.

Capítulo 5

Desarrollo del algoritmo

Una vez se ha hecho un repaso de las técnicas y se ha conseguido el material necesario para realizar las pruebas, el siguiente paso será empezar a crear el algoritmo del sistema. En este capítulo se explicará y detallará cada etapa del algoritmo desarrollado. En la figura 5.1 se muestra un esquema general de su funcionamiento.

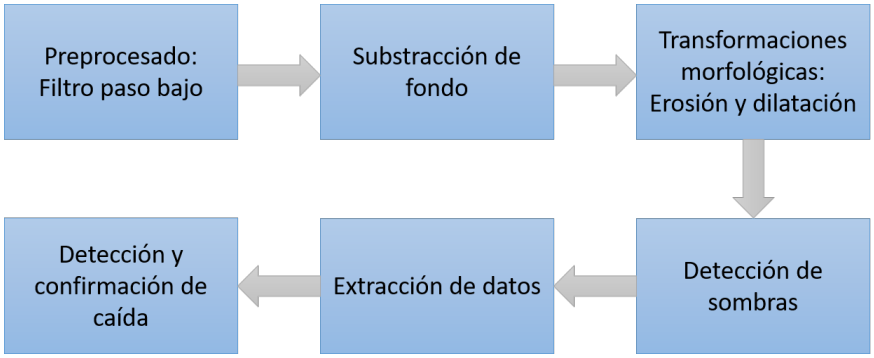

Figura 5.1: Esquema general del algoritmo

5.1. Preparativos iniciales

Una vez se han obtenido las imágenes, el primer paso será aplicar una serie de modificaciones para que el algoritmo funcione correctamente.

5.1.1. Conversión del tamaño de los frames

Antes de aplicar el filtro es necesario darle una resolución específica al vídeo. Este paso, se podría omitir, pero es algo muy útil para unificar los parámetros del algoritmo y reducir memoria y procesamiento.

A la hora de hacer pruebas con vídeos grabados o con cámaras, las resoluciones de éstas no son siempre las mismas. Esto es un problema, ya que en la parte de detección de figuras se toman en cuenta tamaños que dependen del número de píxeles. Es decir, si para una resolución de por ejemplo 320x240 se decide descartar contornos de área menor de 1000, para una resolución de 640x480 este área deberá

ser el cuádruple de grande.

Aparte, al trabajar con imágenes de mucha resolución, es necesario hacer uso de grandes cantidades de memoria, cuando en la mayoría de casos no hace falta tener tanto detalle de la escena. Por consiguiente, el tiempo computacional de las operaciones del algoritmo será mucho mayor.

En el código desarrollado, se ha estipulado una resolución de 320x240.

5.1.2. Conversión a escala de grises

Las imágenes recibidas son imágenes en formato RGB. Esto es, cada imagen es recogida como una matriz de valores de tamaño N x M (ancho x alto) los cuales representan el color de cada píxel. Cada píxel a su vez es un array de tres valores diferentes, que representan la intensidad de rojo, verde, y azul, respectivamente. Trabajar con estos datos supone un alto coste computacional y de memoria, ya que se trabaja con matrices de 3 dimensiones [26].

Para solucionar este hecho y simplificar las operaciones, es conveniente convertir los datos a escala de grises. De esta forma cada imagen será una imagen de 2 dimensiones.

Para realizar este paso, se hará uso de las funciones que ya incluye OpenCV.

5.2. Filtro paso bajo

La primera etapa del preprocesado será aplicar un filtro paso bajo. De esta forma, se minimiza el ruido provocado por la cámara y los cambios en la iluminación. Además, permite reducir el valor de umbral que se aplicará más adelante. Este paso se conoce también como suavizado.

Este filtro reduce los detalles y "desdibuja" los bordes de la imagen. Las técnicas estudiadas solo necesitan las dimensiones y orientación de los objetos en movimiento, por lo que así eliminamos información innecesaria y evitamos falsos positivos.

Para comprender como funciona el filtro es necesario saber que es una convulsión y un kernel. Un kernel (o plantilla) es una matriz de tamaño fijo de coeficientes numéricos con un punto de ancla situado normalmente en el centro.

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figura 5.2: Ejemplo de 5x5 Kernel Gaussiano

Una convolución es una operación entre cada parte de la imagen y un operador (kernel). Funciona de la siguiente manera. Se coloca el ancla del kernel en la localización de un determinado píxel, con el resto del kernel cubriendo los píxeles correspondientes de la imagen. Se multiplican los coeficientes del kernel por los correspondientes valores de píxel de la imagen y se suman los resultados. Se coloca el resultado en la posición del ancla de la imagen en una nueva imagen, y por último, se repite el proceso para todos los píxeles de la imagen.

El uso del kernel también se verá más adelante en operaciones posteriores (filtro de mediana, erosión, dilatación, etc.).

En el código desarrollado se utiliza un filtro Gaussiano para suavizar la imagen. Es decir, la imagen es convulcionada con un kernel Gaussiano mediante la función *cv2.GaussianBlur()*, proporcionada por OpenCV.

En la figura 5.3 se muestra una comparación de las máscaras de foreground creadas con diferentes tamaños de kernel, con un umbral de 20.

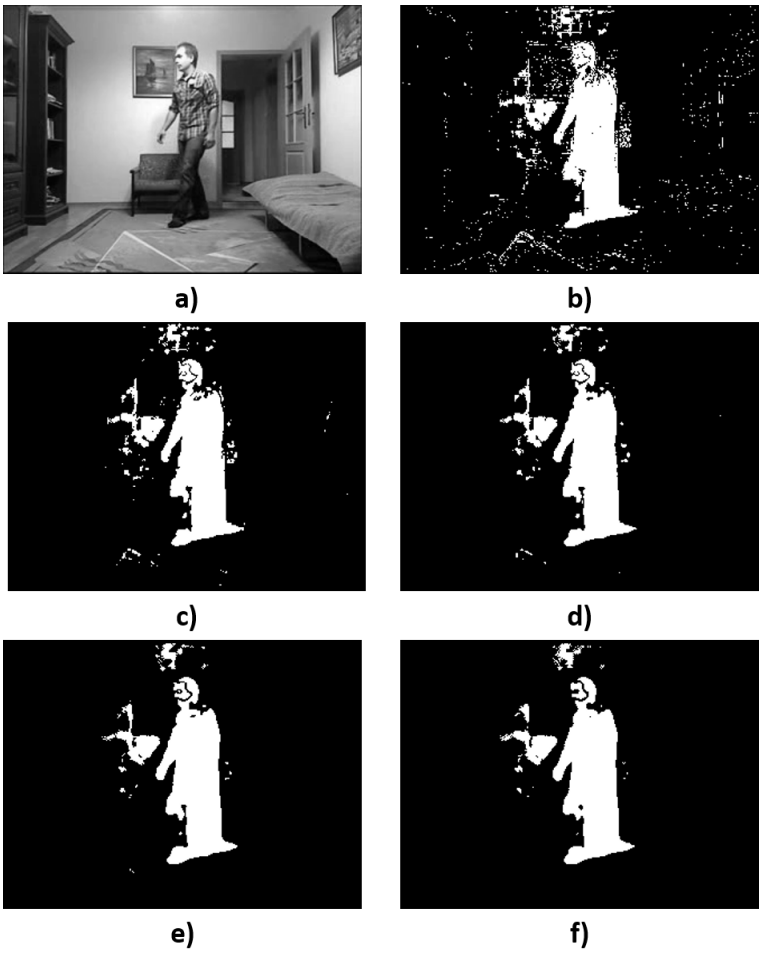

Figura 5.3: Resultado del suavizado
a) Input frame, b) sin blur, c) blur 3x3, d) blur 5x5, e) blur 7x7, f) blur 9x9

Se ha establecido un tamaño de 5x5 con una desviación estándar de 0 para el

kernel Gaussiano.

5.3. Substracción de fondo

La substracción de fondo (o background subtraction, en inglés) es el paso más importante a la hora de desarrollar el código. La correcta elección de la técnica es fundamental para que la detección de personas se pueda conseguir. Como adelantamos al final de la sección 2.4, los métodos desarrollados han sido el filtro con mediana y el filtro con mediana aproximado. Estas técnicas, comparadas con las demás estudiadas, presentan un bajo coste computacional y de memoria, aparte de resultar menos complejas en general.

Para elegir cual será el que se usará en la versión final, se harán varias pruebas con cada uno, y se escogerá el que muestre mejores resultados. En las secciones siguientes se explicará más detalladamente el funcionamiento de estos métodos y como se han implementado en el código.

5.3.1. Filtro con mediana

Esta técnica no recursiva calcula la imagen de fondo con la mediana de cada píxel contenido en los frames del buffer. La idea principal es que el valor del píxel se mantiene en el background por más de la mitad de los frames del buffer [13].

Este método presenta ventajas computacionales respecto a las demás técnicas no recursivas, sin embargo, si se quiere obtener una buena estimación del fondo, este aspecto se resiente en algunas ocasiones. Esto es debido principalmente al tamaño de buffer que queramos establecer. Trabajar con un buffer grande garantiza un buen modelo de fondo, pero implica trabajar con datos de grandes cantidades (N x tamaño del frame), por lo que es necesario usar mucha memoria. Esta técnica ordena todos los píxeles de todos los frames del buffer, por lo que, a más tamaño, mayor carga de procesamiento.

Para suavizar este problema, una solución es actualizar el modelo de fondo cada n_{max} número de frames [4]. Esto permite aligerar carga de procesamiento, ya que evita tener que realizar las operaciones a cada ciclo y las limita a cada cierto tiempo. Aparte, de esta forma se puede ampliar el tamaño del buffer. Por consiguiente, el buffer también se actualiza cada n_{max} número de frames.

Por último, como en todas las técnicas de substracción de fondo, es necesario calcular los valores absolutos de la resta entre el frame actual y el modelo de fondo, para después aplicarle una umbralización de un valor escogido.

En la figura 5.4 se muestra un diagrama donde se pueden observar estos pasos:

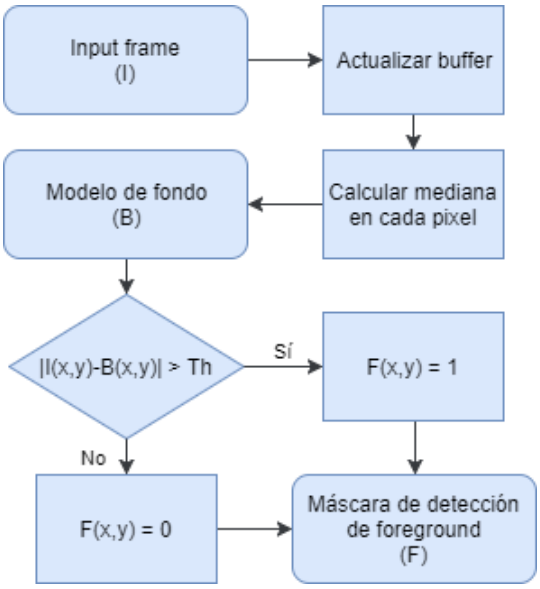

Figura 5.4: Diagrama de flujo del filtro con mediana

A la hora de escribir el código, sin embargo, no es tan sencillo de desarrollar. Python, y en particular Numpy, no trabaja bien con las iteraciones (comparado con otros lenguajes de programación), y es recomendable vectorizar el código lo máximo posible, y usar las funciones que traen las bibliotecas de serie. Es por ello que realizar cada operación de forma iterada en cada píxel supondría un altísimo coste de procesamiento y sería inviable de desarrollar.

Al principio se usaron listas para la ordenación y actualización del buffer, pero más tarde se descartó esta idea, ya que se comprobó que era más óptimo trabajar con arrays de Numpy directamente. Aparte, se estableció que era mejor hacer "alias" de los frames (esto es, igualarlos directamente mediante el símbolo "=" en Python) a la hora de hacer ciertas operaciones, que hacer copias, ya que resultan operaciones costosas. A continuación, se explica cómo se implementan los pasos principales del código.

- **Inicialización del buffer**: El buffer es inicializado como un array vacío de tamaño *buffSize* x height x width, de tipo uint8. Al coger el primer frame, todos los frames del buffer son igualados a este. Esto supone que durante los primeros segundos de la aplicación el background será el mismo que el primer frame.

- **Actualización del buffer**: Rotamos los frames del buffer una posición mediante la función *numpy.roll()* sobre el eje 0. Después igualamos el primer frame del buffer al frame actual.

- **Actualización del background**: Para poder calcular la mediana, es necesario, primero ordenar todos los píxeles de los frames del buffer. Para ello, hacemos uso de la función *numpy.sort()* que proporciona Numpy y le indicamos que haga la ordenación sobre el eje 0. Después cogemos la matriz que se encuentre en la posición central y esta será el modelo de fondo.

Los parámetros que hay que modificar en este caso serán:

- **Tamaño del buffer** (*buffSize*)

- **Número de frames por actualización** (n_{max})

5.3.2. Filtro con mediana aproximada

El filtrado con mediana aproximado, al contrario que el filtrado con mediana, es un tipo de técnica recursiva. Esto significa que no hará uso de ningún buffer, por lo que se puede avanzar una reducción significativa en el uso de los recursos de memoria. La filosofía detrás de esta técnica es la misma que la anterior, es decir el background será el resultado de encontrar la mediana en cada píxel de los frames recibidos. Sin embargo, al no usar un buffer se calcula de forma distinta.

Como se explica en el apartado 2.4.2.2, el fondo se calcula mediante comparaciones, es decir, si el valor de un píxel del frame actual es mayor que el del background, a este píxel de fondo se le suma un 1. Si es al contrario, se le resta 1. Este tipo de modelaje de fondo no requiere de mucha memoria (solo almacena el frame del background), y además no tiene mucho más coste computacional que la diferencia de frames [14].

El valor que se añade o se resta, que se llamará a partir de ahora en la memoria el número de cómputo (*compNum*) puede ser cambiado para cambiar la velocidad de adaptación del background. Esto significa que cuanto mayor sea este número, más rápido se actualizará el fondo. Por otro lado, esta adaptabilidad también dependerá del frame rate del vídeo. Las comparaciones se hacen una vez por cada frame, por lo que, a mayor frame rate, mayor velocidad de actualización.

Por último, se aplica la umbralización a la diferencia entre el primer y segundo plano.

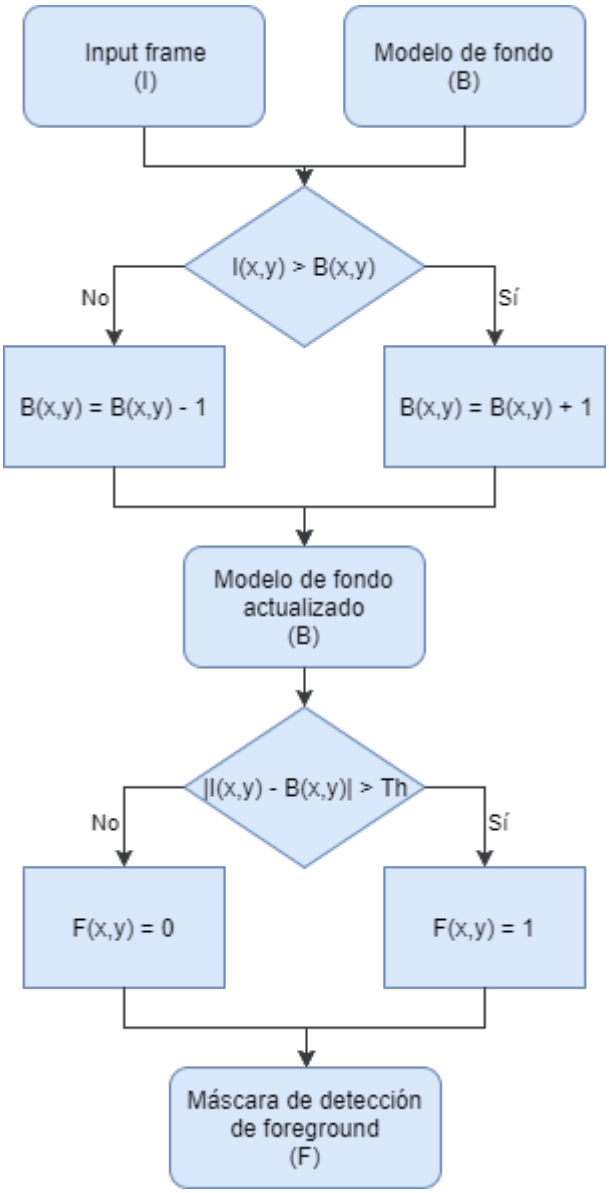

Figura 5.5: Diagrama de flujo del filtro con mediana aproximada

Como se argumenta en el punto anterior, Python no trabaja bien con los cálculos iterativos. Este tipo de algoritmo sería fácil de implementar mediante el uso de un loop que comparase uno por uno los píxeles del frame, pero sería un proceso muy costoso. Por esta razón se ha desarrollado el código de la siguiente manera:

- Con la función *cv2.compare()* que incluye OpenCV, se crean dos máscaras mediante la comparación del frame actual con el fondo. En una estarán los píxeles con valores mayores a los del fondo (*maskCompG*) y en la otra los valores menores (*maskComp*).
- Después, con la función *cv2.add()* y *cv2.substract()* se resta y se suma, respectivamente, el valor de cómputo elegido aplicando las correspondientes máscaras previamente calculadas. Estas funciones impiden que haya desbordamientos.

- Por último, ya que como se verá en las experimentaciones se pueden usar valores decimales para hacer las restas y sumas, creamos un segundo background con valores tipo float, que será el que se usará para hacer los cálculos comparativos, dejando el background con valores enteros para hacer la diferencia con el frame actual.

El único parámetro que habrá que modificar esta vez será:

- **Número de cómputo** (*compNum*)

Después de analizar cada método, el que se escogerá para desarrollar la aplicación sera el filtro con mediana. La comparación de ambas técnicas se encuentra en el capítulo 6.1

5.4. Detección del segundo plano y umbralización

La siguiente etapa consistirá en extraer el foreground de la escena. La detección del segundo plano se realiza con la comparación entre el modelo de fondo generado y el frame entrante. Es decir, se restan las dos imágenes en valores absolutos. Exceptuando el modelo no-paramétrico y la Mezcla de Gaussianos, todos los métodos explicados en el punto 2.4 crean una sola imagen de fondo.

Después llega el momento de la umbralización. A lo largo de la memoria se ha hablado sobre esta operación, pero no se ha detallado su funcionamiento. Su explicación es simple. Si el valor de un píxel de la imagen resultante es mayor que un cierto umbral, el píxel se considera parte del fondo, si no, se considera background. Todos los píxeles que superen ese umbral servirán para crear una máscara que muestre donde se encuentra el foreground, donde se representarán con un valor de 1 (blanco) para el primer plano, y 0 (negro) para lo que no lo es. Como apunte, al hablar de máscaras, los valores 0 y 1 se refieren a los valores 0 y 255 en términos de imagen, respectivamente.

$$F(x,y) = \begin{cases} 1 & \text{si } |I_t(x,y) - B_t(x,y)| > T \\ 0 & \text{si no} \end{cases} \tag{5.1}$$

En la figura 5.6 se muestra la importancia de la elección del umbral.

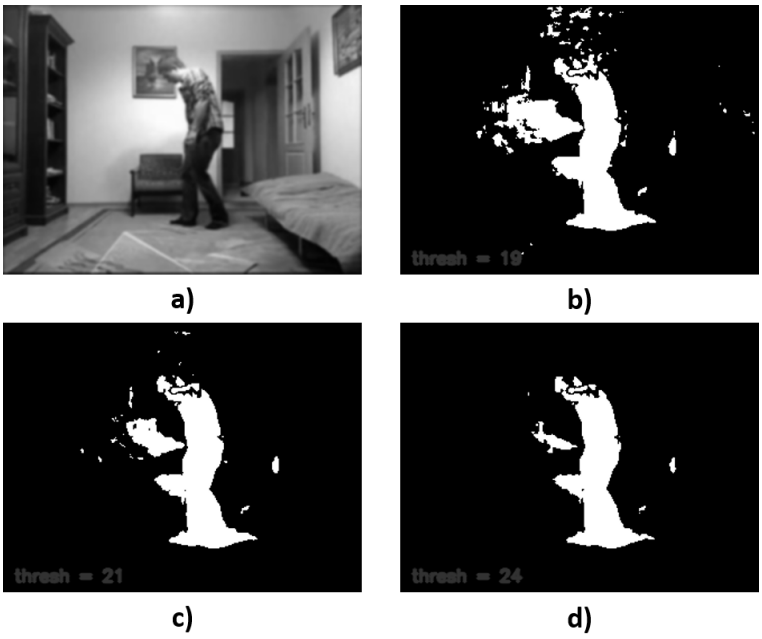

Figura 5.6: Comparación de umbral
a) Input frame, b) umbral de 19, c) umbral de 21, d) umbral de 24

Como se puede observar, al aplicar un umbral bajo, la máscara es más sensible a diferencias de iluminación, dando falsos positivos, mientras que, al aplicar un valor alto, se genera una máscara más fiable. La parte negativa de usar un umbral alto es que en ocasiones no "rellena" bien del todo la silueta de la persona, por lo puede inducir a la detección de varios objetos a la vez cuando solo hay uno en la escena. Como todos los pasos de esta aplicación, el umbral escogido tendrá que ser un valor intermedio que solucione estos problemas.

El principal problema de la umbralización reside en la propia elección del valor del umbral. Este valor es difícil de elegir ya que para cada escenario puede ser diferente. Idealmente el valor del umbral debería ser calculado mediante una función de la localización espacial (x, y). Por ejemplo, el umbral debería ser menor en las regiones con bajo contraste.

Una manera de facilitar la solución a este problema se basa en el uso de la estadística normalizada [13].

$$\frac{|I_t(x,y) - B_t(x,y) - \mu_d|}{\sigma_d} > T_s, \tag{5.2}$$

donde μ_d y σ_d son la media y la desviación estándar de $I_t(x,y)$–$B_t(x,y)$ para todas las localizaciones espaciales (x, y).

Otra manera es la que presentan Fuentes y Velastin [27]. Ellos proponen una modificación donde se usa la diferencia relativa del frame y el fondo, para enfatizar el contraste en las áreas más oscuras como sombras.

$$\frac{|I_t(x,y) - B_t(x,y)|}{B_t(x,y)} > T_c \tag{5.3}$$

Según explican, los píxeles que pertenezcan al fondo tendrán valores próximos al cero. Por otro lado, los píxeles que contengan elementos más oscuros que el fondo presentarán contrastes negativos, mientras que los objetos más claros que el fondo tendrán contrastes positivos. En este caso el valor del umbral ronda el de la unidad.

En la aplicación final se usará una umbralización estándar. Los entornos de grabación no son considerados tan complejos para necesitar de estos tipos de modificaciones, además de que su utilización supondría un aumento en los tiempos de procesamiento, por lo que se han descartado estas opciones.

5.4.1. Uso de histogramas

Como se ha explicado, usar un umbral fijo puede ser un problema cuando se crea una máscara. En las experimentaciones el valor escogido ha servido en la mayoría de casos, sin embargo, existen algunos donde la iluminación o la calidad de la cámara provocan que sea necesario un umbral mucho mayor.

Para arreglar este problema una técnica puede ser el uso de histogramas. Un histograma es un gráfico donde se representa la distribución de la intensidad de la imagen, es decir, un gráfico en el que se introducen los valores de los píxeles (de 0 a 255) en el eje X y el correspondiente número de píxeles en el eje vertical. En el caso de las imágenes RGB serán necesarios 3 histogramas, uno por cada canal, mientras que para las imágenes en escala de grises solo se necesita uno.

Cuando se produce un cambio de iluminación en la escena (apagar una lámpara, abrir las cortinas, etc.), la imagen resultante de la diferencia entre el modelo de fondo y el frame aumenta su valores, pero la forma del histograma tiende a mantener su forma. Es decir, la gráfica del histograma se desplaza a valores de intensidad más altos. Si la substracción se realiza entre imágenes con intensidades lumínicas similares, la mayoría de los píxeles de la imagen tenderá a intensidades próximas a 0. Por el contrario, si han habido cambios en la iluminación entre el fondo y el frame, los valores de la imagen resultante aumentarán de forma más o menos homogénea, tendiendo a agruparse sobre un valor superior a 0, como se observa en la figura 5.7

Figura 5.7: Comparación de histogramas

En la parte superior, la diferencia de iluminación entre background y frame es pequeña. En las imágenes de abajo existe más iluminación en el frame entrante. De izquierda a derecha: Modelo de fondo, input frame, substracción de fondo, histograma de la substracción.

El algoritmo incluirá una elección de umbral adaptativo que evitará falsos positivos cuando existan cambios de este tipo. El funcionamiento es el siguiente. Se calculará el histograma de la imagen resultante de la substracción, y se buscará la intensidad de píxel que se encuentre en mayor cantidad. Si esta cantidad supera un valor *histLim*, que se fijará de 4000, se harán iteraciones hasta encontrar la intensidad de píxel menor cuya cantidad no exceda ese valor, a partir de la intensidad de cantidad máxima. Cuando se haya dado con la intensidad correcta, el valor del umbral final será el valor de la intensidad encontrada más el valor fijado del umbral. Si el valor máximo del histograma es menor que *histLim* simplemente se usa el umbral fijado.

En las figuras siguientes se muestra los resultados del algoritmo, y el diagrama del código.

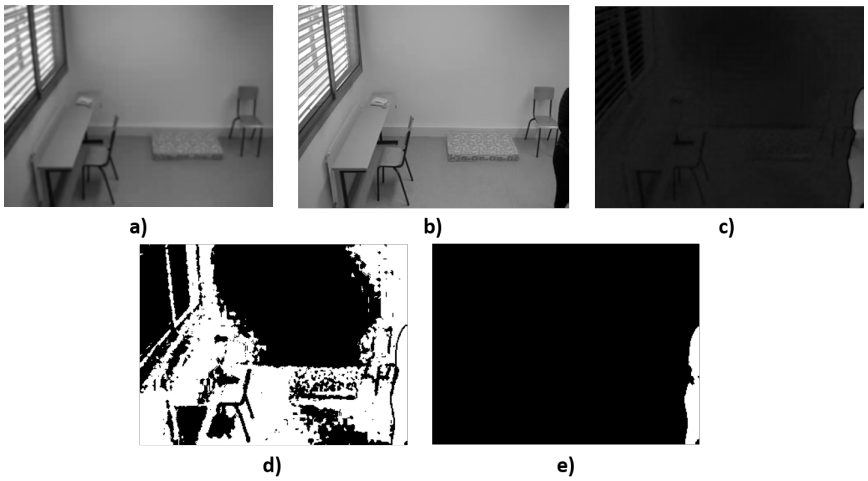

Figura 5.8: Resultados del algoritmo
a) Background, b) input frame, c) substracción de fondo, d) umbralización de la substracción (sin algoritmo), e) umbralización de la substracción (con algoritmo)

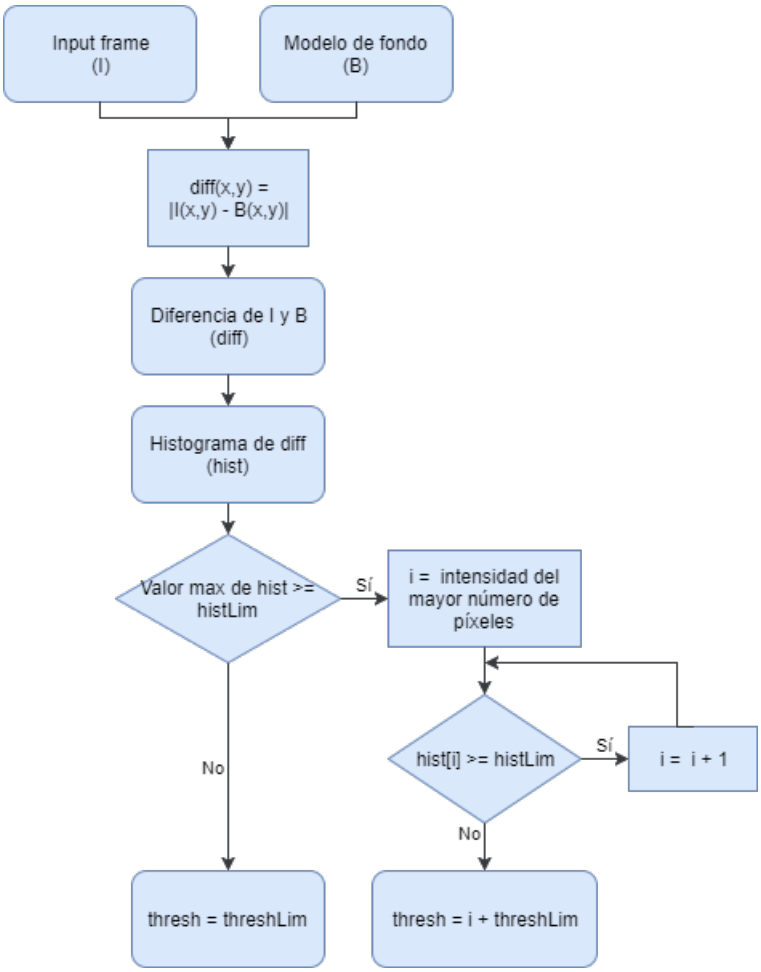

Figura 5.9: Algoritmo de elección de umbral

5.4.2. Zonas con saturación

A lo largo del desarrollo del código, se ha observado que aún con la detección de sombras y las transformaciones morfológicas, explicadas en las secciones siguientes, siguen existiendo zonas en las que se detectan falsos positivos. Después de varias observaciones se ha determinado que esto es debido principalmente a la presencia en los vídeos de áreas con mucha iluminación, en especial en paredes y objetos de colores claros. Éstas provocan que al pasar un objeto cerca de ella cambie ligeramente la iluminación a ojos del observador, pero hacen aparecen regiones cuya diferencia respecto al background sea lo suficientemente grande para que supere el umbral establecido y que no sean detectadas por el detector de sombras. Para solucionar este problema se optado por una opción que ha dado buenos resultados en los vídeos y que consiste en aplicar un umbral mayor a estas zonas.

El primer paso es convertir el frame actual al espacio de colores HSV (también llamado HSB), esto es, Matiz (Hue), Saturación (Saturation) y Valor/Brillo (Value/Brightness). Esto se puede hacer fácilmente gracias a las funciones que proporciona OpenCV. Se explicará de forma resumida el significado de cada canal.

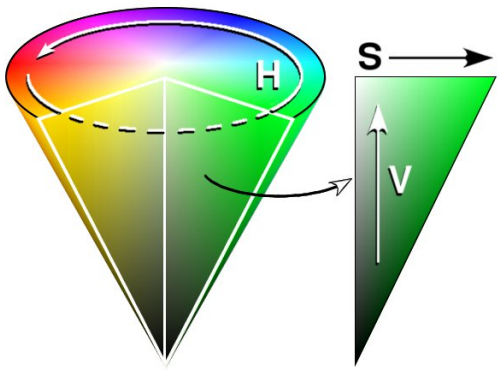

Figura 5.10: Espacio de colores HSV

El espacio de colores HSV se representa habitualmente con una ruleta de colores como se ve la figura 5.10. El matiz sería la parte circular, mientras que la saturación y el brillo compondría la parte triangular. El eje x de este triángulo representaría la saturación y el eje vertical el brillo.

- El matiz representa el color. La ruleta se divide en 360º donde cada ángulo representa un color distinto. Dependiendo del ángulo en el que te encuentres en la ruleta, el color varía por el rojo, verde y azul, obteniendo todos los colores de la gama cromática.
- La saturación, por otro lado, se puede explicar como la cantidad de gris que tiene el color o la intensidad del matiz. Cuanto menor sea la saturación, el color tendrá un tono más grisáceo. Los valores posibles van del 0 al 100 %, siendo el 100 % un matiz puro. Un color saturado tiene un tono vivo e intenso, mientras que otro color menos saturado presenta tonos más grises y apagados.

- El valor se define como el brillo relativo al brillo de un blanco iluminado de forma similar. Dicho de otra forma, sería el brillo o intensidad del color. Se representa en un intervalo de valores de 0 a 100 %, de negro a blanco respectivamente.

Los valores de los 3 canales son normalizados al intervalo de valores de 0 a 255 directamente por la función de OpenCV para su representación visual.

Después de dividir la imagen en estos 3 canales se observó que estas zonas que provocaban falsos positivos correspondían a regiones donde existía poca saturación, en particular, zonas de las paredes donde la luz incidía directamente. Al ser cambios tan repentinos de luminosidad en la escena, el modelo de fondo no tiene tiempo suficiente para actualizarse y origina que durante unos segundos surjan estos falsos positivos. Al convertir los frames a escala de grises, se pierde información por el camino al reducir los datos de 3 canales a solo uno, y lo que en términos de saturación y valor pueden ser diferencias de valores pequeños, en la imagen en blanco y negro la diferencia llega a ser mucho mayor, llegando a superar el umbral. A continuación, se muestra un ejemplo de estas observaciones.

Figura 5.11: Influencia de la saturación en la máscara de movimiento

En la fila de abajo se encuentra una imagen donde se pueden los efectos de la saturación. En la parte superior otro fotograma del mismo vídeo en el mismo escenario donde se ve a la persona pasando delante de la zona de baja saturación. De izquierda a derecha: input frame, frame en escala de grises, background, máscara de movimiento (con detección de sombras), canal de saturación del frame entrante.

Se observó que al cruzar los objetos por delante de estas regiones de baja saturación, estos seguían manteniendo su forma en el canal de saturación, independientemente del color o iluminación del objeto, es decir, que esas zonas aumentaban su saturación. De esta manera, la forma pensada para solucionar este problema fue la de aplicar un umbral mayor en las áreas de baja saturación del frame entrante.

Para ello, se crea una máscara mediante la umbralización del canal de saturación del frame actual. De esta forma se consigue definir una imagen donde se encuentran las zonas de baja saturación. Se estableció un umbral de 10. Acto seguido, se aplica la máscara de baja saturación a la diferencia del frame entrante

con el fondo creado y se le resta a ésta. Así, se separa la imagen de diferencia en dos imágenes, una conteniendo las regiones de baja saturación y la otra las de alta saturación. A continuación, se umbralizan, la de alta saturación con el umbral establecido anteriormente y el de baja saturación con un valor de umbral mayor. Se usó el mismo umbral más 10 (T+10). Por último, se suman las máscaras resultantes para obtener la máscara de detección de movimiento.

Después de aplicar este umbral se pudo comprobar como las máscaras creadas mejoraban sustancialmente. Las regiones con falsos positivos desaparecían y, además, en los vídeos donde no existían estos problemas, simplemente no se notaban los efectos de esta aplicación. Un ejemplo se puede ver en la siguiente figura. En ella se pueden ver los efectos de usar o no el algoritmo. En ambos casos se muestra la máscara de movimiento básica, sin aplicar tratamientos posteriores (erosión, dilatación, etc.).

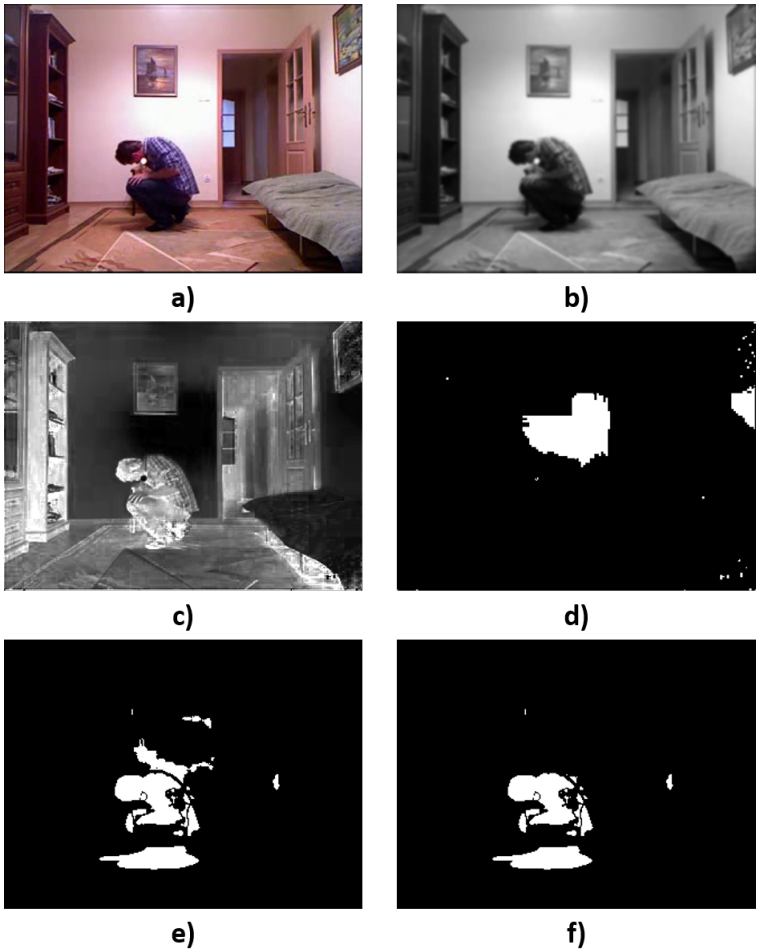

Figura 5.12: Corrección de la saturación en la máscara de movimiento
a) Input frame, b) frame en escala de grises, c) canal de saturación, d) máscara de zonas de baja saturación (umbral de 10), e) máscara de movimiento sin aplicar algoritmo, d) máscara de movimiento con algoritmo

Este algoritmo se aplicará como complemento a la umbralización de la diferencia del modelo de fondo y el frame actual. En la figura siguiente se muestra como se hace.

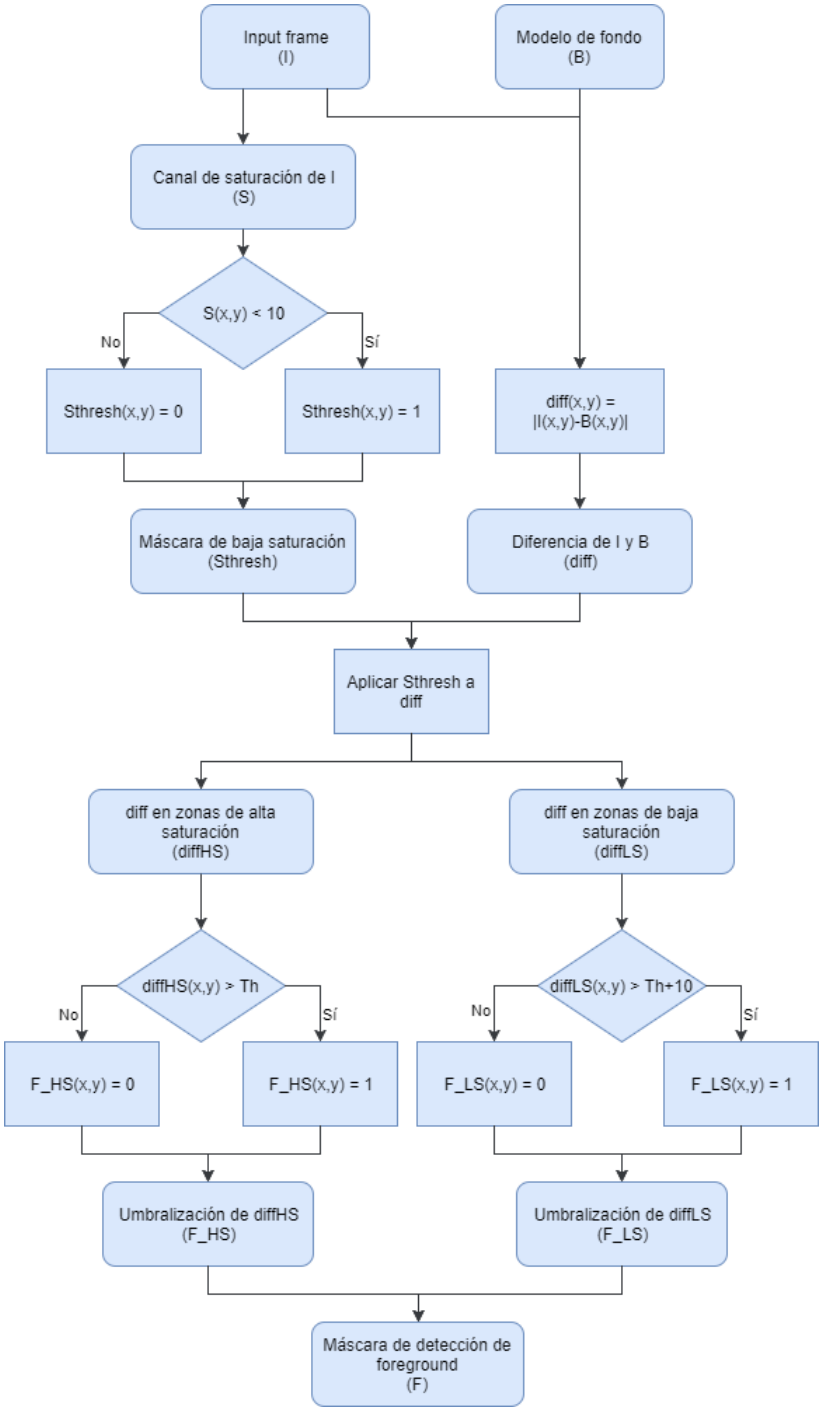

Figura 5.13: Diagrama de flujo del algoritmo de saturación

5.5. Detección de sombras

Con el filtro escogido se ha obtenido una máscara de primer plano apta, que sirve de forma básica para la aplicación. Sin embargo, siguen existiendo fallos de detección debido principalmente a los cambios de la iluminación y sombras de los objetos. Las sombras movibles asociadas a los objetos del foreground causan detecciones erróneas de los objetos de la escena. Una primera solución consistiría en aumentar el valor del umbral para evitar estos fallos, pero al hacer eso no se terminaría de detectar la figura total del objeto al excluir regiones de foreground, resultando impreciso. Además, solo se eliminarían las sombras poco intensas, es decir las sombras cuya intensidad no difiera demasiado respecto a la misma región del fondo. Es por tanto que es necesario crear un código que permita detectar las sombras y eliminarlas.

Para poder eliminarlas de nuestra máscara de movimiento, primero es necesario entender como son creadas y en qué consisten. Las sombras aparecen cuando un objeto opaco obstaculiza una fuente lumínica. La sección eficaz de la sombra es la silueta bidimensional del objeto que obstaculiza la luz. Las sombras se dividen en dos partes diferenciadas, la umbra y la penumbra. La umbra seria la sombra en sí. Es la región más oscura de la sombra y donde la luz está totalmente bloqueada por el objeto. La penumbra por otro lado, es la región donde la fuente de luz se encuentra bloqueada parcialmente. Esto se puede ver en la figura 5.14.

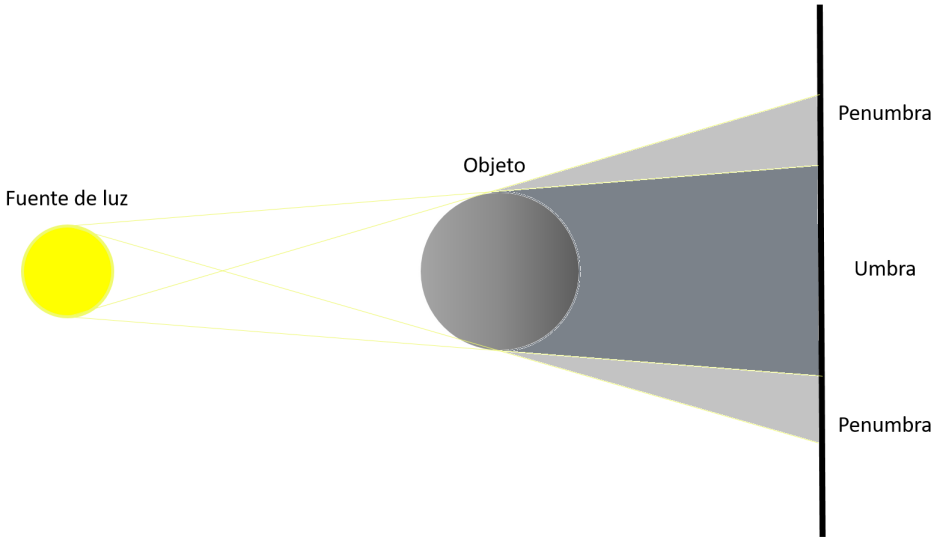

Figura 5.14: Umbra y penumbra

La detección de las sombras en computación visual ha sido considerada por mucho tiempo un componente crucial en las interpretaciones de escena. Pero a pesar de su importancia y su estudio, sigue siendo un problema. La dificultad principal se debe a las complejas interacciones entre la geometría, albedo e iluminación [28]. Localmente no se puede saber si una superficie es oscura por una sombra o por el albedo. Para saber si se trata o no de una sombra es necesario comparar la región con los alrededores y con otras con el mismo material y orientación.

No existe una única manera de detectar las sombras. Un primer acercamiento a este problema sería la descomposición de las imágenes en sus espacios de color. Las zonas sombreadas tienen la misma cromacidad que sus correspondientes píxeles del fondo, pero presentan distintos niveles de intensidad. A. Prati et al. [29] explican dos métodos que se basan en la comparación del modelo de fondo y frame actual usando la información del color de las imágenes. Uno usa el espacio de colores RGB mientras que el otro usa el HSV. Sin embargo, ambos necesitan modelos de fondo a color. El algoritmo desarrollado puede usar la información de color de los frames entrantes, pero el background se crea en escala de grises, por lo que no se pueden usar en la aplicación. Trabajos como los de Ruiqi Guo et al. [28] detectan las sombras a partir de las regiones de las propias imágenes, comparándolas con otras del mismo material. Sin embargo, esta técnica resulta demasiado compleja para la aplicación, además de ser probada en escenarios abiertos y bien iluminados.

Después de estudiar varias técnicas se usará la propuesta por J.C.S. Jacques et al. [30]. Ellos proponen un sistema de detección de sombras usando imágenes de vídeo en escala de grises, mediante la comparación del modelo de fondo y los frames entrantes, por lo que resulta perfecto para la aplicación. Se basan en la suposición de que las áreas sombreadas están correlacionadas con su correspondiente área en el modelo de fondo. Se divide en dos pasos principales, en el primero se crea una estimación de la sombra y por último se mejora en una segunda etapa de refinamiento.

A continuación, se muestra el diagrama de flujo del algoritmo.

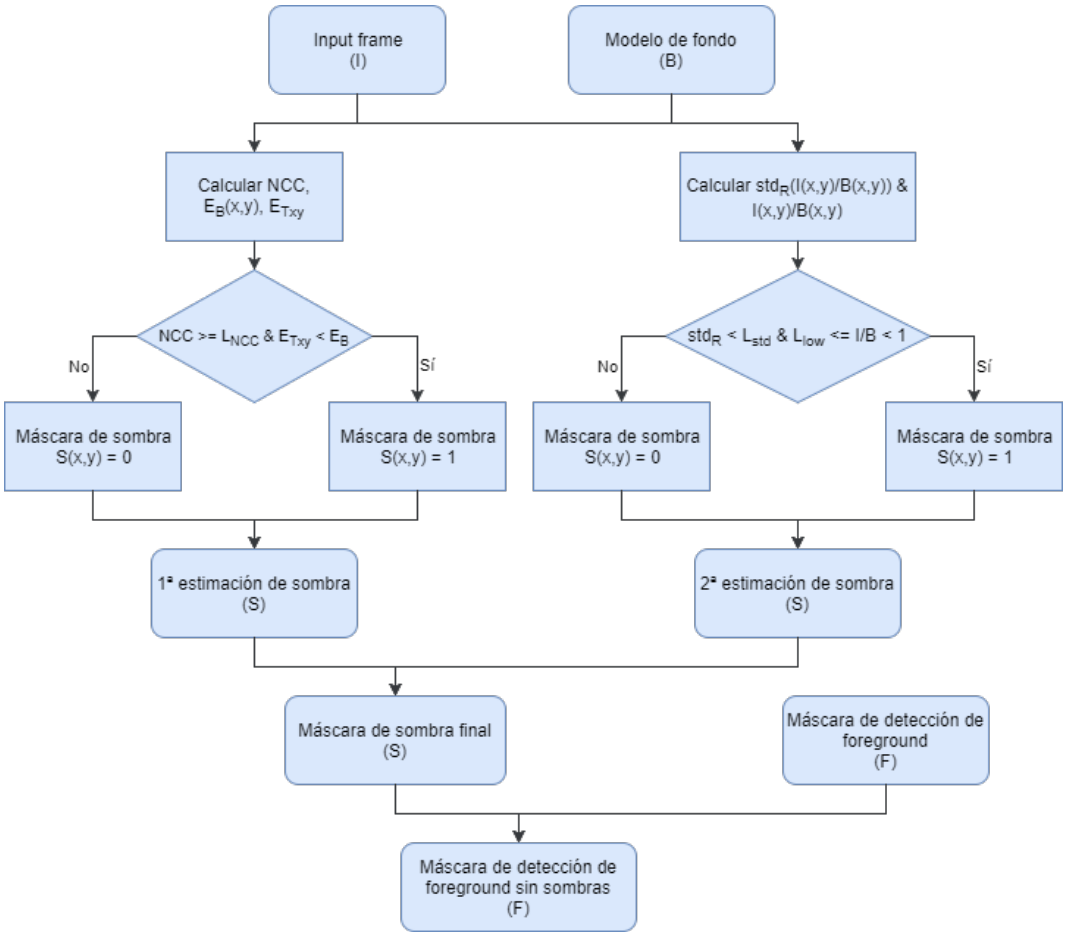

Figura 5.15: Diagrama de flujo de la detección de sombras

5.5.1. Identificación de sombras

Las áreas sombreadas son creadas cuando una cierta fracción de luz es bloqueada. Existen diversos factores que pueden influenciar la intensidad de un píxel, pero para el algoritmo se considerará que la intensidad de los píxeles de sombra es directamente proporcional a la luz incidente. Es decir, los píxeles de sombra serán versiones más oscuras de los respectivos píxeles en el modelo de fondo.

5.5.2. Detección de píxeles candidatos de sombra

La correlación cruzada normalizada, en inglés normalized cross correlation (NCC), es una medida de la similitud entre dos señales, o en este caso, el background y las regiones sombreadas. Esta técnica es útil para detectar píxeles candidatos de sombra, ya que puede identificar versiones escaladas de la misma señal. Lo primero que hay que hacer es crear una plantilla T_{xy} de tamaño $(2N + 1)$ x $(2N + 1)$, de tal forma que $T_{xy}(n, m) = I(x + n, y + m)$ para $-N \leq n \leq N$, $-M \leq m \leq M$, para cada píxel del foreground, es decir, T_{xy} es la "vecindad" del píxel $I(x, y)$. Una vez se haya definido la plantilla se procede a calcular la correlación cruzada normalizada entre la plantilla T_{xy} y el fondo B en cada píxel

mediante la ecuación:

$$NCC(x,y) = \frac{ER(x,y)}{E_B(x,y)E_{T_{xy}}}, \tag{5.4}$$

donde

$$ER(x,y) = \sum_{n=-N}^{N}\sum_{m=-N}^{N} B(x+n,y+m)T_{xy}(n,m) \tag{5.5}$$

$$E_B(x,y) = \sqrt{\sum_{n=-N}^{N}\sum_{m=-N}^{N} B(x+n,y+m)^2} \tag{5.6}$$

$$E_{T_{xy}} = \sqrt{\sum_{n=-N}^{N}\sum_{m=-N}^{N} T_{xy}(n,m)^2} \tag{5.7}$$

$E_{T_{xy}}$ es la energía de la intensidad del área definida por T_{xy}, mientras que $E_B(x,y)$ es la energía de la intensidad de la correspondiente región en el modelo de fondo. Para cada píxel (x,y) que se encuentre en una zona sombreada, el NCC en su área vecina T_{xy} deberá dar un valor grande, próximo a 1, y además, la energía $E_{T_{xy}}$ deberá ser menor que la energía $E_B(x,y)$ de su correspondiente región del background. Por tanto, un píxel (x,y) se clasificará como sombra si:

$$NCC(x,y) \geq L_{NCC} \text{ y } E_{T_{xy}} < E_B(x,y), \tag{5.8}$$

donde L_{NCC} se corresponde a un valor de umbral fijo. Si se incrementa L_{NCC}, el algoritmo será más restrictivo y solo detectará sombras intensas, descartando posibles sombras. Si se baja, por el contrario, clasificará más píxeles como sombra, pero puede dar falsos positivos.

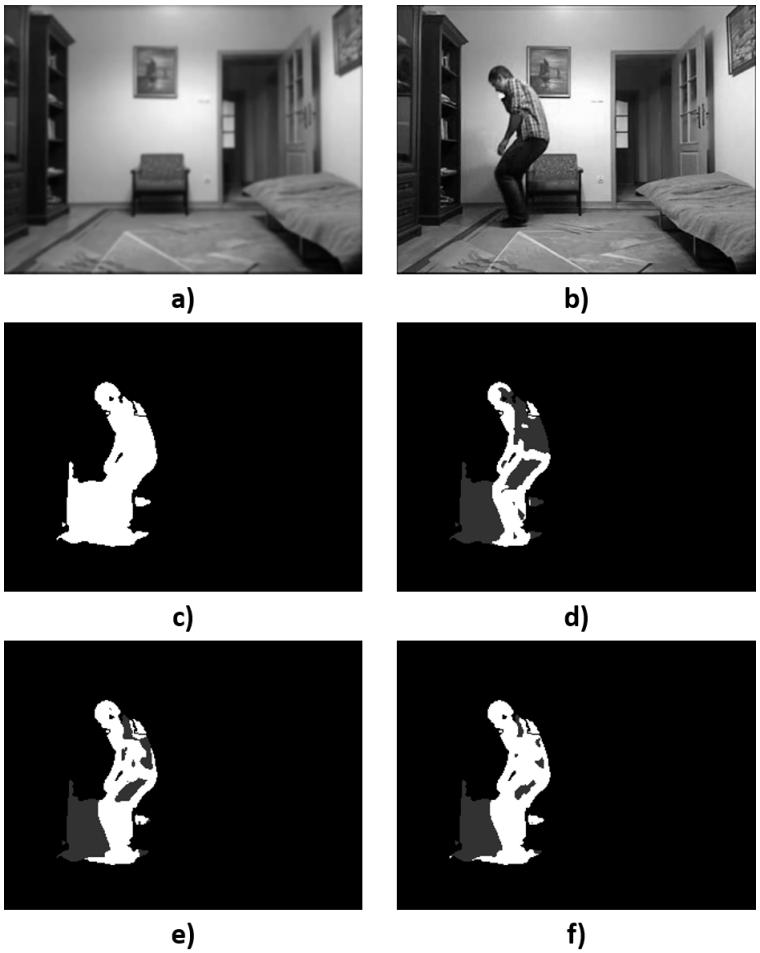

Figura 5.16: Comparación de distintos valores de L_{NCC}
Las zonas grises corresponden con las sombras detectadas. a) Modelo de fondo, b) input frame, c) imagen umbralizada, d) $L_{NCC} = 0.95$, e) $L_{NCC} = 0.98$, f) $L_{NCC} = 0.99$

El primer paso en los experimentos fue el de elegir el tamaño de la plantilla. Cuanto mayor es la plantilla, menos erosiona la figura del objeto, pero aumenta el tiempo de procesamiento del código. Además, un tamaño muy grande puede no detectar todas las sombras. Se estableció que un buen valor sería $N = 4$, como se propone en el artículo de J.C.S. Jacques et al. [30]. En cuanto al valor de L_{NCC} se muestra en la figura 5.16 una comparativa entre distintos umbrales, aplicados con la plantilla de tamaño $N = 4$. Los mejores resultados se obtuvieron con 0.98 y 0.99. Como se ha explicado antes, el primer valor detecta mejor las sombras, pero el segundo no deteriora tanto la silueta de la persona. Finalmente se optó por un valor de 0.99.

La implementación en Python es la siguiente:

- Primero se creará una ROI (región de interés) del frame filtrado y el modelo de fondo que contenga todos los contornos detectados más una distancia de N+1 píxeles a cada lado (para que la plantilla pueda operar correctamente).

Gracias a esta técnica se consigue reducir una media de 10 veces el tiempo de procesamiento en este paso.

- Seguidamente, para calcular ER, E_B y $E_{T_{xy}}$, se creará una matriz resultante de cada operación correspondiente y se usará la función *cv2.filter2D()* que haga el sumatorio en cada píxel. Esta función no admite matrices de 32 bits, por lo que para que funcione primero será necesario dividir las matrices (de 16 bits) entre 2^8, y después de aplicar la función se volverá a multiplicar por 2^8. Para obtener NCC simplemente se hace la división de forma directa.

- Después se utilizará *cv2.compare()* para hacer las comparaciones. De esta forma se obtendrán dos máscaras cuyos positivos serán las comparaciones de los positivos de éstas.

- Finalmente con *cv2.bitwise_and()* se hará una operación tipo AND entre las dos máscaras anteriores, y una última entre el resultado y la zona correspondiente de la imagen umbralizada para obtener la primera máscara de sombras.

5.5.3. Refinamiento de sombra

Como se puede ver, la técnica anterior ha proporcionado buenos resultados, pero siguen existiendo fallos que corrompen la figura del objeto, al detectar demasiados píxeles de sombra incorrectos. Para reducir estas clasificaciones incorrectas, se necesita una etapa refinamiento que minimice estos errores.

Lo que proponen J.C.S. Jacques et al. consiste en calcular el ratio $I(x,y)/B(x,y)$ en cada píxel candidato de sombra y comprobar si es aproximadamente constante, mediante la desviación estándar de $I(x,y)/B(x,y)$ en una región R de tamaño $(2M+1)$ x $(2M+1)$. El tamaño de R que aconsejan es de 3x3, que es el que se usará en los experimentos. En resumen, un píxel (x, y) será considerado un píxel de sombra si:

$$std_R\left(\frac{I(x,y)}{B(x,y)}\right) < L_{std} \text{ y } L_{low} \leq \frac{I(x,y)}{B(x,y)} < 1, \tag{5.9}$$

siendo L_{std} y L_{low} valores fijos de umbral y $std_R(I(x,y)/B(x,y))$ la desviación estándar en la región R. L_{std} controla la máxima desviación en la región R analizada, mientras que L_{low} previene la clasificación errónea de objetos oscuros con poca intensidad que sean interpretados como píxeles de sombra. J.C.S. Jacques et al. proponen unos valores para L_{std} y L_{low} de 0.05 y 0.5, respectivamente, pero los valores que mejores resultados han dado han sido 0.05 y 0.6. Un ejemplo del refinamiento se puede ver en la figura 5.17.

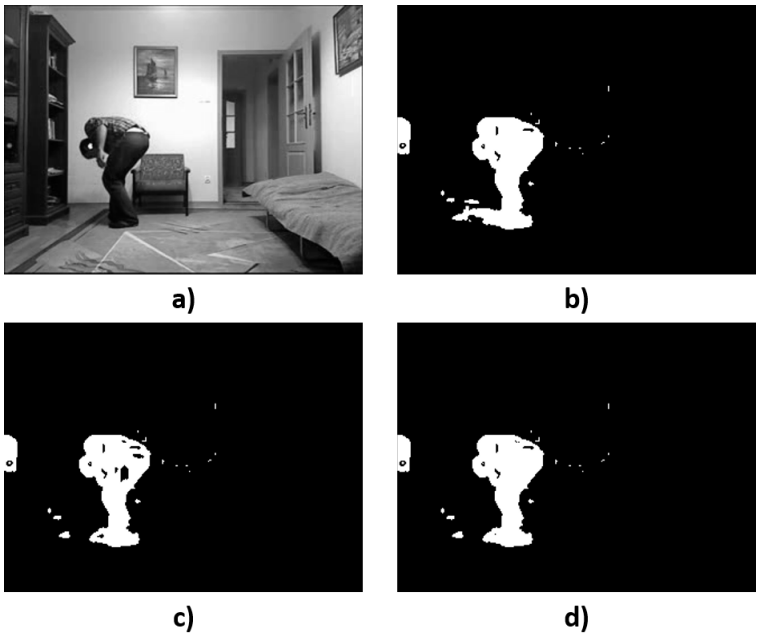

Figura 5.17: Refinamiento de sombras
a) Input frame, b) imagen umbralizada, c) foreground en primera etapa de detección de sombras, d) foreground final después de aplicar el refinado de sombras

La implementación del refinamiento es similar al paso anterior:

- Se volverán a usar los ROI para ahorrar tiempo. El cálculo del ratio I/B se hace de forma directa.
- Para conseguir la desviación estándar de I/B, primero se calculará la varianza cuya fórmula es $\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2$. Aplicar este algoritmo directamente sería algo difícil y lento por lo que se usará la fórmula equivalente $\left(\frac{1}{n}\sum_{i=1}^{n} X_i^2\right) - \overline{X}^2$. La primera parte se realizará calculando el cuadrado del ratio y después con *cv2.filter2D()* se hará el sumatorio píxel a píxel y se dividirá entre 9 (tamaño de la plantilla 3x3). En la segunda se usará la misma función con el ratio y se dividirá entre 9, y luego se elevará al cuadrado. Se restan los resultados para obtener la varianza. Por último, se obtiene la desviación estándar con la raíz cuadrada de la varianza calculada.
- Al igual que en el paso anterior, se realizan las comparaciones y operaciones AND correspondientes para obtener la máscara de sombras, y con una última operación AND entre las dos máscaras de sombras calculadas se obtiene la máscara de sombras final, la cual se restará a la zona correspondiente de la imagen umbralizada para obtener la máscara de foreground refinada final.

Después de corregir las zonas sombreadas de la máscara de foreground, el siguiente paso será aplicar una transformación morfológica que ayude a completar espacios vacíos y eliminar regiones de píxeles aislados. Algunos investigadores realizan esta etapa antes de la detección de sombras [4], pero en las experimentaciones se ha preferido hacerlo de esta forma.

5.6. Transformaciones morfológicas

Una vez es construida la máscara del foreground, es necesario aplicar una serie de transformaciones morfológicas para mejorar la detección de movimiento. Las transformaciones morfológicas son operaciones basadas en la forma de la imagen, que ayudarán a filtrar el ruido presente rellenando huecos o eliminando regiones pequeñas de ruido. Se usarán las transformaciones de erosión y dilatación.

Ambas se pueden realizar de forma directa con funciones de OpenCV.

5.6.1. Erosión

La idea básica de la erosión, como indica su nombre, consiste en la erosión de los bordes de los objetos del primer plano. La erosión es útil para descartar regiones aisladas de pequeño tamaño que no forman parte del foreground. ¿Cómo funciona? La idea es simple. Al igual que en el filtro paso bajo, se necesita crear un kernel o plantilla que se deslice por la imagen. En este caso el kernel estará compuesto por unos y ceros. Un píxel de la imagen original será considerado 1 siempre y cuando todos los píxeles bajo el kernel sean igual a 1. Si no, es erosionado, es decir, pasa a valer 0.

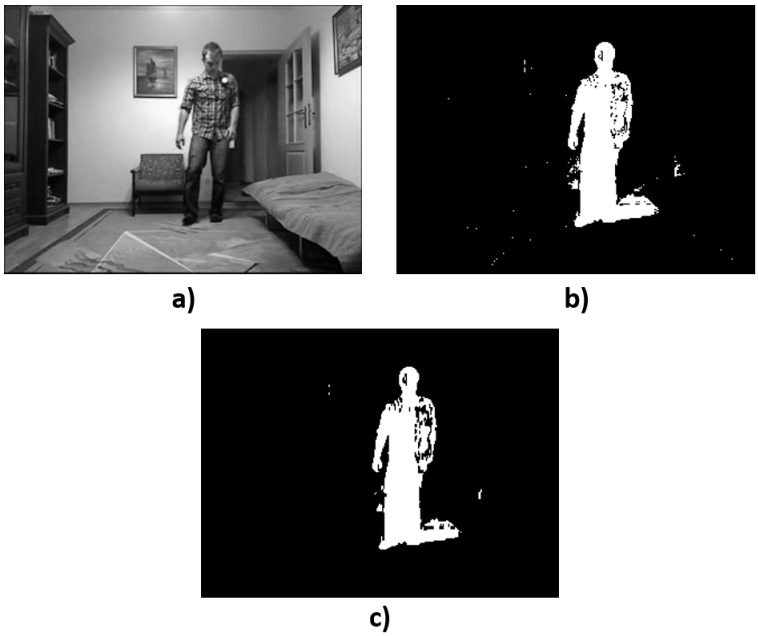

Figura 5.18: Ejemplo de la erosión
a) Input frame, b) imagen umbralizada, c) resultado de la erosión

En la figura anterior se puede observar los resultados de la erosión usando el kernel descrito en el siguiente párrafo. Estos resultados están aplicados a una imagen a la que no se ha aplicado suavizado, ni detección de sombras, para acentuar los efectos de la erosión. Se puede ver como las regiones con puntos blancos desaparecen después de aplicar el procesado.

Siguiendo los consejos de Jared Willems [4] se optará por usar un kernel de 3x1 como el que se observa en la figura 5.19. De esta forma, se consigue eliminar ruido sin deteriorar la figura de la persona.

Figura 5.19: Kernel usado para la erosión

Después de aplicar la erosión, el siguiente paso será usar la dilatación.

5.6.2. Dilatación

El principio de la dilatación es el contrario que el de la erosión. Al igual que en éste, se crea una plantilla que se deslizará por todos los píxeles de la imagen, sin embargo, el píxel será igual a 1 cuando al menos un píxel bajo el kernel sea igual a 1. De esta manera, se incrementan de tamaño las regiones blancas de la imagen, rellenando huecos.

La dilatación se hará después de la erosión. Al eliminar las regiones blancas, también se encoge el foreground. Mediante la dilatación se consigue volver a su tamaño original eliminando el ruido. Además, se puede usar para crear un foreground más consistente al unir partes separadas de un objeto.

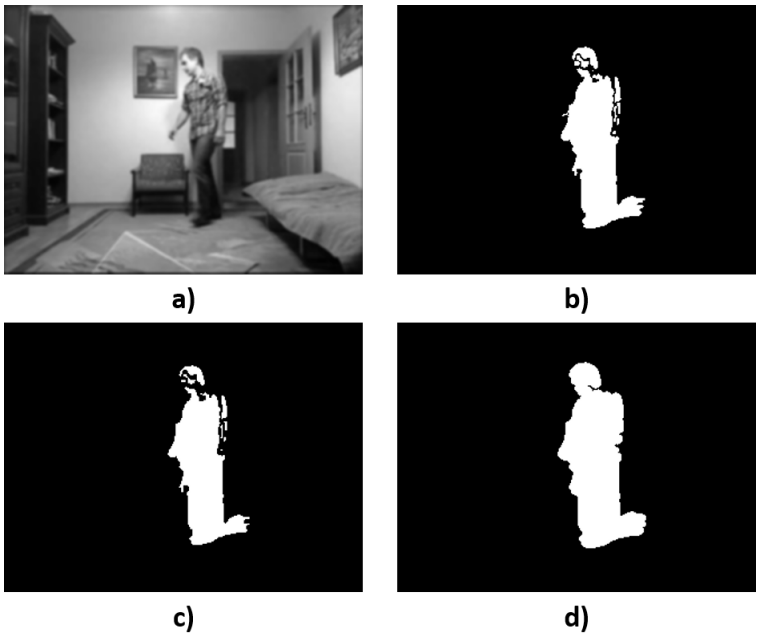

Figura 5.20: Ejemplo de la dilatación
a) Input frame, b) imagen umbralizada, c) resultado de la erosión, d) resultado de la dilatación posterior

En la figura anterior se observa como al aplicar la dilatación, la imagen erosionada recupera su forma y llena los huecos de la silueta de la persona. Para la prueba de la imagen se usó el elemento estructural descrito a continuación y no se aplicó la detección de sombras.

El kernel usado, al igual que el anterior ha sido el mismo que propone Jared Willems [4]. Es una plantilla de tamaño 7x7 pero siguiendo una estructura elipsoidal.

0	0	0	1	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
1	1	1	1	1	1	1
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	1	0	0	0

Figura 5.21: Kernel usado para la dilatación

5.7. Extracción de parámetros

5.7.1. Detección de blob

En las secciones previas se ha conseguido crear una máscara de movimiento consistente que ha servido para tener una buena estimación del movimiento . La siguiente etapa consistirá en la detección de la caída. El primer paso será localizar los blobs de la máscara. Como se ha explicado anteriormente, la máscara creada será un conjunto de píxeles blancos y negros, de valores 0 y 1 respectivamente (0 y 255 en el código). Un blob es un conjunto de píxeles blancos conectados entre sí, o lo que es lo mismo, que tienen píxeles vecinos del mismo valor.

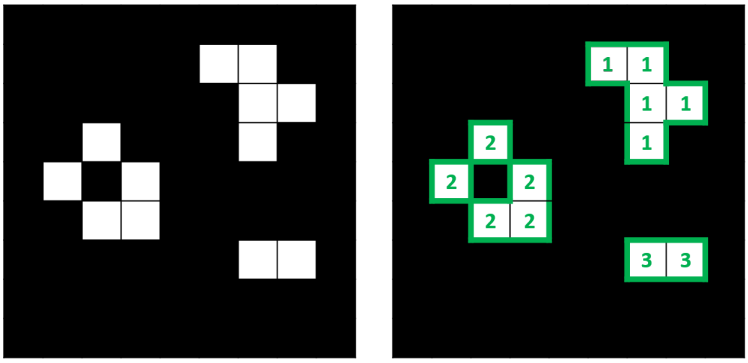

Figura 5.22: Ejemplo de blobs

Para localizar las dimensiones y parámetros de las personas detectadas es necesario localizar el contorno del blob que forman. Para ello, OpenCV proporciona funciones que buscan estos contornos y devuelven un array con los valores de los contornos encontrados.

Por último, habrá que descartar los blobs más pequeños, es decir, aquellos que en principio no pertenecen a la persona y han pasado el filtrado y umbralizado de la imagen. Para ello, se establecerá un área mínima bajo el cual estos contornos no serán analizados. Se determinó un valor de 700. Además se establecerá también un área máxima de 20000, para evitar contornos demasiado grandes, debidos principalmente a fallos de la máscara. El cálculo de las áreas de los blobs se explica en el apartado siguiente.

5.7.2. Cálculo de parámetros

Una vez se hayan localizado los contornos a analizar, se procederá a sacar los parámetros que puedan ser más importantes de cara a la detección de la caída. Los primeros parámetros a calcular serán los momentos geométricos del contorno. A partir estos momentos se pueden calcular varios de los parámetros requeridos. Los momentos geométricos del blob se calculan mediante la ecuación siguiente.

$$M_{pq} = \sum_{x} \sum_{y} f(x,y) x^p y^q \tag{5.10}$$

Donde $p, q = 0, 1, 2, \ldots$ y $f(x, y)$ es la intensidad en valor de escala de grises del píxel en el punto (x, y). OpenCV ya tiene una ecuación que calcula los momentos directamente.

5.7.2.1. Área y centroide

Gracias a los momentos es posible calcular el área y el centroide de la persona. Sus ecuaciones son las siguientes:

$$\text{Área} = M_{00} \tag{5.11}$$

$$\text{Centroide } (x_0, y_0) = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right) \tag{5.12}$$

El área también se puede calcular con la función *cv2.contourArea()* de OpenCV de forma directa.

5.7.2.2. Rectángulo y elipse "de confinamiento"

Otros parámetros interesantes son el rectángulo y elipse "de confinamiento". En el caso del rectángulo (o bounding box) se refiere al rectángulo que delimita la silueta del objeto, sin rotación. Es decir, el rectángulo que se forma al coger los píxeles más extremos del blob. Por otro lado, la elipse delimitante es la elipse con menor área que se ajusta al objeto.

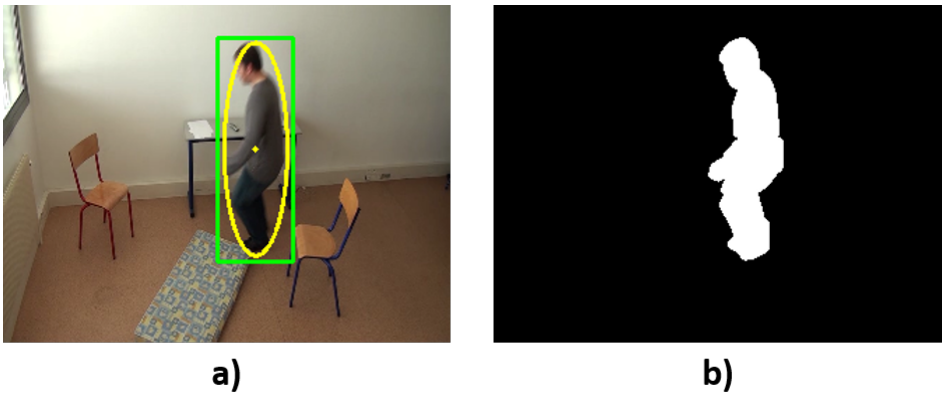

Figura 5.23: Ejemplo de ajuste de rectángulo y elipse

Ambas figuras se pueden obtener con las funciones de OpenCV. *cv2.boundingRect()* devuelve las coordenadas del vértice izquierdo superior del rectángulo, el ancho y la altura, mientras que *cv2.fitEllipse()* devuelve las coordenadas del centro, el ancho, la altura y el ángulo del rectángulo en el cual la elipse está inscrita. La representación gráfica se hace gracias a otras funciones de OpenCV.

5.8. Ajustes finales

Antes de extraer la información final de la escena, serán necesarios ciertos ajustes que ayudarán a tener una detección más precisa.

5.8.1. Unión de la silueta

Cuando se crea una máscara de movimiento, las siluetas de los objetos detectados no son siempre correctas. Debido a diversos factores como la aparición de oclusiones en la escena o cambios en la iluminación, las siluetas pueden quedar segmentadas en pequeñas siluetas, como se ve en la figura 5.24. Las aplicaciones de visión artificial destinadas al seguimiento de personas usualmente requieren de técnicas de rastreo de objetos, que ayudan a diferenciar el número de objetos del escenario y solucionan en mayor o menor medida el problema de la segmentación [31, 32].

Figura 5.24: Ejemplo de segmentación de silueta

La división de los objetos, ocasiona que las personas no sean detectadas de forma precisa, dando lugar a falsos positivos y falsos negativos. Por ello se necesita crear un algoritmo que solucione el problema. Debido a la complejidad de estas técnicas y a que la aplicación está destinada al rastreo de una sola persona (se supone que si hay más de una persona en la escena no se precisa del sistema), se optó por una solución sencilla que solucionaba con soltura estas complicaciones.

Se observó que la mayoría de segmentaciones se daban cuando la persona estaba caminando, es decir, se encontraba en una posición vertical, dándose la división a la altura de las rodillas. Es por ello, que el algoritmo consistirá en juntar aquellos objetos detectados que formen una figura vertical, evitando así, muchas detecciones de caídas erróneas.

El algoritmo considera el frame actual y localiza los contornos. Después coge los objetos relacionados entre sí para juntarlos en uno solo. Los criterios que sigue la aplicación para relacionar los contornos son los siguientes:

1. Para que dos contornos se consideren como uno solo, la distancia horizontal entre los centroides de ambos objetos deberá ser menor que una distancia *distXCenMax*.

2. La distancia vertical entre los perímetros de cada bounding box debe ser menor que el valor *distYBoxMax*.

Para *distXCenMax* y *distYBoxMax*, se estableció un valor de 20 y 25, respectivamente. Aparte, se realizó un segundo filtrado de contornos por área con un área mínima de 1500 y un área máxima de 20000, además de un filtrado por área mínima del bounding box de 1500 para no usar contornos erróneos.

Es necesario aclarar que, si un contorno se asocia con alguno de los contornos de un conjunto ya relacionado, éste entrará también en el conjunto. De esta forma, dos objetos no relacionados entre sí pasarán a estarlo gracias a un tercer contorno posicionado entre medias.

En la figura 5.25 se puede observar el efecto del algoritmo en la aplicación. En la imagen b) no se ha hecho uso del algoritmo, mientras que en c) sí. En b), la aplicación da una detección de caída positiva al medir el contorno verde. Gracias al algoritmo, en c) se crea una mejor detección de la persona y evita una detección equivocada.

Figura 5.25: Aplicación del algoritmo de unión de silueta

5.8.2. Delimitación del centroide

Cuando una persona sale o entra en la escena, el bounding box y la elipse que los contiene tienden a provocar fallos, ya que al recoger solo una parte del cuerpo (una pierna, la cabeza, etc.) el aspect ratio y el ángulo cambian de forma, dando lugar a detecciones erróneas. Además, como se verá en la siguiente sección, al desaparecer de escena la diferencia entre VPHs cambiará confirmando la caída.

Para solucionar este problema se pondrá una nueva condición que impedirá que se detecten caídas cuando el centroide del objeto se encuentre fuera de unos límites establecidos. Este límite será de 25 píxeles respecto a los extremos horizontales del frame y de 15 respecto a los verticales. Como se ve en la imagen, usando esta técnica, cuando la persona sale de la escena se evita una confirmación errónea.

Figura 5.26: Delimitacion del centroide

5.8.3. Desactivación de detección

Al igual que es importante evitar la obtención de falsos negativos, también es primordial evitar las confirmaciones incorrectas. Este tipo de confirmaciones son innecesarias y suponen una privación de la intimidad del usuario, al ser captado por las imágenes que son enviadas a los receptores de la alarma.

Un momento donde existen más falsos positivos puede ocurrir a la hora de apagar o encender las luces, cuando la cámara es cambiada de lugar, o si la persona o algún objeto pasa muy cerca de la cámara. Si ocurre esto, la máscara detectará múltiples blobs de gran tamaño, ocupando casi toda la pantalla.

Para evitar estos problemas, se implementará un algoritmo que deshabilite la detección cuando la cámara detecte demasiados positivos. La forma escogida será la de comprobar el área de los positivos de la máscara final. Si ésta supera un cierto límite, el programa no confirmará las caídas detectadas.

5.9. Detección de caída y confirmación

Una vez se han extraído los datos más significativos de la persona, habrá que usar esa información para detectar la caída. Como se ha explicado en la sección 2.5, existen diferentes formas de interpretar los datos. La aplicación hará uso de varias de ellas para ser más eficiente. Primero habrá una primera etapa de detección en la que se hará uso del ángulo de caída y del aspect ratio. Una vez se haya hecho la detección, se pasará a una segunda en la que se utilizarán los histogramas de proyección verticales para confirmar la caída.

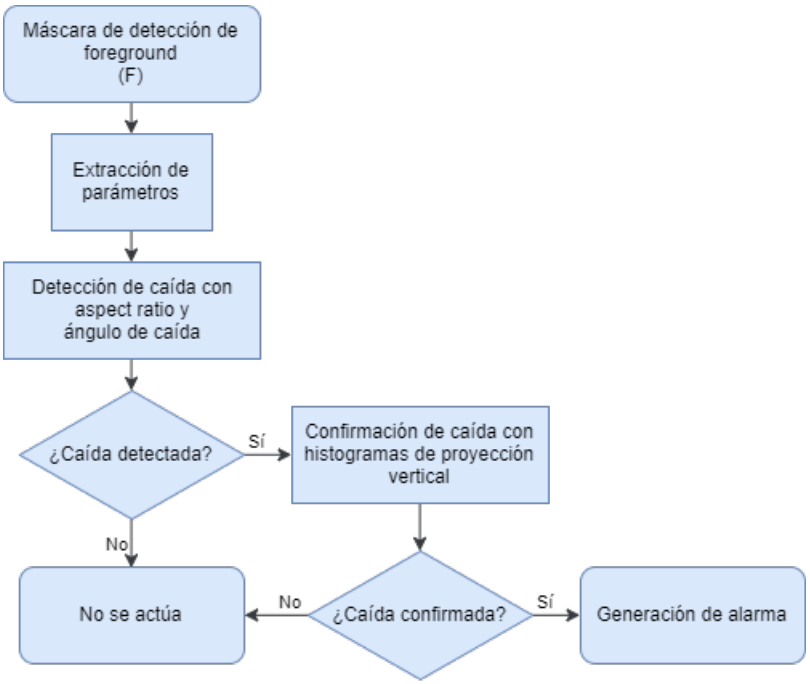

Figura 5.27: Diagrama de flujo de la detección de caídas

5.9.1. Ángulo de caída

Un aspecto bastante significativo en una caída se encuentra en el ángulo que forma la persona al caer. En esta aplicación, se usa el ángulo de caída que devuelve la función de ajuste de la elipse, es decir, el ángulo que forma el eje mayor de la elipse con el eje horizontal. Sabiendo el valor de este ángulo, se puede hacer una primera estimación y comprobar si la persona ha caído. Para un cálculo mejor y una representación gráfica más visual, se establece los 0 grados cuando el objeto está en horizontal y 90 grados cuando está en posición vertical [0 – 90º], sin considerar la dirección de la caída. Según V. Vishwakarma et al. [20], cuando una persona se encuentra de pie, se asume que tendrá un ángulo α de 90 grados. Cuando camina, este ángulo variará de 45 a 90 grados, y cuando caiga el ángulo será siempre menor que 45 grados.

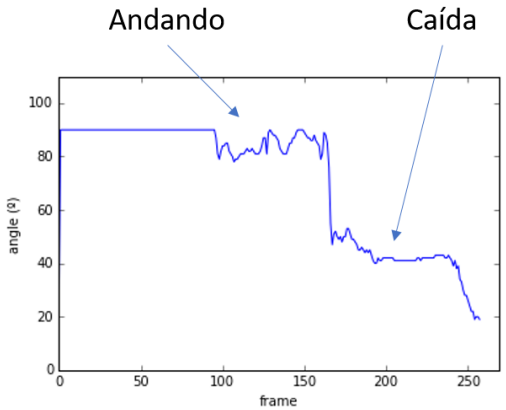

Figura 5.28: Ejemplo de la evolución del ángulo de caída

V. Vishwakarma et al. proponen usar este parámetro como método de confirmación, sin embargo, se pensó que es más conveniente usarlo en la primera etapa de detección. Si la máscara está bien construida esta característica puede ser un factor importante de confirmación, pero si ha habido algún error puede dar falsos positivos o lo que es peor, falsos negativos, por lo que no se ha considerado que sea un parámetro decisivo. Además, este método solo funciona si la caída se ha hecho de lado con respecto a la cámara, por lo que una caída frontal o hacia atrás no producirá confirmación.

5.9.2. Aspect ratio

El aspect ratio es también una manera rápida y sencilla de comprobar si ha habido una postura anormal de la persona. Se calculará mediante la división de la altura entre el ancho del bounding box de la persona. Cuando la persona cae, estos parámetros cambian drásticamente. Cuando el ratio es menor que 1, se asume que la persona ha caído [20].

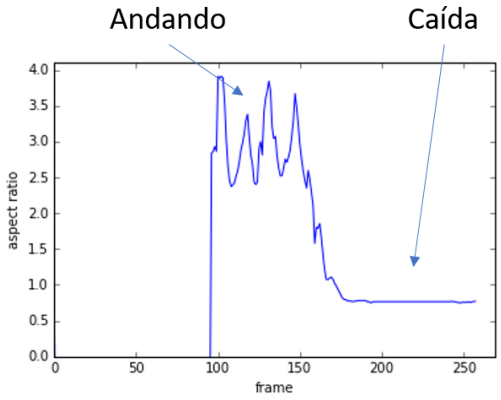

Figura 5.29: Ejemplo de la evolución del aspect ratio

5.9.3. Histogramas de proyección vertical

Una vez se ha activado alguna de las dos condiciones anteriores, se necesita hacer una segunda comprobación para reafirmar los resultados. Debido a que el foreground obtenido mediante la substracción de fondo puede dar falsos positivos, es posible que se haya detectado una caída que no ha ocurrido. Esta confirmación se hará mediante el uso de los histogramas de proyección vertical (VPH). Es propuesta como técnica de detección de caída por Lin et al. [33]. La forma de calcular los histogramas se muestra en la sección 2.5.

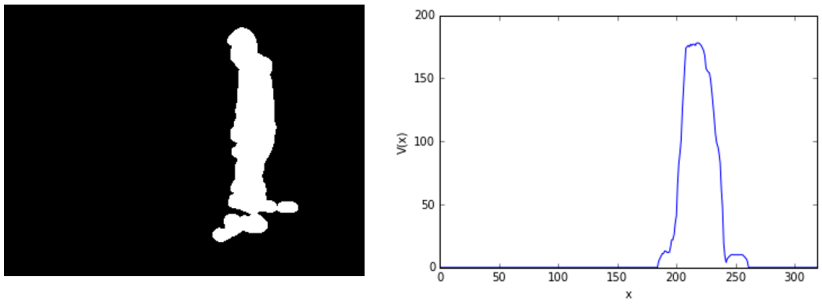

Figura 5.30: Histograma de proyección vertical

Cuando se produce una caída, según indican, el histograma de proyección vertical cambia significativamente en el periodo de la caída. Este periodo suele durar un rango de entre unos 0.4 $\sim$ 0.8 s. Mediante la comparación de los histogramas del frame actual con el frame de 1 segundo anterior podemos comprobar si se ha producido el evento o no. Con el uso del intervalo de 1 segundo se consigue que los movimientos lentos como agacharse o tumbarse no sean confirmados como caídas. Una manera de realizar las comparaciones puede ser calculando el ratio de cambio entre los máximos valores de los histogramas:

$$CR_{norm} = \frac{V_{prev_{max}} - V_{cur_{max}}}{V_{prev_{max}}}, \tag{5.13}$$

donde $V_{prev_{max}}$ es el valor máximo del VPH del frame 1 segundo antes de detectarse la caída, y $V_{cur_{max}}$ es el valor máximo del histograma del frame actual.

Lin et al. definen varios tipos de movimiento para los cuales han establecido diferentes umbrales desde 10 % a 30 %. Esta técnica tiene la ventaja de requerir muy poca carga de procesado, al usar solamente los valores máximos y no tener que hacer cálculos extras. Sin embargo, aunque este método da resultados aceptables, no son suficientes. La diferencia de los valores máximos de los histogramas no distingue del todo bien cuando una detección se ha producido por una caída o porque la persona se ha tumbado, aun modificando el intervalo de tiempo.

Ya que $V(x)$ es una distribución unidimensional, la comparación se puede hacer usando la distancia de Bhattacharyya [33], usando la ecuación 5.14. Este método permite conocer con mayor precisión la similitud entre dos muestras unidimensionales. Cuanto menor sea la distancia resultante, mayor será la diferencia entre cada histograma.

$$d(V_1, V_2) = \sum_x \sqrt{\frac{V_1(x)}{\sum_u V_1(u)} \frac{V_2(x)}{\sum_v V_2(v)}} \tag{5.14}$$

Usando este método se consiguen resultados mucho más exactos, descartando muchas confirmaciones erróneas. Este último tiene el inconveniente de que requiere bastante tiempo de procesamiento, pero solo es activado cuando ocurre una detección, por lo que no afecta al funcionamiento normal de la aplicación.

5.9.3.1. Mejora de la técnica de Bhattacharyya

Como se ha explicado, se ha conseguido obtener una medida más correcta de la diferencia entre histogramas. No obstante, con esta técnica no se puede distinguir si la diferencia ha sido generada por una bajada o una subida de los valores, cosa que sí hace la primera técnica. Este aspecto es importante si se quiere tener una confirmación más fiable. Un ejemplo puede ser cuando una persona tumbada ha pasado a formar parte del fondo. Cuando éste se levante, el sistema detectará la posición donde se encontraba y donde se encuentra actualmente. La primera zona será interpretada como una caída al tener un ángulo y ratio inferior al fijado. Al medir la diferencia de histogramas confirmará la caída al haber aumentado sus valores. Como medida para evitar este tipo de confirmaciones se usará una segunda etapa de confirmación que utilice también los valores máximos.

Figura 5.31: Confirmación errónea con Bhattacharyya
Arriba: frame 1 segundo anterior, abajo: frame actual
De izquierda a derecha: Modelo de fondo, input frame, máscara de movimiento, histograma de proyección vertical

Este paso consiste simplemente en hacer la comparación de los valores máximos de los histogramas, entre los valores de la distancia horizontal que ocupa la persona en el frame de la caída. Esta distancia será la distancia que ocupa el bounding box más un valor de 10 a cada lado. Si el valor máximo del frame de hace un segundo es mayor que el actual, querrá decir que han bajado los valores, y por tanto será una caída. De esta forma, se consigue mejorar la técnica de Bhattacharyya, obteniendo hasta un 6 % más de resultados correctos, según las experimentaciones.

5.10. Sistema de alarma

La generación de la alarma supone la etapa final de la aplicación. La alarma puede ser creada de múltiples modos. Desde mensajes de texto, a llamadas a la persona encargada de socorrer a la víctima, pasando por incluso alarmas sonoras que avisen en las inmediaciones.

El sistema de alarma que se instalará en el dispositivo será la creación de un mensaje de correo electrónico que se envíe a un familiar o cuidador, que contenga

una foto del momento de la caída. Se ha elegido esta opción debido principalmente a su sencilla implementación en el código, su inmediatez y la opción de enviar imágenes. La forma de implementar este aviso es la siguiente:

1. Primero se definirán las direcciones de los correos que recibirán y mandarán los mensajes, y la contraseña de este último. En el caso del remitente, la dirección deberá ser una cuenta gmail, al haberse establecido una conexión con el servidor de correo saliente de gmail (smtp.gmail.com). De todas formas, se podría cambiar la dirección del servidor si la cuenta es de otro tipo.

2. Al momento de confirmar una caída, el programa hará una captura del momento y la almacenará en su memoria. Esta imagen será la que se envíe.

3. Se establece la conexión con el servidor SMTP (Protocolo para Transferencia Simple de Correo), se habilita el modo TTL (también se podría utilizar SSL) y se identifica en el servidor.

4. Con la biblioteca *email* se crean variables tipo MIME que permitirán crear el contenido del mensaje. El contenido incluirá un mensaje de texto y la foto del momento de la caída.

5. Por último, se envía el correo electrónico al destinatario y se cierra la conexión con el servidor.

Figura 5.32: Envío de correo de alarma

Capítulo 6

Experimentación

En las secciones anteriores se ha visto el funcionamiento e implementación del algoritmo. Estos apartados han explicado paso a paso las etapas del sistema de detección de caídas, argumentando la selección de los diferentes métodos usados. Para ser capaces de elegir las técnicas y parámetros más adecuados, es necesario realizar pruebas que determinen el desarrollo del algoritmo. Todas las pruebas, menos la prueba final de implementación en la Raspberry Pi, se realizarán en un ordenador personal, por lo que los tiempos de computación pueden variar respecto a los experimentados en el sistema final.

En este capítulo se mostrarán las distintas experimentaciones realizadas durante el desarrollo de la aplicación y sus resultados.

6.1. Elección de técnica de substracción de fondo

En la sección 5.3 se hace un estudio de las dos técnicas de substracción de fondo que parecen las más adecuadas para la aplicación. La aplicación final usará solo uno de estos dos métodos. Para elegir cuál, se han harán varias pruebas con diferentes vídeos que ayuden a observar cuál ofrece mejores resultados. Es conveniente explicar que también se hicieron pruebas aplicando la mezcla de Gaussianos, con la función *cv2.createBackgroundSubtractorMOG()* que ofrece OpenCV, pero como se explica en el apartado 2.4.2.2, este tipo de técnica no es la más adecuada, además de dar resultados no muy satisfactorios.

La elaboración de estas pruebas se enfocará en comprobar qué método realiza la mejor creación del fondo, por lo que no se aplicará ningún procesado antes o después de la sustracción de fondo. Las únicas modificaciones serán la conversión a escala de grises de los frames y el cambio de resolución.

6.1.1. Criterios de aprobación

Escoger un criterio para decidir cuándo un método funciona o no es una tarea compleja, ya que se deben tener muchos factores en cuenta. La forma de proceder en esta experimentación será la de comprobar qué técnica tiene menores tiempos

de procesado y cuál construye un modelo de fondo más fiable.

El primer criterio a seguir se medirá mediante el tiempo medio que un frame es procesado a lo largo de los vídeos de prueba. Una fórmula sencilla pero eficaz para comparar las dos técnicas.

El segundo es más complicado de medir. La manera idónea sería la de obtener una imagen exacta del fondo de forma continuada e ir comparándola con la creada por el algoritmo. Sin embargo, es algo casi imposible, debido a que el propio fondo va cambiando en iluminación y elementos en la escena. Por ello, para realizar las pruebas se considerará que el fondo no cambia en todo el vídeo, es decir, que es estático.

Para ello cogeremos un frame del vídeo donde aparezca el background sin objetos en movimiento, que normalmente será el primero. Después cogeremos el fondo que se construye a cada frame y se lo restaremos al background previamente escogido, en valores absolutos. Estos valores se sumarán y se dividirán por el número de píxeles del frame, obteniendo una media que indicará cuánta es la diferencia entre el fondo estimado y el calculado por frame. Por último, se hará una última media con estas medias, que reflejará la precisión obtenida en cada vídeo.

Antes de empezar la experimentación, el primer paso que se dará será el de elegir los parámetros más adecuados de cada método. Esto puede resultar una tarea tediosa, ya que existen múltiples combinaciones y sería muy difícil ir comprobando una a una cual es la que ofrece mejor resultados. Para facilitar este problema, se escogerán varios vídeos, y se realizará con ellos una medición iterativa definiendo en cada loop parámetros diferentes.

Después de realizar estas mediciones, se escogerán los parámetros que presenten una diferencia de background menor y un tiempo de procesado menor en los vídeos, y sobre esos valores se trabajará y se harán las comparaciones. Cabe destacar que en las experimentaciones se llegó a la conclusión de que el tiempo de procesado sufría una reducción significativa si a la hora de realizar las pruebas, no se mostraba el vídeo por pantalla, por lo que se harán de esta forma.

Es necesario aclarar que los parámetros que den como resultados una buena media del background no significan que sean los parámetros más adecuados, sino que son los que crean un fondo más parecido al elegido. Es por ello que son solo orientativos, es decir, ayudarán a hacer las comparaciones y filtrar parámetros. Evidentemente, la selección de parámetros que ocasionen una baja velocidad de actualización del fondo serán los que mejor resultados den, al no darle tiempo al fondo para actualizarse antes de que acabe el vídeo. La elección final requerirá la observación e interpretación de sus efectos de manera manual.

Ya que la experimentación en todos los vídeos sería una tarea que consumiría mucho tiempo y algo innecesario, se han escogido 4 vídeos (2 con caída y 2 sin caída) al azar, para hacer las pruebas. A continuación, se muestran los resultados

más relevantes obtenidos en los vídeos y se elegirán y explicarán los parámetros.

6.1.2. Filtro con mediana

Para el filtro con mediana se establece una medición iterativa del número de frames por actualización (n_{max}) de 1 a 10 y un tamaño de buffer (*buffSize*) de 1 a 50. Las iteraciones se harán incrementando el tamaño del buffer (con números enteros) en cada loop. Cuando llegue a 50, o la media de tiempo de procesado supere un cierto valor (0,05 seg/frame), se parará ese loop y se incrementará el valor de n_{max}, iniciando de nuevo las iteraciones.

Los resultados obtenidos distan mucho entre sí. El tiempo medio de procesado varía desde los 800 μs hasta los 0,05 seg (este último valor debido a la limitación anterior), mientras que la media varía entre valores de 10 a 0. Debido a que el número de combinaciones es mucho mayor en este caso que en el filtro con mediana aproximado, se filtrarán los resultados descartando los cuales tengan un tiempo medio de procesado mayor que 0,02 seg/frame y una media mayor que 0,9. También se descartarán aquellos que tengan una media de 0, ya que aparecen cuando n_{max} y *buffSize* tienen valores demasiados altos, es decir, cuando el background no tiene tiempo de actualizarse antes de que se acabe el vídeo.

A continuación, ya que existen muchos resultados distintos, se mostrarán los que mejores resultados han dado en las observaciones, en cada vídeo.

Vídeo	n_{max}	**buffSize**	**Tiempo medio de procesado (μs/frame)**	**Media**
adl-01-cam0_MOD	10	17	3228,420015	0,184087493
	10	18	3170,932077	0,164619162
	10	26	3978,144275	0,007455082
	9	28	4927,171911	0,014584469
adl-06-cam0_MOD	10	27	4686,971777	0,23475937
	10	31	5253,021771	0,111938228
	9	29	5726,583436	0,303391728
	9	30	5977,792329	0,296975016
fall-01-cam0_MOD	10	18	3201,623784	0,132996757
	10	21	3424,045722	0,057717505
	9	29	4825,553669	0,006071312
	9	30	4814,275241	0,001838108
fall-05-cam0_MOD	10	21	3515,399719	0,167069097
	10	28	4254,087056	0,00030599
	9	28	4533,718962	0,011487066
	9	30	4814,275241	0,001838108

Tabla 6.1: Resultados del Filtro con mediana

Como es de esperar, a menor tiempo de actualización del buffer (n_{max}), y a mayor tamaño de éste, el tiempo de procesado medio por frame aumenta considerablemente. Por otro lado, la media disminuye al incrementar el tamaño y el tiempo de actualización. Esto último, se debe a que el fondo tarda más en actualizarse, lo que es conveniente para fondos estáticos. Sin embargo, no es preferible

tener un segundo plano poco adaptativo, ya que será poco sensible a cambios del fondo.

Los parámetros elegidos finales serán:

- ***buffSize***: 24
- $\boldsymbol{n_{max}}$: 10

6.1.3. Filtro con mediana aproximada

Para el filtro con mediana aproximado se establece un valor de cómputo de 0,1 a 5. En este caso el incremento se hace con 0,1 y se termina el proceso cuando llega a 10. En la tabla 6.2 solo se muestran los resultados desde 0,1 a 0,4 ya que a medida que el parámetro *compNum* aumenta, la media aumenta de igual forma hasta un valor aproximado de 10 para el valor de 10.

Vídeo	compNum	Tiempo medio de procesado (μs/frame)	Media
adl-01-cam0_MOD	0,1	2057,102456	2,08472831
	0,2	1867,948433	3,254744302
	0,3	2275,089376	4,183246819
	0,4	2055,583494	4,854806697
adl-06-cam0_MOD	0,1	1823,417023	3,396289915
	0,2	1903,73218	5,208986536
	0,3	2090,752438	6,172403737
	0,4	1999,046559	6,78748931
fall-01-cam0_MOD	0,1	1864,704675	1,843019032
	0,2	1906,202426	2,382624885
	0,3	1842,826842	2,777163014
	0,4	1959,139426	3,061207334
fall-05-cam0_MOD	0,1	1751,214457	2,88554158
	0,2	1943,701416	3,71659592
	0,3	1776,31298	4,012736632
	0,4	1822,377538	4,226671615

Tabla 6.2: Resultados del Filtro con mediana aproximado

Después de realizar las pruebas con diferentes parámetros, se ha llegado a la conclusión de que la manera de obtener un fondo más estable es la de escoger un valor pequeño para la variable de cómputo (*compNum*). De esta forma se obtiene un fondo que cambia lentamente, permitiendo que para escenarios estáticos como los que serán grabados por la aplicación funcione bien. La parte negativa de escoger un valor bajo es que reacciona de forma lenta a cambios de la iluminación o de elementos en el fondo, por ello no es aconsejable escoger los valores más pequeños en ciertas situaciones.

El valor final escogido será:

- ***compNum***: 0,1

6.1.4. Comparación de técnicas

Después de elegir los parámetros, se muestra la media de los resultados de los 4 vídeos aplicando los respectivos métodos:

	Tiempo medio de procesado (μs/frame)	**Media**
Filtro con mediana aproximada	1874,109653	2,55239471
Filtro con mediana	4143,732363	0,09404084

Tabla 6.3: Comparación de técnicas

El primer resultado que se puede observar es la diferencia en el tiempo de procesado. Mientras que el filtro con mediana aproximada requiere poco tiempo (independientemente de los parámetros elegidos), y resulta constante a lo largo del vídeo, el filtro con mediana en cambio aumenta su valor por 4. Además, el tiempo de procesado no es igual en cada frame, sino que la duración normal de ejecución es muy baja, pero en el momento en que se actualiza el fondo, ésta sube drásticamente, aumentando la media.

En el caso de la media de las comparaciones entre el fondo real y los creados por el algoritmo, sin embargo, ocurre lo contrario. Quien presenta mejores resultados es el filtro con mediana, obteniendo una media muy baja. El filtro con mediana aproximada da un valor 27 veces mayor, dando unos resultados bastante peores.

En la figura 6.1, se ve una comparación visual de ambos métodos, con un umbral de 20.

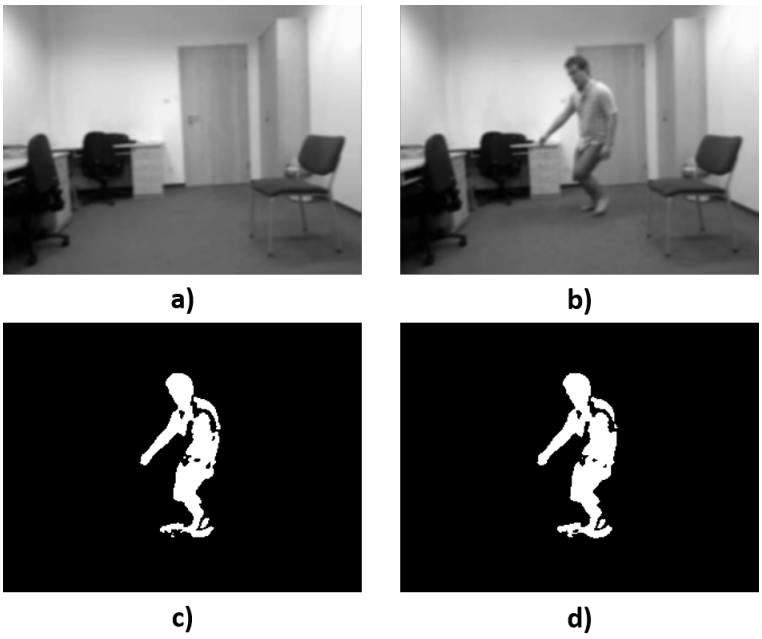

Figura 6.1: Comparación de técnicas
a) Background creado manualmente, b) input frame, c) foreground creado por el Filtro con Mediana Aproximado (umbral de 20), d) foreground creado por el Filtro con Mediana (umbral de 20)

Después de observar todas las ventajas e inconvenientes de cada método, se elegirá el filtro con mediana para la realización del sistema. Aunque el tiempo de procesado sea mayor, no es una diferencia muy excesiva y consigue una imagen de fondo mucho más fiable, lo que lo hace preferible para la aplicación. Además podemos determinar con mejor precisión los tiempos de actualización.

Los parámetros que usaremos en el filtro con mediana serán los comentados anteriormente para las pruebas en los vídeos, pero a la hora de implementarse en la aplicación final el tiempo de actualización será incrementado para obtener un fondo más constante. Este tiempo será de alrededor de 1 segundo (30 frames) y se subirá el tamaño del buffer a 26.

A partir de esta sección, todas las pruebas se realizarán usando el filtro con mediana como método de substracción de fondo.

6.2. Elección de umbral

Para establecer el mejor valor de umbral para nuestra aplicación, usaremos varios vídeos y escogeremos manualmente los valores que den una máscara de foreground más parecida a la real. La manera ideal de elegir los valores sería mediante la comparación de estas máscaras con una máscara fiable, pero ya que no se dispone de ésta, se hará mediante observaciones.

En la tabla 6.4 se muestran los resultados:

Vídeos	Valores óptimos de umbral
adl-01-cam0_MOD	23 - 26
adl-02-cam0_MOD	24 - 25
adl-03-cam0_MOD	24 - 28
adl-05-cam0_MOD	26 - 28
adl-06-cam0_MOD	27 - 28
adl-08-cam0_MOD	26 - 27
fall-01-cam0_MOD	19 - 27
fall-03-cam0_MOD	19 - 25
fall-04-cam0_MOD	21 - 26
fall-05-cam0_MOD	21 - 26
fall-04-cam0_MOD	23 - 26
fall-10-cam0_MOD	22 - 26

Tabla 6.4: Valores óptimos de umbral

Después de realizar las observaciones se puede ver que los valores más repetidos son los del intervalo $24 - 26$, 9 veces cada uno. El valor escogido final será un umbral de 24.

6.3. Elección de técnica de confirmación de caída

La etapa de confirmación de caída es un paso crucial en la aplicación. Es la última etapa antes de la generación de la alarma y permite distinguir cuando se ha generado un evento por una caída o por otra causa diferente. En la sección 5.9.3 se han visto dos maneras diferentes de establecer la confirmación, más una tercera que corresponde a una mejora de la segunda. La manera en la que se comprobará cuál usar de las dos consistirá en aplicarlas en la colección de vídeos obtenidos y comparar cuál ofrece mejores resultados. Los parámetros usados serán comunes a ambos, exceptuando las variables que corresponden a cada técnica. Los parámetros y vídeos elegidos serán descritos con mayor detalle en el siguiente apartado. Esta sección se centrará en evaluar las diferencias entre usar el valor máximo de los histogramas de proyección vertical o la distancia de Bhattacharyya como técnicas de confirmación.

Para el primer método se ha establecido un ratio máximo de un 25 %, mientras que para la distancia de Bhattacharyya se usará un límite de 0.6. En la tabla siguiente se muestran los resultados obtenidos con ambas técnicas.

	VHPmax	Bhattacharyya
Aciertos	69 %	69 %
Falsos positivos	21 %	20 %
Falsos negativos	10 %	11 %

Tabla 6.5: Comparación de técnicas de confirmación

Como se puede ver, ambos métodos ofrecen resultados similares, obteniendo el mismo porcentaje de aciertos. El método de máximos de los VPH tiene un porcentaje de falsos positivos mayor que el Bhattacharyya, mientras que pasa al contrario en el caso de los falsos negativos. En otras palabras, la comparación de máximos tiende a realizar más confirmaciones que la distancia de Bhattacharyya.

A primera vista se podría pensar que no hay demasiadas diferencias, sin embargo, es necesario estudiar cada vídeo para entender cuáles han sido los fallos. Para empezar, un 2 % de los falsos positivos de la segunda técnica ocurren en vídeos donde ha ocurrido una caída que ha sido confirmada. Es decir, que en realidad en esos vídeos sí se ha hecho una confirmación correcta, pero en algún punto del vídeo se ha vuelto ha hacer una confirmación equivocada. Otro punto a destacar es que un 13 % de los vídeos corresponden a situaciones en las que el sujeto se tumba en un colchón y al cabo del tiempo se levanta. Este hecho provoca que al cabo del tiempo, la persona pase a formar parte del background. En este tipo de vídeos los motivos de los errores son diferentes en cada caso. La comparación de máximos falla en todos estos vídeos al considerar la acción de tumbarse como caída. Por otro lado, la distancia de Bhattacharyya evita estas confirmaciones, pero fracasa al confirmar cuando se levanta, debido a los fallos del background comentados anteriormente. En estos vídeos podemos ver que la distancia de Bhattacharyya, realiza una mejor distinción entre caídas y acciones como tumbarse y solamente falla cuando se tiene un fondo erróneo.

Por estas razones se elegirá la distancia de Bhattacharyya y se le añadirá la mejora mencionada en el 5.9.3.1 para mejorar los resultados, los cuales se presentan en el siguiente apartado.

6.4. Resultados finales

Después de elegir las técnicas y establecer los valores de las distintas variables del algoritmo, se procederá a obtener los resultados del algoritmo finales. Se utilizarán los vídeos del segundo dataset, y se seleccionarán únicamente los que empiecen con el frame inicial sin personas en la escena. Esto hace un total de 83 vídeos, siendo 31 de caídas y 52 de actividades cotidianas. Los parámetros usados en la aplicación serán los siguientes:

	Parámetros		
Filtro Paso Bajo	blurKernel	2	
Umbral	threshLim	24	
	histLim	4000	
Median Filter	buffSize	26	
	nmax	30	
Deteccion de sombras	N	4	
	Lncc	0.99	
	Lstd	0.05	
	Llow	0.6	
1º Filtrado de área de contorno	areaMin	700	
	areaMax	20000	
Union de silueta	distXCenMax	20	
	distYBoxMax	25	
Limitador de centroide	limX	25	
	limY	15	20 (Lecture room)
2º Filtrado de área de contorno	areaMin2	1500	900 (Lecture room)
	areaMax2	20000	
Filtrado de área de bounding box	areaMinBox	1500	900 (Lecture room)
Desactivacion de deteccion	areaConfLim	45000	
Detección y confirmación de caída	angleLim	45	
	aspectRatioLim	1	
	vphNFrames	25	
	vphBhatLim	0.6	

Tabla 6.6: Tabla de parámetros

Como se verá en al apartado de instalación del dispositivo, las variables de limitación del centroide y filtrado de área serán distintas según la disposición de la habitación de la escena. Por esa razón, en los experimentos se estableció unos parámetros distintos en los vídeos de un escenario en concreto para conseguir mejores resultados.

Los resultados más importantes para determinar el éxito del algoritmo son el porcentaje de detecciones acertadas y su velocidad de procesamiento.

6.4.1. Tiempo de procesamiento

Primero, se hará un cálculo de los tiempos de ejecución de cada etapa del programa, comparando los datos obtenidos en la placa Raspberry Pi frente a los obtenidos por un ordenador convencional. Los valores mostrados son los tiempos máximos de cada etapa. Los datos de la tabla están en ms, mientras que los de la gráfica están en segundos.

	Ordenador	Raspberry Pi	
Obtención de imagen	0,5868	8,1679	ms
Actualización background (Median filter)	21,5998	73,9335	ms
Ajuste de resolución	0,0470	0,3162	ms
Conversión BGR - GRAY	0,3471	0,9555	ms
Filtro paso bajo	0,3178	3,2418	ms
Sustracción de fondo	0,0688	0,5087	ms
Histograma para elección de umbral	0,2372	0,9092	ms
Algoritmo baja saturación	2,3083	8,2852	ms
Umbralización	0,2339	1,1072	ms
Detección de sombras	3,5350	13,4808	ms
Transformaciones morfológicas	0,4722	12,2034	ms
Actualización buffer de background	4,9925	8,2968	ms
Actualización buffer de VPH	0,3819	1,3042	ms
Extracción de parámetros y ajustes	1,1576	3,0337	ms
Distancia de Bhattacharyya	8,5531	23,9935	ms

Tabla 6.7: Tiempos de procesamiento en cada etapa

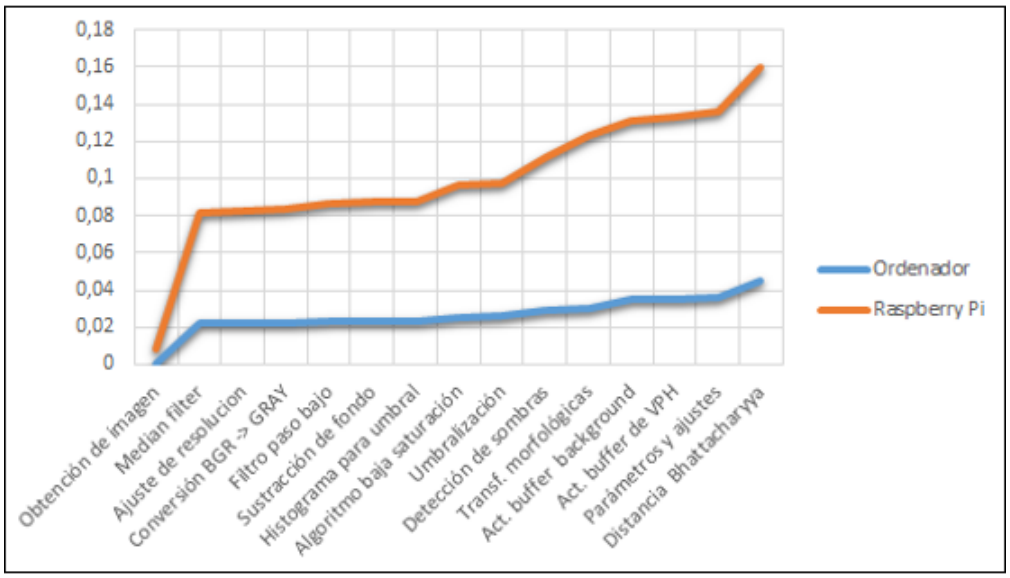

Figura 6.2: Tiempos de procesamiento en cada etapa

A continuación, se calculará el rendimiento final del dispositivo, es decir, el tiempo medio que tarda el programa desde que obtiene un fotograma al siguiente, pasando por todas las etapas del algoritmo explicadas anteriormente. Para calcular el tiempo de procesado por frame, se probará directamente en la placa Raspberry Pi. No se mostrará la imagen por pantalla, de esta forma, se ahorrará tiempo por frame. Se han escogido un total de 15 vídeos para realizar la prueba. El tiempo medio total se obtendrá de la manera siguiente. Se calculará el tiempo que tarda

en acabarse cada vídeo y después se dividirá por el número de frames del vídeo. Cabe destacar que el tiempo medio se mantiene similar en cada vídeo. Más tarde, se hará una promedio de todas las medias y se obtendrá un valor final. Este valor es el siguiente:

Tiempo medio de procesado (seg/frame)
0,0846

Tabla 6.8: Tiempo de procesamiento del programa

Para que un sistema como el que se presenta funcione en situaciones de la vida real, es necesario contar con un programa que sea capaz de trabajar en tiempo real. Estos resultados no nos permitirían tener un sistema de tiempo real (11,82 frames/seg), por lo que necesitan ser mejorados. Sin embargo, son relativamente buenos comparados con otros algoritmos desarrollados similares [4]. Una opción para mejorar la velocidad de procesado podría ser el uso de un lenguaje de programación más rápido o el uso de computación en paralelo. También, otras soluciones podrían ser una reducción de la resolución de las imágenes y una mejor optimización del código.

6.4.2. Resultados de detección de caídas

Los resultados obtenidos son los siguientes:

	nº vídeos	**Porcentaje**
Aciertos	62	75 %
Falsos positivos	12	14 %
Falsos negativos	9	11 %

Tabla 6.9: Resultados finales del algoritmo

Como se puede ver, el uso de la técnica descrita en 5.9.3.1 permite obtener una mejora de un 6 % respecto a no usarla. Igual que en el apartado anterior, será preciso estudiar a fondo los resultados, y entender cuáles han sido los problemas del sistema para poder mejorarlo.

Para empezar, existe una colección de 11 vídeos de larga duración donde la persona se tumba y se vuelve a levantar. En estos vídeos la persona acaba siendo parte del fondo, por lo que al levantarse aparecen fallos y se produce la confirmación. De estos vídeos, solo en 3 son evitadas las confirmaciones, lo que quiere decir que solamente estos vídeos constituyen casi un 67 % de los fallos por falsos positivos.

Por otro lado, a la hora de detectar caídas, para el sistema realmente solo existen 2 tipos de caídas. Éstas son las caídas laterales y las que se realizan hacia delante o hacia atrás con respecto a la cámara. Según sean estas caídas, el aspect ratio y ángulo de caída (parámetros elegidos de detección de caída) tendrán una

forma u otra. De los 31 vídeos con caídas, 11 son caídas del segundo tipo. Según está diseñado el algoritmo, las caídas no laterales son difíciles de detectar, por lo que no se han confirmado 8 caídas de este tipo. Es decir, estos vídeos suponen el 89 % de los falsos negativos.

Por último, los 5 últimos falsos positivos y negativos se han producido por fallos de creación de máscara. Estos problemas se han debido a diversos motivos:

- Cuando la persona lleva ropa de la misma intensidad que el fondo, la máscara no detecta bien la silueta.
- Cuando una persona se encuentra en el mismo sitio por un periodo largo de tiempo, la persona pasa a formar parte del background, por lo que al alejarse de la zona, la máscara detectará a la persona y el área donde estaba antes.
- Al mover objetos, sobre todo si son grandes (sillas, lámparas, etc.), son detectados por la máscara e interfieren en la búsqueda de contornos.
- Baja calidad de la cámara.

Es necesario mencionar que las pruebas se han hecho con vídeos grabados de una colección de internet, por lo que no se ha podido comprobar el efecto que ha tenido la cámara escogida sobre los resultados. Sin embargo, experimentaciones posteriores han detectado algunos problemas. La cámara no tiene mucha calidad y ocasiona que en entornos con poca luz aparezca mucho ruido en las imágenes. Aparte, tiene demasiado zoom, lo que no permite realizar una buena panorámica de la habitación.

En resumen, estos parámetros nos han dado unos resultados finales buenos, sin embargo, son insuficientes para constituir un sistema de detección por sí solo. El 29 % de las caídas no han sido detectadas, mientras que un 23 % de los vídeos con actividades cotidianas han tenido falsos positivos. Por ello, serán necesarios varios ajustes y añadidos que ayuden a subir el porcentaje de aciertos. Entender estos fallos ha servido para poder crear una lista de mejoras, que suplirán esos errores y servirán para perfeccionar el sistema, los cuales se presentarán en el capítulo 11.

Capítulo 7

Instalación del dispositivo

Una vez que se ha creado el programa y ha sido implementado en la placa Raspberry Pi, el siguiente paso será la instalación del dispositivo en la vivienda del cliente. Sin embargo, antes de ser instalado, se deberán tomar en cuenta algunas consideraciones y recomendaciones para que la aplicación funcione mejor.

7.1. Requisitos iniciales de la instalación

El primer uso del dispositivo se deberá hacer siguiendo una serie de instrucciones. Si no se llevan a cabo, el sistema podría no funcionar. Estos son los requisitos iniciales:

1. En primer lugar, ya que la alarma se envía a través de internet, será necesario que la vivienda de la persona mayor tenga instalado un sistema de conexión a internet. La conexión puede ser por ethernet o por WiFi, al contar la placa con los dos tipos de periférico.

2. Se introducirán los datos del destinatario de la alarma en la aplicación, para que pueda ser avisado.

3. La cámara deberá situarse en un lugar donde no pueda ser movida, es decir, se deberá instalar en un sitio fijo para que no se produzcan desencuadres del background respecto a las imágenes del primer plano.

4. Antes de iniciar la aplicación, será necesario introducir los valores de las variables de filtrado de área por contorno y bounding box, y límites del centroide. Las distribuciones de cada habitación son distintas y precisan diferentes parámetros según sean éstas y esté colocada la cámara. Por ejemplo, en los escenarios donde la cámara se encuentra más alejada, los tamaños mínimos de los filtros de área deberán ser menores, o también, si hay una pared a un lado de la escena, no será necesario limitar esa zona.

Una vez se hayan seguido estos pasos, simplemente se iniciará la aplicación. Ésta creará un fondo inicial en 15 segundos, durante los cuales no se detectarán caídas. Después se ejecutará la aplicación de forma normal.

7.2. Recomendaciones

La posición de la cámara juega un papel muy importante a la hora de detectar las caídas. Una mala elección puede suponer la diferencia entre un sistema completamente funcional o uno ineficaz. A continuación se muestran algunas recomendaciones:

- Para conseguir una buena visión de la escena, la cámara deberá estar situada a una distancia elevada, lo más cerca posible de la pared para que recoja la habitación al completo. Esta distancia deberá ser de unos 1.5-2.5 metros aproximadamente. Si se coloca a una distancia inferior o superior, pueden no detectarse del todo bien las caídas. Es recomendable también, ser usado en estancias amplias.

- Se pondrán en lugares donde existan las menos oclusiones posibles. Si existen muchas oclusiones se formarán fallos en la máscara y el programa no funcionará.

- Se intentará evitar ser colocadas en lugares donde se graben ventanas y focos de luz. Si se recoge una fuente de luz de forma directa, provocará mucho ruido en la substracción del fondo, apareciendo positivos erróneos en la máscara. Además, se intentará omitir la grabación de zonas que puedan causar reflejo, como espejos y cristales.

En este capítulo se han dado varias pautas para que el dispositivo pueda funcionar correctamente. Obviamente, cada estancia es distinta y no será siempre posible seguir estas recomendaciones. Por ello, habrán viviendas donde no sea posible instalar este tipo de sistema o simplemente funcione peor.

Capítulo 8

Planificación

Para alcanzar los objetivos del proyecto, éste se ha dividido en las siguientes fases, con el objetivo de organizar mejor el documento y la carga de trabajo.

- **Fase 1 - Estudio de la tecnología a utilizar**

  Para comenzar el proyecto, el primer paso ha consistido en buscar información sobre el funcionamiento del software y hardware a utilizar. Esto es, principalmente, información sobre la placa Raspberry Pi, el lenguaje Python, y la librería OpenCV, además de las librerías adicionales.

- **Fase 2 - Obtención de material**

  En este punto se incluye la compra del material y la elección de escenas.

  En cuanto al material, se buscaron varias marcas y tipos de cada componente para seleccionar los componentes con una buena relación de calidad y precio.

  La elección de escenas se divide en dos etapas. Primero se utilizó una colección proporcionada por el tutor, para realizar las pruebas. Sin embargo, finalmente se determinó que esta colección no servía para conseguir una buena experimentación, por lo que en una segunda etapa se buscaron nuevos dataset que sirvieran para el propósito del proyecto.

- **Fase 3 - Estudio de la bibliografía de técnicas de visión artificial**

  Una vez adquiridos los conocimientos necesarios para hacer funcionar los componentes y el software del sistema, el siguiente paso fue la búsqueda de información relacionada con las técnicas de visión artificial, que han permitido diseñar el programa.

- **Fase 4 - Estudio sobre técnicas de generación de alarma**

  Para crear la alarma, se tuvo que buscar información sobre las diferentes opciones de generación de alarma. Finalmente se optó por el aviso por correo.

- **Fase 5 - Desarrollo del código**

  Con la bibliografía ya estudiada, se procedió a la escritura del código. Esta fase se hizo en paralelo con la mayoría de fases del proyecto.

- **Fase 6 - Experimentación en ordenador**

  En esta etapa se probó el código en las colecciones de vídeos y se extrajeron los resultados, usando un ordenador personal. Esta fase, al igual que el desarrollo del código, se realizó durante gran parte del proyecto. Las pruebas y la programación han ido en paralelo, con el fin de asegurar el buen funcionamiento de programa.

- **Fase 7 - Configuración Raspberry Pi**

  A la vez que se desarrollaba el código, fue necesario la configuración de la Raspberry Pi. Primero se descargó el sistema Raspbian en una tarjeta microSD, para instalarlo en la placa y después se instaló todo lo necesario para el funcionamiento del código.

- **Fase 8 - Experimentación en Raspberry Pi**

  Una vez se consiguió desarrollar el código en el ordenador y se configuró la Raspberry Pi, la última etapa del proyecto consistió en probar el programa en la placa para extraer los resultados finales.

- **Fase 9 - Elaboración de la memoria**

  La última fase consistió en la documentación del proyecto.

A continuación, se muestra una tabla con las etapas del proyecto, indicando las fechas aproximadas de comienzo y final de cada tarea, y su representación en un diagrama de Gantt.

Tarea	Inicio	Fin
Estudio de la tecnología	15/10/2016	14/03/2017
Obtención de material	20/11/2016	03/08/2017
Búsqueda de componentes	20/11/2016	30/01/2017
Búsqueda 1º dataset	18/03/2017	19/03/2017
Búsqueda 2º dataset	02/08/2017	03/08/2017
Estudio visión artificial	10/03/2017	08/08/2017
Estudio generación de alarma	04/09/2017	14/09/2017
Desarrollo del código	15/04/2017	14/09/2017
Experimentación en ordenador	17/04/2017	14/09/2017
Configuración Raspberry Pi	15/06/2017	21/06/2017
Experimentación en Raspberry Pi	13/09/2017	15/09/2017
Elaboración de memoria	15/07/2016	24/09/2017

Tabla 8.1: Tareas y fechas de planificación

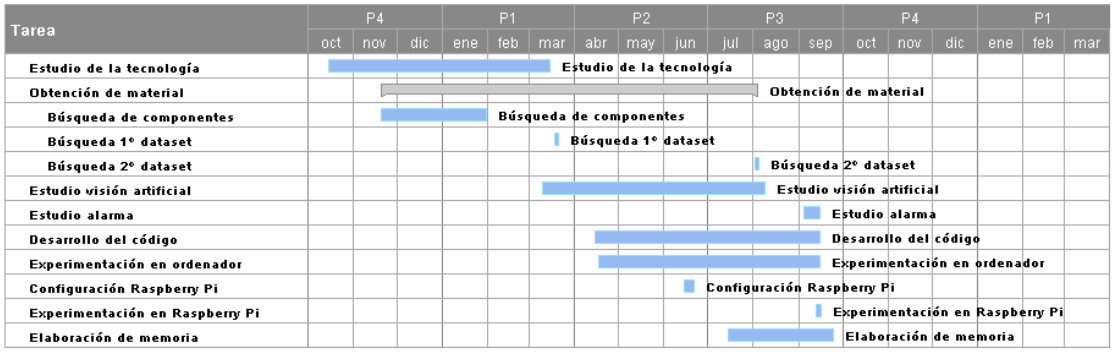

Figura 8.1: Diagrama de Gantt

Capítulo 9

Marco regulador

En este capítulo se estudiarán las implicaciones legales que supondría instalar un sistema de este tipo en una vivienda. Para empezar, se considerará que el producto será comercializado, ya que de no ser así, no sería necesario cumplir ciertas normativas que se mencionarán a continuación. Además, se considerará que el proyecto se desarrolla en Europa, y más concretamente en España, por lo que los marcos reguladores de otros países quedarían fuera de este análisis.

El proyecto presentado implica la captación de imágenes del interior de viviendas privadas. Aunque la videovigilancia del dispositivo no es continua, en el sentido de que lo grabado no se envía a ninguna fuente externa, ni es almacenado, la detección de las caídas sí supone el envío de imágenes. La captación de imagen es un dato de carácter personal y por tanto, está regulado por la normativa de protección de datos. Es necesario aclarar que la mera captación de imágenes del interior de una vivienda propia no infringe la normativa de protección de datos, ya que se considera que se realiza en el ejercicio de una actividad personal o doméstica. Sin embargo, si la cámara capta la imagen de otras personas, este hecho sí es objeto de regulación y es necesario cumplir con una serie de garantías recogidas en la Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal. [34].

Por otro lado, dependiendo del destinatario de las imágenes, también será necesario la aplicación de otras normativas. Si el destinatario es un familiar o conocido del titular, no sería necesario aplicar ninguna norma adicional. Sin embargo, si el encargado de recibir las imágenes es alguna organización de salud pública, el tema varía. En España, el concepto de vigilancia en salud pública está regulado con la Ley 33/2011 General de Salud Pública (LGSP) [35].

Al ser un dispositivo electrónico y eléctrico, será por tanto, también objeto de la normativa nacional sobre compatibilidad electromagnética, material eléctrico, y equipos radioeléctricos. Estas normas están recogidas en el Real Decreto 186/2016, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos [36], el Real Decreto 187/2016, por el que se regulan las exigencias de seguridad del material eléctrico destinado a ser utilizado en determinados límites de tensión [37] y el Real Decreto 188/2016, por el que se aprueba el Reglamento por el que se establecen los requisitos para la comercialización, puesta en servicio

y uso de equipos radioeléctricos, y se regula el procedimiento para la evaluación de la conformidad, la vigilancia del mercado y el régimen sancionador de los equipos de telecomunicación [38], respectivamente.

Por último, se debe mencionar que el proyecto no pretende crear ninguna patente, ya que uno de los objetivos es que el proyecto sea de código abierto. Por ello, debería tener una licencia de código libre, como puede ser GPL, ofrecida por la Free Software Foundation (FSF).

Capítulo 10

Entorno socio-económico

10.1. Presupuesto

En esta sección, se detallará la estimación de los costes de la elaboración del proyecto. Para su fácil comprensión, se presentan los costes en diferentes apartados:

- Coste de personal
- Coste de software y hardware
- Costes indirectos y amortizaciones

10.1.1. Coste de personal

En esta subsección se detallan las personas involucradas en el proyecto, así como la cantidad de tiempo invertido y el precio por hora.

Código	Descripción	Unidad	Cantidad	Precio	Precio Total	
1.1	**Ingeniero industrial especializado en electrónica y automatización.** Ingeniero industrial con conocimientos básicos en programación de visión por computador, programación en Python y circuitos.	Mes/ 2h/día	9	375,00	3.375,00	€
1.2	**Doctor Ingeniero industrial.** Tutor, amplios conocimientos en visión por computador y redes neuronales. Enrique Pelayo Campillos.	Horas	10	60,00	600,00	€
				Coste Total	4.350,00	€

Tabla 10.1: Coste de personal

10.1.2. Coste de software y hardware

En este apartado se calcula el coste del material empleado. Todo el software utilizado ha sido software libre, por lo que no existen gastos de este tipo.

Código	Descripción	Cantidad	Precio Ud.	Precio Total	
2.1	**Raspberry Pi 3 Model B+.** Microcontrolador usado como controlador principal del sistema.	1	39,95	39,95	€
2.2	**Carcasa para Raspberry Pi.** Carcasa oficial de la Raspberry Pi 3 Model B+.	1	4,95	4,95	€
2.3	**Cámara web Selecline 862050.** Cámara web que capta las imágenes.	1	7,90	7,90	€
2.4	**Cargador micro-USB - 2,1 A.** Fuente de alimentación del sistema.	1	11,25	11,25	€
2.5	**Tarjeta MicroSD.** Tarjeta MicroSD SanDisk Ultra 16GB, clase 10.	1	6,99	6,99	€
2.6	**Teclado inalámbrico Selecline 855289.** Teclado para controlar la placa Raspberry Pi.	1	9,90	9,90	€
2.7	**Cableado conexiones multimedia.** Cable VGA y conversor HDMI-VGA.	1	5,13	5,13	€
			Coste Total	86,07	€

Tabla 10.2: Coste de hardware y software

10.1.3. Costes indirectos y amortizaciones

Aquí se detallan los costes indirectos y una estimación de los costes de amortización de los equipos usados.

Según la nueva Ley aprobada el 27 de Noviembre de 2014 con fecha de entrada en vigor el 1 de Enero de 2015 del Impuesto sobre Sociedades, el coeficiente lineal máximo de amortización para equipos electrónicos destinados a procesos de información es de un 25 % anual (en un total de 4 años) [39].

Código	Descripción	Precio	Coeficiente Amor.	Amor. Mes	Nº Meses	Precio Total	
3.1	**Ordenador ASUS VivoBook S200E.** Ordenador usado para realizar proyectos y prácticas.	295,59	25 %	1,54	9	13,86	€
3.2	**Gastos Internet.** Consumo de red móvil.					20,00	€
3.4	**Gasto de luz.** Gasto de luz y baterías.					56,00	€
					Coste Total	89,86	€

Tabla 10.3: Costes indirectos y amortizaciones

10.1.4. Coste total

En este último apartado, se incluye el coste total suponiendo que el proyecto hubiese sido realizado por una empresa:

Partida	Descripción	Precio Total	
1	Coste personal	4.350,00	€
2	Coste software y hardware	86,07	€
3	Costes indirectos y amort.	89,86	€
Total		4.525,93	€
	IVA (21 %)	950,44	€
	Coste Total	5.476,38	€

Tabla 10.4: Coste total

Por tanto, el coste final del proyecto es de 5.476,38 €.

10.2. Impacto socio-económico

El proyecto presentado se encuentra en una fase de investigación, por lo que como se ha comentado anteriormente, este producto no podría ponerse al mercado en la situación actual. Sin embargo, se realizará un pequeño análisis sobre los costes y beneficios que supondría fabricar un producto de este tipo según su estado actual, comparado con dispositivos similares.

Aunque las investigaciones en los sistemas de detección de caídas están teniendo un gran avance, actualmente no existen demasiados dispositivos en el mercado con los que hacer comparaciones. Por tanto, para este análisis se tendrán en cuenta sistemas con características y funciones similares, como pueden ser los sistemas de telealarma y de videovigilancia. Este tipo de dispositivos no realizan necesariamente detecciones de caídas, pero son comparables en cuanto a componentes y servirán para tener una estimación de costes.

El primer tipo de sistema ofrece precios y opciones muy variadas. Se pueden encontrar kits de marcadores telefónicos con botón de pánico, con precios que oscilan desde los 79 € hasta los 200 € [40–42]. También hay opciones más completas, que incluyen monitorización del usuario, con cuotas mensuales de unos 13 € al mes [43]. Aparte de los precios comentados, también hay que incluir los gastos provocados por el uso de la línea telefónica fija o GSM que producen el aviso, según el tipo de producto.

En cuanto a los sistemas de videovigilancia, igualmente existen numerosos tipos de kits y precios, determinados principalmente por el número de cámaras incluidas y sus prestaciones, así como los servicios que éstas ofrecen [44]. Las opciones más económicas rondarían los 450-550 €, mientras que los kits que ofrecen más funciones, como análisis de vídeo, detección de movimiento, alerta con imágenes y/o vídeo, etc., alcanzan precios de 550-800 €. Además, muchos de estos sistemas requieren de una suscripción mensual para poder aprovechar todas esas funciones.

Para determinar el precio que supondría la venta del dispositivo desarrollado, se procederá de la forma siguiente. Para empezar se establecerá un precio de fabricación del dispositivo. Los componentes usados no compondrían los elementos

finales del dispositivo, por lo que el precio será una estimación tomando a estos como referencia. El coste propuesto será de unos de unos 80 €, resultado de la mejora de ciertos componentes (principalmente la cámara).

En un futuro, la idea del proyecto es hacer un sistema autónomo, es decir, que no requiera la supervisión de ningún técnico para hacerlo funcionar. Sin embargo, la instalación y mantenimiento del dispositivo actualmente depende de una persona que sepa manejarlo. Por esa razón, se incluirá en el coste total los servicios de un técnico. El salario de un técnico instalador se estimará de 1200 €. Este coste se dividirá entre un número estimado de dispositivos vendidos al mes (30 dispositivos, uno y medio por cada día laborable). Este coste haría un total de 40 € por dispositivo.

Por último, se considerarán unos beneficios del 30 % respecto al coste total.

A continuación, se mostrará la estimación de los costes y beneficios del producto. Para crear el presupuesto se han tomado en cuenta los siguientes aspectos:

- Todos los precios irán en euros.
- Ne se tomarán en cuenta impuestos o similares.

10.2.1. Costes y beneficios

Partida	**Descripción**	**Precio Total**	
1	Coste personal	40,00	€
2	Coste componentes	80,00	€
Total		120,00	€
	Beneficio del 30 %	36,00	€
	Coste Total	156,00	€

Tabla 10.5: Costes y beneficios

Según el presupuesto, el coste total de fabricación del dispositivo ascendería a los 120,00 €. Si se quiere conseguir un beneficio del 30 %, se obtendrán unos ingresos de 36,00 € por unidad. El precio final de venta del dispositivo será, por tanto, de 156,00 €.

Naturalmente, este presupuesto es solo una estimación y variará dependiendo de diferentes factores. Los factores principales serán el coste de componentes y del personal. El coste de los componentes puede variar, disminuyendo si se quiere adquirir un producto más económico, o aumentando si se quiere crear un sistema más competente (mejores cámaras y sistema de procesamiento, mayor número de cámaras, etc.). Además, el precio del personal es difícil de estimar, y dependerá mayormente de los salarios de la empresa que comercialice el producto y de la amortización de éste. Otro factor será el beneficio que se quiera obtener, el cual

también dependerá de las preferencias de la empresa.

Por último, no se ha tomado en consideración ningún tipo de impuesto, por lo que dependiendo del país donde se venda, el coste total se podría ver incrementado.

Como se puede observar, el precio final del dispositivo, respecto a los demás productos comparados resulta económico, y entra dentro de los límites esperados, teniendo en cuenta las funciones que presenta. Además, al contrario que algunos de los dispositivos mencionados, no requiere de ningún tipo de suscripción que pueda incrementar el precio.

Capítulo 11

Mejoras futuras

En el proyecto se ha desarrollado un algoritmo que ha permitido crear un sistema competente de detección de caídas. Sin embargo, todavía existe un amplio margen de mejora. Gracias a las experimentaciones se han podido encontrar los puntos fuertes de nuestro dispositivo, y corregir los débiles. A continuación, se enumerarán distintas propuestas que pueden ayudar a perfeccionar el sistema.

- **Múltiples cámaras por habitación:** La utilización de más de una cámara por habitación posibilitaría poder grabar el entorno desde varios ángulos diferentes. De esta forma se solucionaría el problema de las caídas hacia delante y atrás respecto a la cámara. La utilización de dos cámaras por habitación sería suficiente para captar los dos tipos de caídas.

  Esta propuesta tiene el inconveniente de requerir el doble, o más, tiempo de procesamiento, al trabajar con múltiples vídeos a la vez. Por ello, debería usarse una segunda placa de procesamiento, incrementando de esta forma el coste. Otra solución podría ser bajar la resolución de los frames para trabajar con menos datos.

- **Sistema multicámara:** El proyecto desarrollado sólo sirve para controlar una sola habitación, por lo que, si se quiere obtener un sistema más seguro para el usuario, será conveniente instalar un sistema multicámara que recoja el mayor número de zonas de la vivienda. Como se comenta en la propuesta anterior, lo ideal sería la instalación de un par de cámaras por cuarto.

  Para solucionar el problema del tiempo de procesado y no incrementar demasiado el coste del sistema, una opción podría ser la utilización de sensores de movimiento, que activasen los pares de cámaras cuando la persona entre a la habitación.

- **Cámaras IP:** Para evitar instalaciones complejas y el uso excesivo de cableado, una propuesta es la utilización de cámaras IP. Estas cámaras no necesitan estar conectadas a la placa. Envían las imágenes a un servidor a través de redes IP como redes LAN, WAN e internet, al que accederá la placa para realizar los análisis. De esta forma, las únicas conexiones serán

mediante ethernet o por WiFi. Además, añaden la posibilidad de vigilancia remota.

- **Algoritmo de seguimiento de objetos:** Debido a las segmentaciones de la silueta producidas por los errores de la máscara, la aplicación no es capaz de distinguir cuando un cúmulo de contornos son parte del mismo objeto o son distintos. Una forma de arreglar este problema sería la implementación de un algoritmo de seguimiento de objetos, que sea capaz de predecir las formas y posiciones de las personas.

- **Mejor calidad de componentes:** Un factor también importante reside en los propios componentes del dispositivo. Una cámara mejor podrá obtener imágenes de más calidad, evitando muchos fallos en las máscaras. La cámara adquirida sirve como un primer acercamiento a la creación del dispositivo, pero el sistema final precisará de una cámara con mejores prestaciones. Además, tener un equipo con mayor capacidad de procesamiento permitirá conseguir tiempos mucho menores y la implementación de algoritmos que requieran más carga computacional y memoria. La parte negativa de esta mejora es el aumento de precio que supondrían estos componentes.

- **Diferentes tipos de cámara:** Mediante la combinación de una cámara convencional con otro tipo de cámaras, como pueden ser cámaras infrarrojas o de profundidad, se podrían conseguir resultados mucho mejores, al suplir carencias que pueden ser subsanadas con estos tipos de sensores.

- **Imágenes a color:** Todo el desarrollo del proyecto se ha hecho haciendo uso de imágenes en escala de grises, principalmente con el fin de mejorar los tiempo de procesamiento del programa y reducir la cantidad de memoria precisada. Este hecho ha limitado ciertos algoritmos que hacen uso de imágenes a color, que podrían haber funcionado mejor que los implementados. Mediante una mejora del sistema de procesamiento, se podrían usar este tipo de imágenes, pudiendo hacer uso de algoritmos más eficaces.

Capítulo 12

Conclusiones

En este capítulo se reflejarán las conclusiones generales obtenidas después de haber concluido el proyecto.

Este trabajo de fin de grado nació con la idea de crear un dispositivo que sirviera de ayuda para personas en riesgo, en especial para personas mayores, haciendo uso de la visión artificial. El riesgo de caídas para este segmento de la población es enorme, siendo una de las causas que provocan más muertes y accidentes al año. Por esa razón, se consideró una buena idea enfocar el proyecto en el diseño y fabricación de un dispositivo que fuera capaz de detectar esas caídas y mandar una alarma que pudiera socorrer a la víctima. Este sistema, además debería ser un producto que sirviera para un ámbito comercial, y que no supusiera un gran coste para los clientes.

Para poder crear este dispositivo fue necesario hacer un análisis general de los diferentes métodos y tecnologías disponibles. Se llegó a la conclusión de que no hay una tecnología mejor que otra, si no que cada una tiene sus ventajas y será necesaria su utilización dependiendo del contexto de la detección. Dentro de los algoritmos de visión artificial se tuvo que hacer un estudio exhaustivo de las numerosas técnicas existentes. En cuanto a los algoritmos de detección de personas se optó por hacer uso de algoritmos de substracción de fondo como el filtro con mediana y filtro con mediana aproximado. Este tipo de métodos de formación de fondo han permitido crear máscaras de movimiento que han servido como punto de partida para diseñar el programa. Uno de los mayores retos ha sido la elección de los parámetros, como se ha visto reflejado en las experimentaciones. Las condiciones de cada estancia son distintas, y los parámetros que sirven para un tipo de situación y entorno no siempre son los más adecuados para otros. Debido a que la substracción de fondo no es suficiente para crear una máscara fiable, se han tenido que buscar maneras para refinarla, como la detección de sombras, las transformaciones morfológicas, o el uso de histogramas para la umbralización. Aparte, también han sido necesarios ciertos ajustes que han servido para perfeccionar la máscara con el fin de evitar errores y confirmaciones innecesarias. A la hora de buscar criterios de detección y confirmación, se han usado el aspect ratio y el ángulo de caída en la primera etapa y los histogramas de proyección vertical en la segunda, añadiendo una mejora que ha conseguido mejorar el porcentaje de

resultados. La última etapa ha consistido en la creación de la alarma, para la cual, se ha implementado un sistema de aviso por correo electrónico que permite mandar un aviso instantáneo, acompañado de una imagen de la caída. De esta forma se puede valorar mejor la situación y actuar en consecuencia.

Los resultados obtenidos han sido buenos, pero como ya hemos explicado, están muy lejos de ser los adecuados para ser llevados al ámbito comercial. Por un lado, la velocidad de procesado no es lo suficientemente alta, por lo que no será muy recomendable en entornos de tiempo real. En cuanto al porcentaje de aciertos, aunque ha sido alto, no es lo suficiente para un sistema de este tipo. Mientras que en las caídas laterales respecto a la cámara se consigue un porcentaje muy alto de aciertos, en las caídas no laterales el porcentaje se desploma. Aparte, el sistema falla cuando la persona aparece en escena en el mismo sitio durante mucho tiempo, ya que el fondo se actualiza con ella y provoca detecciones falsas. Para solucionar estos problemas se ha hecho una lista de posibles modificaciones futuras que pueden ser interesantes para conseguir unos resultados mejores, destacando el sistema multicámara, y la mejora de componentes.

Por otro lado, se ha conseguido crear un sistema económico, y que no ha necesitado de grandes y caras tecnologías para funcionar. Además, su instalación es sencilla y solo necesita de un par de requisitos y recomendaciones para funcionar correctamente.

En resumen, este proyecto ha servido como una primera toma de contacto al desarrollo de los sistemas de detección de caídas. Se ha demostrado que se puede conseguir un sistema competente de detección de caídas, aunque con amplio margen de mejora, haciendo uso de tecnologías baratas y que no supongan un gran desembolso para el cliente. Para poder ser desarrollado con éxito, será necesario una segunda etapa donde se investiguen nuevas técnicas para poder ser aplicadas al sistema y mejorarlo. También se deberá hacer una nueva evaluación para determinar si los componentes elegidos son los más adecuados para los nuevos algoritmos.

Anexo A

Código del programa

- **AlgoritmoDeteccion.py:**

```
ï»¿# -*- coding: utf-8 -*-
"""
Created on Fri Sep 22 01:13:22 2017

@author: sergio
"""

import numpy as np
import cv2
import time
from EnvioCorreo import *

# ------------------------------ Clase Persona
    ------------------------------------
class Persona:
    def __init__(self): #tamBuffer: Tamano de los buffer
        # Buffer del centroide
        self.centroid = np.empty((2), int)
        self.centroid[:] = -1
        # Buffer del bounding box
        self.boundingBox = np.empty((4), int) # (x, y, w, h)
        self.boundingBox[:] = -1
        # Lista de los contornos del frame actual
        self.contour = 0
        # Lista de contornos asociados
        self.associated = []

# ------------------------------ Funciones
    ------------------------------------
# Funcion que actualiza el background
def actBackground(buff):
    bufferOrdenado = np.sort(buff, axis = 0)  # Ordenamos elemento a
    elemento del buffer
    bg = bufferOrdenado[int(len(buff)/2)]     # Cogemos el frame de la
    posicion intermedia (mediana)
    return bg
# Funcion que actualiza el buffer
def actBuffer(buff, fr, n):
    buff = np.roll(buff, n, axis = 0)  # Rotamos los frames del buffer
    buff[0] = fr  # Actualizamos el primer frame
    return buff

# ------------------------------ Programa
    ------------------------------------
def main(path):
    if(visualizacion==1):
        pathf = path+'.avi'
    else:
        pathf=path
```

```
45      cap = cv2.VideoCapture(pathf)
47      # Leemos el primer frame
        ret,frame = cap.read()
49      # Si se ha leido bien, iniciamos el programa
        if ret == True:
51          # Imprimos el nombre del video
            if(visualizacion==0):
53              print('Camara')
            else:
55              print(path[23:])
            # ---------------------- Ajuste de resoluciones
        ----------------------
57          print('res original: '+str(cap.get(3))+'x'+str(cap.get(4)))
            width = 320
59          height = 240
            print('res nueva:', width, 'x', height)
61
            # Inicializamos variables
63          # ------------------------ Deteccion de movimiento
        ------------------------
            #                         #### Filtro paso bajo ####
65          blurKernel = 2
            #                          #### Umbralizacion ####
67          threshLim = 24
            histLim = 4000
69          #                 #### Inicializo variables del buffer ####
            buffSize = 26  # Tamano del buffer
71          buffer = np.empty((buffSize, height, width), np.uint8) # Buffer
        vacio
            #       #### Inicializo variables de actualizacion de background
        ####
73          nmax = 30  # Numero de frames por cada actualizacion
            nAct = 0
75          act = 0   # Indica cuando se produce la actualizacion
            # ---------------- Inicializo variables de deteccion de sombras
        ----------
77          # First estimate shadow:
            N = 4 # Valor que construye el tamano del template
79          nccKernel = np.ones((2*N+1,2*N+1), np.uint8) # Template para
        deteccion de sombras
            Lncc = 0.99
81          # Shadow refinement:
            Lstd = 0.05
83          Llow = 0.6
            # ---------------------------- 1o Filtrado de area
        ----------------------
85          areaMin = 700
            areaMax = 20000
87          # ------------------------------ Union de silueta
        ------------------------
            distXCenMax = 20 # Distancia maxima horizontal entre centroides
89          distYBoxMax = 25 # Distancia maxima vertical entre perimetros de
        bounding box
            # ------------------------ Limitador de centroide
        ----------------------
91          limX = 25 # Limite externo horizontal entre las dimensiones del
        frame
            limY = 20 # Limite externo vertical entre las dimensiones del frame
93          # ---------------------------- 2o Filtrado de area
        ----------------------
            areaMin2 = 900
95          areaMax2 = 20000
            # ---------------------- Filtrado de area bounding box
        ---------------------
97          areaMinBox = 900
            # ---------------------- Desactivacion de deteccion
        ---------------------
99          areaConfLim = 45000
            activo = 1
101         # ------------------------ Confirmacion de caida
        ------------------------
```

```
        #                                   #### Deteccion ####
103     angleLim = 45
        aspectRatioLim = 1
105     #                                 #### Confirmacion ####
        vphMaxNFrames = 25 # Numero de frames entre comparacion de VPHmax
107     vphBhatLim = 0.6
        size = 30
109     vphBuff = np.zeros((size, width), int) # Buffer del VPH
        confirmacion = 0 # Variable para determinar la confirmacion
111     # ------------------------ Variables auxiliares
------------------------
        threshImgROI = np.zeros((height, width, 3), np.uint8)
113
        # ------------------ Inicializacion del primer frame
--------------------
115     # Ajustamos resolucion del frame
        frame = cv2.resize(frame, (width, height))
117     # Pasamos el primer frame a escala de grises
        frameGris1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
119     # Aplicamos suavizado para eliminar ruido
        frameGris = cv2.GaussianBlur(frameGris1, (2*blurKernel+1,2*
blurKernel+1), 0)
121     # Anadimos el primer frame a todo el buffer
        for i in range(buffSize):
123         buffer[i] = frameGris
        # Ponemos como background el primer frame
125     background = buffer[0]
        # ---------------- Generacion de background inicial(camara)
---------------
127     # Si se activa la camara se genera un background
        if(visualizacion==0):
129         print('Espere 15 segundos para generar el background')
            print('Despeje la estancia')
131         f = cv2.getTickFrequency()
            t1 = cv2.getTickCount()
133         n = 15 # Variable para contar el numero de sugundos
            t1 = cv2.getTickCount()
135         while(n>0):
                t2 = cv2.getTickCount()
137             ret, frame = cap.read()
                frame = cv2.resize(frame, (width, height))
139             if((t2-t1)/f>=0.5): # Se actualiza el background cada medio
segundo
                    t1 = cv2.getTickCount()
141                 # Disminuyo los segundos
                    n=n-0.5
143                 print(n)
                    # Preparo el frame
145                 frameGris1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                    frameGris = cv2.GaussianBlur(frameGris1, (2*blurKernel
+1,2*blurKernel+1), 0)
147                 # Actualizo el backgound
                    buffer = actBuffer(buffer, frameGris, 1)
149                 background = actBackground(buffer)
                # Mostramos las imagenes
151             cv2.imshow('frame', frame)
                cv2.imshow('background', background)
153             k = cv2.waitKey(25) & 0xff
                # Si se ha pulsado 's', paramos el video
155             if(k == ord('s')):
                    k = 0
157                 print('## PARADO ##')
                    while (k!=ord('s')):
159                     k = cv2.waitKey(0) & 0xff
                    print('## PLAY ##')
161             # Si ha pulsado la letra esc, salimos del bucle
                if k == 27:
163                 break
            print('Background creado')
165         print('--------------------')
        #
-----------------------------------------------------------------------
```

```
167
        while(1):
169         # Leemos el frame
            ret, frame = cap.read()
171         # Si hemos llegado al final del video salimos del bucle
            if ret == False:
173             break

175         # ---------------- Inicializamos la actualizacion
    ----------------
            act = 0
177         # Cada nmax frames se actualiza el background
            nAct = nAct+1
179         if nAct>=nmax:
                nAct = 0
181             act = 1
            # Actualizamos el background
183         if act == 1:
                background = actBackground(buffer)
185
            # ---------------------- Ajuste de resolucion
    ----------------------
187         frame = cv2.resize(frame, (width, height))

189         # ---------------- Inicializamos variables auxiliares
    ----------------
            frame2 = frame.copy()
191         frame3 = frame.copy()
            # Dibujamos los limites del centroide:
193         #cv2.rectangle(frame2, (limX, limY), (width-limX, height-limY),
    (255,255,255), 1)
            # Igualamos el umbral
195         thresh=threshLim
            # Inicializamos aspect ratio y angulo
197         angle = -1
            aspectRatio = -1
199
            # ---------------------- Filtro paso bajo
    ----------------------
201         # Convrtimos el frame a escala de grises
            frameGris1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
203         # Aplicamos suavizado para eliminar ruido
            frameGris = cv2.GaussianBlur(frameGris1, (2*blurKernel+1,2*
    blurKernel+1), 0)
205         # -------------------- Background subtraction
    --------------------
            # Restamos el frame actual al background
207         diff = cv2.absdiff(frameGris, background)
            # ---------------------- Eleccion de umbral
    ----------------------
209         # Histograma de diff
            hist = cv2.calcHist([diff],[0],None,[256],[0,256])
211         # Si el maximo del histograma de diff es mayor que histLim y el
    umbral
            # es menor que su posicion, el nuevo umbral sera la posicion del
213         # histograma donde sea menor que histLim
            if(hist.max()>=histLim):
215             i=hist.argmax()
                while(hist[i]>=histLim or i>=255):
217                 i=i+1
                if(i>=255):
219                 thresh=255
                else:
221                 thresh = i+threshLim
            else:
223             thresh=threshLim
            # ---------------------- Baja saturacion
    ----------------------
225         # Descomponemos el frame actual en el espacio de colores HSV
            frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
227         frameS = frameHSV[:,:,1]
            # Umbralizamos el canal de saturacion para indicar zonas de baja
```

```
 saturacion
229             frameS = cv2.GaussianBlur(frameS, (2*blurKernel+1,2*blurKernel
    +1), 0)
                _, frameSthresh = cv2.threshold(frameS,10,255,cv2.
    THRESH_BINARY_INV)
231             # Aislamos las zonas de baja saturacion de diff
                diffLS = cv2.bitwise_and(diff, diff, mask=frameSthresh) # Zonas
    de baja saturacion
233             diffHS = diff - diffLS # Zonas de alta saturacion
                # ------------------------ Umbralizacion
    ------------------------------
235             # Si la diferencia entre el frame actual y el background es
    mayor que el umbral,
                # el pixel esta en el foreground
237             _, threshImgHS = cv2.threshold(diffHS,thresh,255,cv2.
    THRESH_BINARY)
                _, threshImgLS = cv2.threshold(diffLS,thresh+10,255,cv2.
    THRESH_BINARY)
239             threshImg = threshImgHS + threshImgLS
                # ---------------------- Deteccion de sombras
    ---------------------
241             # Hallamos los puntos extremos de la mascara para crear un ROI
    para consumir menos tiempo
                # Mostrar contorno del ROI
243             threshImgROI = cv2.bitwise_or(threshImgROI, (255, 255, 255),
    mask = threshImg)
                contornosimg = threshImg.copy()
245             _, contornos, hierarchy = cv2.findContours(contornosimg,cv2.
    RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
                # Solo se admitiran contornos grandes
247             listaROI = []
                for c in contornos:
249                 if cv2.contourArea(c) > 700:
                        listaROI.append(c)
251             if(len(listaROI)>0): # Si no hay movimiento, este paso se omite
                    # Hallamos puntos extremos
253                 cGlobal = np.concatenate((listaROI))
                    (x, y, w, h) = cv2.boundingRect(cGlobal)
255                 x, y, w, h = x-(N+1), y-(N+1), w+2*(N+1), h+2*(N+1)
                    if(x<0):
257                     x = 0
                    if(y<0):
259                     y = 0
                    # Dibujo los contornos del ROI
261                 cv2.rectangle(threshImgROI, (x, y), (x + w, y + h), (255, 0,
     0), 1)
                    # Creo los ROI
263                 roi_fg = frameGris[y:y+h, x:x+w]
                    roi_bg = background[y:y+h, x:x+w]
265                 # Convertimos las imagenes en array de 16 bits para que se
    puedan multiplicar
                    fg = np.array(roi_fg,np.uint16)
267                 bg = np.array(roi_bg, np.uint16)
                    # ################## First shadow estimate
    ######################
269                 # Calculamos los parametros:
                    er = fg*bg/256 # Dividimos entre 2^8 para que la funcion cv2
    .filter pueda operar
271                 er = cv2.filter2D(er,-1,nccKernel)*256 # Volvemos a
    multiplicar por 2^8
                    eb = bg*bg/256
273                 eb = cv2.filter2D(eb,-1,nccKernel)*256
                    eb = np.sqrt(eb)
275                 et = fg*fg/256
                    et = cv2.filter2D(et,-1,nccKernel)*256
277                 et = np.sqrt(et)
                    ncc = er/(eb*et)
279                 # Un pixel sera sombra del background si: ncc(x,y) >= Lncc
    and et(x,y) < eb(x,y)
                    # 1er criterio (ncc(x,y) >= Lncc):
281                 maskNcc = cv2.compare(ncc, Lncc, cv2.CMP_GE)
                    # 2o criterio (et(x,y) < eb(x,y)):
```

```
283                 maskE = cv2.compare(et, eb, cv2.CMP_LT)
                    # 1er AND 2o criterio:
285                 maskShadow = cv2.bitwise_and(maskNcc, maskE)
                    maskShadow = cv2.bitwise_and(maskShadow, threshImg[y:y+h, x:
    x+w])
287                 # ###################### Shadow refinement
    ######################
                    ratio = fg/bg
289                 # varianza = std^2 = E[(xi -u)^2] = E[xi^2] - u^2
                    # std = sqrt(varianza)
291                 # Explicacion en https://es.wikipedia.org/wiki/Varianza
                    # E[xi^2]:
293                 ratio2 = ratio*ratio
                    meanXi2 = cv2.filter2D(ratio2,-1,np.ones((3,3))/9)
295                 # u^2:
                    mean = cv2.filter2D(ratio,-1,np.ones((3,3))/9)
297                 mean2 = mean*mean
                    # std^2 = E[xi^2] - u^2:
299                 std = abs(meanXi2-mean2)
                    # std:
301                 std = np.sqrt(std)
                    # Un pixel sera sombra del background si: stdR(x,y) < Lstd
    and Llow <= ratio < 1
303                 # 1er criterio (stdR(x,y) < Lstd):
                    maskStd = cv2.compare(std, Lstd, cv2.CMP_LT)
305                 # 2o criterio (Llow <= ratio < 1):
                    maskLow = cv2.compare(Llow, ratio, cv2.CMP_LE)
307                 maskRatio = cv2.compare(ratio, 1, cv2.CMP_LT)
                    mask2 = cv2.bitwise_and(maskLow, maskRatio)
309                 # 1er AND 2o criterio:
                    maskShadow2 = cv2.bitwise_and(maskStd, mask2)
311                 maskShadow2 = cv2.bitwise_and(maskShadow2, maskShadow)
                    # ################### Final shadow mask
    ########################
313                 foregroundNoShadow = threshImg.copy()
                    foregroundNoShadow[y:y+h, x:x+w] = cv2.subtract(
    foregroundNoShadow[y:y+h, x:x+w], maskShadow)
315                 foregroundNoShadow2 = threshImg.copy()
                    foregroundNoShadow2[y:y+h, x:x+w] = cv2.subtract(
    foregroundNoShadow2[y:y+h, x:x+w], maskShadow2)
317             else:
                    foregroundNoShadow2 = threshImg
319             # ------------------ Transoformaciones morfologicas
    ------------------
                # Aplicamos preprocesado
321             erosion = cv2.erode(foregroundNoShadow2, np.ones((3,1),np.uint8)
    )
                dilation = cv2.dilate(erosion, cv2.getStructuringElement(cv2.
    MORPH_ELLIPSE,(7,7)))
323             # ---------- Operaciones despues de obtener mascara final
    ----------
                # Actualizamos el buffer
325             if act == 1:
                    buffer = actBuffer(buffer, frameGris, 1)
327             # Actualizamos el vph
                # (Han pasado vphMaxNFrames frames desde el comienzo de la caida
    )
329             _, dilation1 = cv2.threshold(dilation,2,1,cv2.THRESH_BINARY)
                vph = dilation1.sum(0)
331             vphBuff = actBuffer(vphBuff, vph, -1)
                # ------------------- Desactivacion de deteccion
    --------------------
333             activo = 1
                if(dilation1.sum() > areaConfLim):
335                 activo = 0
                if(activo == 1):
337                 # ---------------------- Deteccion de blobs
    ----------------------
                    # Copiamos el la imagen procesada para detectar los
    contornos
339                 contornosimg = dilation.copy()
                    # Buscamos contornos en la imagen
```

```
341             im, contornos, hierarchy = cv2.findContours(contornosimg,cv2
    .RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
                # Diferentes colores
343             color = [(0, 255, 0),(0, 100, 100),(0, 0, 255),(100, 0, 100)
    , (0, 100, 100)] # color[co]
                co = 0
345             # ---------- Contornos detectados
    ----------
                n = -1 # Numero del contorno detectado aceptado
347             cIndice = -1 # Numero del contorno detectado
                listaObjAct = [] # Lista de contornos detectados en el frame
     actual
349             for c in contornos:
                    # Actualizacion de numero de contorno total
351                 cIndice = cIndice+1
                    # Eliminamos los contornos mas pequenos
353                 if cv2.contourArea(c) < areaMin or cv2.contourArea(c) >
    areaMax:
                        continue
355                 # Actualizacion de numero de contorno
                    n = n+1
357                 # Inicializamos objeto actual:
                    listaObjAct.append("")
359                 listaObjAct[n] = Persona()
                    # Contornos
361                 listaObjAct[n].contour = cIndice
                    # Bounding box (Obtenemos el bounds del contorno, el
    rectangulo mayor que engloba al contorno)
363                 listaObjAct[n].boundingBox = cv2.boundingRect(c) # (x, y
    , w, h)
                    cv2.rectangle(frame2, (listaObjAct[n].boundingBox[0],
    listaObjAct[n].boundingBox[1]), (listaObjAct[n].boundingBox[0] +
    listaObjAct[n].boundingBox[2], listaObjAct[n].boundingBox[1] +
    listaObjAct[n].boundingBox[3]), color[co], 2) # Dibujamos el rectangulo
    del bounds
365                 # ---------- Elipse
    ----------
                    ellipse = cv2.fitEllipse(c)  # ellipse = ((center),(
    width,height of bounding rect), angle)
367                 cv2.ellipse(frame2,ellipse,color[co],2) # Dibujamos la
    elipse
                    # ---------- Area y centroide
    ----------
369                 m = cv2.moments(c)
                    x0 = int(m['m10']/m['m00'])
371                 y0 = int(m['m01']/m['m00'])
                    listaObjAct[n].centroid = [x0, y0]
373                 cv2.circle(frame2,(int(x0), int(y0)),2,color[co],-1) #
    Dibujamos el centroide
                    # ---------- Actualizacion
    ----------
375                 # Actualizacion del color
                    co=co+1
377                 if(co>=5):
                        co=0
379
                # ---------- Asociacion de contornos
    ----------
381             # Buscamos en el frame actual los objetos asociados
                for n in range(len(listaObjAct)): # Objetos del frame actual
383                 for m in range(n+1,len(listaObjAct)): # Objetos del
    frame actual
                        # ---------Distancia horizontal entre centroides
    ---------
385                     distXCen = abs(listaObjAct[n].centroid[0] -
    listaObjAct[m].centroid[0]) # Restamos la posicion x
                        if(distXCen < distXCenMax): # Distancia x max entre
    centroides
387                         # -------- Distancia vertical entre perimetros
    --------
                            # Comprobamos cual esta arriba o abajo
389                         if(listaObjAct[n].centroid[0] < listaObjAct[m].
```

```
centroid[0]): # n esta arriba
                                distYBox = listaObjAct[m].boundingBox[1] - (
listaObjAct[n].boundingBox[1]+listaObjAct[n].boundingBox[3]) # (x, y, w,
 h)
391                         else: # m esta arriba
                                distYBox = listaObjAct[n].boundingBox[1] - (
listaObjAct[m].boundingBox[1]+listaObjAct[m].boundingBox[3]) # (x, y, w,
 h)
393                         # ------------------ Asociamos los objetos
_____________
                            if(distYBox<distYBoxMax):
395                             listaObjAct[n].associated.append(m)
                # Creamos un lista de los objetos ya asociados
397             listaObjNew = [] # Lista de contornos detectados en el frame
 actual nueva
                listaAux = [] # Lista de contornos para saltar
399             for n in range(len(listaObjAct)): # Objetos del frame actual
                    # Si n esta en la listaAux no se crea un nuevo objeto
401                 exit = 0
                    for la in range(len(listaAux)):
403                     if(n == listaAux[la]):
                            exit = 1
405                         break
                    if(exit == 0):
407                     # Creamos un nuevo objeto
                        listaObjNew.append("")
409                     listaObjNew[len(listaObjNew)-1] = Persona()
                        listaObjNew[len(listaObjNew)-1].associated.append(n)
 # Le asociamos el objeto propio
411                     listaObjNew[len(listaObjNew)-1].associated.extend(
listaObjAct[n].associated) # Igualamos los objetos correspondientes
                        i = 0 # Indice que pasa por la lista de listaObjAct[
n].associated[i]
413                     while(1):
                            # Compruebo que listaObjAct[n].associated no
este vacio
415                         if(i<len(listaObjAct[n].associated)):
                                listaObjNew[len(listaObjNew)-1].associated.
extend(listaObjAct[listaObjAct[n].associated[i]].associated) # Ponemos
la lista del contorno
417                             del(listaObjAct[listaObjAct[n].associated[i
]].associated[0:len(listaObjAct[listaObjAct[n].associated[i]].associated
)]) # Elimino el indice ya asociado
                                listaAux.append(listaObjAct[n].associated[i
]) # Anado el indice ya asociado a la lista auxiliar
419                             i=i+1
                            else:
421                             break

423             # ------------ Calculo de informacion de nuevos contornos
_________
                # Pasamos por los nuevos contornos y recalculamos los datos
425             for n in range(len(listaObjNew)):
                    #Creamos los nuevos contornos
427                 contornoNew = contornos[listaObjAct[listaObjNew[n].
associated[0]].contour] # Asocio contorno propio
                    for i in range(1, len(listaObjNew[n].associated)):
429                     contornoNew = np.concatenate((contornoNew, contornos
[listaObjAct[listaObjNew[n].associated[i]].contour]))
                    # Eliminamos los contornos mas pequenos
431                 if cv2.contourArea(contornoNew) < areaMin2 or cv2.
contourArea(contornoNew) > areaMax2:
                        continue
433                 (x, y, w, h) = cv2.boundingRect(contornoNew)
                    if w*h < areaMinBox:
435                     continue
                    # Bounding box (Obtenemos el bounds del contorno, el
rectangulo mayor que engloba al contorno)
437                 listaObjNew[n].boundingBox = cv2.boundingRect(
contornoNew) # (x, y, w, h)
                    cv2.rectangle(frame3, (listaObjNew[n].boundingBox[0],
listaObjNew[n].boundingBox[1]), (listaObjNew[n].boundingBox[0] +
```

```
        listaObjNew[n].boundingBox[2], listaObjNew[n].boundingBox[1] +
        listaObjNew[n].boundingBox[3]), (50,50,255), 2) # Dibujamos el
        rectangulo del bounds
439                     # ------------------------ Elipse
        ------------------------------
                        ellipse = cv2.fitEllipse(contornoNew)  # ellipse = ((
        center),(width,height of bounding rect), angle)
441                     cv2.ellipse(frame3,ellipse,(50,50,255),2) # Dibujamos la
         elipse
                        # ------------------------ Area y centroide
        ------------------------
443                     m = cv2.moments(contornoNew)
                        x0 = int(m['m10']/m['m00'])
445                     y0 = int(m['m01']/m['m00'])
                        listaObjNew[n].centroid = [x0, y0]
447                     cv2.circle(frame3,(int(x0), int(y0)),2,(50,50,255),-1) #
         Dibujamos el centroide
                        # ------------------------ Angulo
        ------------------------------
449                     angle = ellipse[2] + 90
                        if(angle>180):
451                         angle=angle-180
                        if(angle>90):
453                         angle=180-angle
                        # ------------------------ Aspect ratio
        ------------------------
455                     aspectRatio = listaObjNew[n].boundingBox[3]/listaObjNew[
        n].boundingBox[2]
                        # ------------------------ Deteccion
        ------------------------
457                     # Condiciones de deteccion de caida
                        if((angle<=angleLim and angle>-1) or (aspectRatio<=
        aspectRatioLim and aspectRatio>-1)):
459                         # Condiciones de centroide
                            if((x0>limX and x0<width-limX) and (y0>limY and y0<
        height-limY)):
461                             # Se ha producido una deteccion de caida
                                '''print('Deteccion de caida')'''
463                             # ------------------ Confirmacion de caida
        ---------------
                                # Si se ha detectado la caida se procede a la
        confirmacion
465                             #                 #### Bhattacharyya ####
                                # Calculo distancia de Bhattacharyya
467                             distB = 0
                                for x in range(len(vphBuff[0])):
469                                 distB = distB + np.sqrt((vphBuff[-
        vphMaxNFrames][x]/vphBuff[-vphMaxNFrames].sum())*(vphBuff[0][x]/vphBuff
        [0].sum()))
                                # Confirmacion:
471                             if(distB <= vphBhatLim):
                                    #         #### Bhattacharyya (con VPHmax)
        ####
473                                 # Creo los limites
                                    izqDet = listaObjNew[n].boundingBox[0]-10
475                                 if(izqDet<0):
                                        izqDet=0
477                                 derDet = listaObjNew[n].boundingBox[0]+
        listaObjNew[n].boundingBox[2]+10
                                    if(derDet>width-1):
479                                     derDet=width-1
                                    # Confirmacion:
481                                 if(vphBuff[-vphMaxNFrames][izqDet:derDet].
        max()-vphBuff[0][izqDet:derDet].max() >= 0):
                                        confirmacion = 1
483                                     if(visualizacion != 0):
                                            print('Confirmacion de caida en
        frame', cap.get(1))
485                                     else:
                                            print('Confirmacion de caida a las',
         time.strftime("%H:%M:%S"))
487
```

```
                                        # Guardo imagen de la caida:
489                                     archivo = 'Capturas/Caida '+time.
    strftime("%d-%m-%y %H,%M,%S")+'.png'
                                        guardado = cv2.imwrite(archivo,frame)
491                                     # Si no se ha enviado, no mandamos el
    correo
                                        if(guardado == 0):
493                                         print('No se ha guardado la foto')
                                        else:
495                                         print('Enviando correo')
                                            envio = envioCorreo(archivo)
497                                         if(envio==1):
                                                print('Correo enviado')
499                                         else:
                                                print('Correo no enviado')
501
                                        print('----------------')
503             else:
                    print('Mascara colapsada, espere a que se estabilice la
    camara')
505
                # Mostramos las imagenes
507             cv2.imshow('frame', frame)
                cv2.imshow("Contornos", frame2)
509             cv2.imshow("Contornos corregidos", frame3)
                cv2.imshow("frameGris", frameGris)
511             cv2.imshow("background", background)
                cv2.imshow("diff", diff)
513             cv2.imshow("threshImg", threshImg)
                cv2.imshow("threshImgROI", threshImgROI)
515             cv2.imshow("foregroundNoShadow2", foregroundNoShadow2)
                cv2.imshow("dilation (7,7)", dilation)
517
                # Capturamos una tecla para salir
519             k = cv2.waitKey(25) & 0xff
                # Si se ha pulsado 's', paramos el video
521             if(k == ord('s')):
                    k = 0
523                 print('## PARADO ##')
                    while (k!=ord('s')):
525                     k = cv2.waitKey(0) & 0xff
                    print('## PLAY ##')
527             # Si ha pulsado la letra esc, salimos del bucle
                if k == 27:
529                 break

531     # Si NO se ha leido bien, avisamos por terminal
        else:
533         print("\nNo se ha podido abrir el archivo o la camara")

535     # Liberamos la camara y cerramos todas las ventanas
        cv2.destroyAllWindows()
537     cap.release()

539     print('----------------')

541 ################################ Inicio programa
        ####################################
    # Video a analizar:
543 path = 'Videos_TFG\Fall Detection Dataset/Office/video (2)'

545 # Elegir metodo de visualizacion:
    visualizacion = 1 # 0-Camara, 1-Video
547
    if(visualizacion==0):
549     main(0)
    elif(visualizacion==1):
551     main(path)
```

- **EnvioCorreo.py:**

```
ï»¿# -*- coding: utf-8 -*-
"""
Created on Fri Sep 22 01:13:22 2017

@author: sergio
"""

# importamos la libreria smtplib
import smtplib
# importamos librerias para construir el mensaje
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.image import MIMEImage

# Correos y contrasena
remitente = 'alarmasistcaidas@gmail.com'
contrasena = '********'
destinatario  = '100315049@alumnos.uc3m.es'

def envioCorreo(arch):

    # Establecemos conexion con el servidor smtp de gmail
    try:
        mailServer = smtplib.SMTP() # Creamos variable que gestionara el envio
        mailServer.connect("smtp.gmail.com", 587) # Nos conectamos al servidor de correo saliente de gmail, smtp.gmail.com que esta a la escucha en el puerto 587
        mailServer.ehlo() # Enviamos ehlo() para que nos acepte el protocolo
        mailServer.starttls() # Habilitamos el modo TTL, esto es uno de los metodos de enviar el correo de forma segura, el otro modo seria utilizar SSL
        mailServer.ehlo() # Enviamos ehlo() para que nos acepte la entrada a sesion
        mailServer.login(remitente, contrasena) # Nos identificamos en el servidor
    except:
        print('Error de conexion con el servidor')
        return False

    # Construimos un mensaje Multipart, con un texto y una imagen adjunta
    try:
        mensaje = MIMEMultipart()
        mensaje['From'] = remitente
        mensaje['To'] = destinatario
        mensaje['Subject'] = "Aviso de caida"
        # Adjuntamos el texto
        texto = MIMEText("""Deteccion de caida confirmada. \nSe adjunta fotografia del evento.""")
        mensaje.attach(texto)
        # Adjuntamos la imagen
        file = open(arch, "rb")
        imagen = MIMEImage(file.read())
        file.close()
        imagen.add_header('Content-Disposition', 'attachment; filename = foro'+arch)
        mensaje.attach(imagen)
    except:
        print('Error de creacion de contenido')
        return False

    # Envio del mensaje
    try:
        mailServer.sendmail(remitente,
                            destinatario,
                            mensaje.as_string())
    except:
        print('Error al enviar el mensaje')
        return False
```

```
61
        # Cierre de la conexion
63      mailServer.close()
        return True
```

Bibliografía

[1] J. M. Suelves, V. Martínez, and A. Medina, "Lesiones por caídas y factores asociados en personas mayores de cataluña, españa," 2010.

[2] OMS, "Caídas," *Centro de prensa de la OMS*, p. 344, Agosto 2017.

[3] G. Pérolle and I. E. Arritxabal, "Detector automático de caídas y monitorización de actividad para personas mayores," *Revista Española de Geriatría y Gerontología*, vol. 41, pp. 33–41, 2006.

[4] J. Willems, "Camera system for elderly fall detection," *De Nayer Instituut*, 2009.

[5] R. Cucchiara, C. Grana, A. Prati, and R. Vezzani, "Computer vision system for in-house video surveillance," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 242–249, 2005.

[6] A. Nogué and J. Antiga, "Aplicación práctica de la visión artificial en el control de procesos industriales," *Gobierno de España (Ministerio de Educación)*, 2012.

[7] Belongie, S. [Nat and Friends], *How Computer Vision Is Finally Taking Off, After 50 Years*. `https://www.youtube.com/watch?v=eQLcDmfmGB0`, [Último acceso: Agosto 2017].

[8] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[9] G. Martinsanz and J. de la Cruz García, *Visión por computador: imágenes digitales y aplicaciones*. RA-MA, 2001.

[10] N. L. Fernández García, *Tema 1.- Introducción a la Visión Artificial*. Visión Artificial Avanzada, Máster de Ingeniería Informática, Universidad de Córdoba: Escuela Politécnica Superior, 2016/2017.

[11] J. Yáñez García, "Tracking de personas a partir de visión artificial," Master's thesis, 2010.

[12] Z. Hussain, *Digital Image Processing: Practical Applications of Parallel Processing Techniques*. Ellis Horwood series in digital and signal processing, E. Horwood, 1991.

[13] C. Kamath and S. Cheung, "Robust techniques for background subtraction in urban traffic video," tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2003.

[14] S. Benton, *Background subtraction, part 1: MATLAB models EE Times.* `http://www.eetimes.com/document.asp?doc_id=1275604`, [Último acceso: Agosto 2017].

[15] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," *Computer Vision—ECCV 2000*, pp. 751–767, 2000.

[16] C. Martin, "Background subtraction using running gaussian average: A color channel comparison," in *Seminar aus Bildverarbeitung und Mustererkennung*, 2014.

[17] N. J. McFarlane and C. P. Schofield, "Segmentation and tracking of piglets in images," *Machine vision and applications*, vol. 8, no. 3, pp. 187–193, 1995.

[18] M. Piccardi, "Background subtraction techniques: a review," in *Systems, man and cybernetics, 2004 IEEE international conference on*, vol. 4, pp. 3099–3104, IEEE, 2004.

[19] A. Williams, D. Ganesan, and A. Hanson, "Aging in place: fall detection and localization in a distributed smart camera network," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 892–901, ACM, 2007.

[20] V. Vishwakarma, C. Mandal, and S. Sural, "Automatic detection of human fall in video," *Pattern Recognition and Machine Intelligence*, pp. 616–623, 2007.

[21] Página web oficial de Raspbian. `http://www.raspbian.org/`, [Último acceso: Agosto 2017].

[22] Página web oficial de OpenCV. `http://www.opencv.org/`, [Último acceso: Agosto 2017].

[23] Página web oficial de Raspberry Pi. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`, [Último acceso: Agosto 2017].

[24] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer methods and programs in biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[25] I. Charfi, J. Miteran, J. Dubois, M. Atri, and R. Tourki, "Optimised spatio-temporal descriptors for real-time fall detection: comparison of svm and adaboost based classification," *Journal of Electronic Imaging*, vol. 22, no. 4, pp. pp–17, Octubre, 2013.

[26] L. del Valle Hernández, *Detección de movimiento con OpenCV y Python.* `https://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/`, [Último acceso: Agosto 2017].

[27] L. M. Fuentes and S. A. Velastin, "From tracking to advanced surveillance," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, vol. 3, pp. III–121, IEEE, 2003.

[28] R. Guo, Q. Dai, and D. Hoiem, "Single-image shadow detection and removal using paired regions," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 2033–2040, IEEE, 2011.

[29] A. Prati, I. Mikic, C. Grana, and M. M. Trivedi, "Shadow detection algorithms for traffic flow analysis: a comparative study," in *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pp. 340–345, IEEE, 2001.

[30] J. C. S. Jacques, C. R. Jung, and S. R. Musse, "Background subtraction and shadow detection in grayscale video sequences," in *Computer Graphics and Image Processing, 2005. SIBGRAPI 2005. 18th Brazilian Symposium on*, pp. 189–196, IEEE, 2005.

[31] A. Senior *et al.*, "Tracking people with probabilistic appearance models," in *ECCV workshop on Performance Evaluation of Tracking and Surveillance Systems*, pp. 48–55, 2002.

[32] A. E. Pece, "From cluster tracking to people counting," in *IEEE Workshop on Performance Evaluation of Tracking and Surveillance*, pp. 9–17, 2002.

[33] C.-W. Lin and Z.-H. Ling, "Automatic fall incident detection in compressed video for intelligent homecare," in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pp. 1172–1177, IEEE, 2007.

[34] "Ley orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal.," in *Boletín Oficial del Estado*, no. 298, pp. 43088–43099, 14 de Diciembre de 1999.

[35] "Ley 33/2011, de 4 de octubre, general de salud pública.," in *Boletín Oficial del Estado*, no. 240, pp. 104593–104626, 5 de Octubre de 2011.

[36] "Real decreto 186/2016, de 6 de mayo, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos.," in *Boletín Oficial del Estado*, no. 113, pp. 31015–31038, 10 de Mayo de 2016.

[37] "Real decreto 187/2016, de 6 de mayo, por el que se regulan las exigencias de seguridad del material eléctrico destinado a ser utilizado en determinados límites de tensión.," in *Boletín Oficial del Estado*, no. 113, pp. 31039–31054, 10 de Mayo de 2016.

[38] "Real decreto 188/2016, de 6 de mayo, por el que se aprueba el reglamento por el que se establecen los requisitos para la comercialización, puesta en servicio y uso de equipos radioeléctricos, y se regula el procedimiento para la evaluación de la conformidad, la vigilancia del mercado y el régimen sancionador de los equipos de telecomunicación.," in *Boletín Oficial del Estado*, no. 113, pp. 31055–31094, 10 de Mayo de 2016.

[39] Página web Cuéntica, *Tabla de años y porcentajes de amortización para sociedades a partir de 2015.* https://ayuda.cuentica.com/tabla-anos-y-porcentajes-de-amortizacion-sociedades-a-partir-de-2015/, [Último acceso: Septiembre 2017].

[40] Página web Domoticalia, *Marcador telefonico con boton de panico (teleasistencia).* https://www.domoticalia.es/es/teleasistencia-para-mayores/183-marcador-telefonico-con-boton-de-panico-teleasistencia-5412810212422.html, [Último acceso: Septiembre 2017].

[41] Página web Tiendadealarmas, *Pulsador de emergencia para personas mayores o personas dependientes sin Cuotas.* https://www.tiendadealarmas.com/alarmas-compactas/pulsador-de-emergencia-para-personas-mayores-o-personas-dependientes, [Último acceso: Septiembre 2017].

[42] Página web Domoticalia, *Botón de pánico GSM con localizador GPS y envio de SMS.* https://www.domoticalia.es/es/teleasistencia-para-mayores/669-boton-de-panico-para-ancianos-emergencia-gsm-8712412676156.html, [Último acceso: Septiembre 2017].

[43] Página web Sensovida. https://www.sensovida.com/, [Último acceso: Septiembre 2017].

[44] Página web CleverLoop, *Comparativa de los mejores kits de videovigilancia para el Hogar.* http://blog.cleverloop.es/2017/04/07/comparativa-de-los-mejores-kits-de-videovigilancia-para-el-hogar/, [Último acceso: Septiembre 2017].