Máster Universitario Oficial en Sistemas Telemáticos e Informáticos

Curso académico 2009-2010

Trabajo Fin de Máster

Sistema autónomo de detección de caídas con recepción de alarmas en un teléfono móvil

Tutor: José María Cañas Plaza

Autor: Sara Marugán Alonso

A mi familia y amigos.

Agradecimientos.

En primer lugar debo dar las gracias a mi tutor José María por darme la oportunidad de trabajar en proyectos de investigación innovadores y estimulantes. Por permitirme aprender de él enriqueciendo mucho mi formación en la universidad. Gracias.

A David y a Roberto quiero agradecerles todas las dudas resueltas a lo largo del proyecto, que no han sido pocas.

Gracias también a los "robóticos" en general por el buen ambiente de trabajo y su ayuda en los experimentos del proyecto. Sus "caídas" han aportado mucho a este trabajo fin de máster :p

También quiero agradecer a mi familia todo el apoyo y consejos que he recibido durante estos años, sin ella no hubiera podido hacerlo.

A Juan Antonio quiero agradecerle el que haya estado siempre ahí, dispuesto a escucharme y darme palabras de ánimo.

Por último, pero no por ello menos importante, a mis amigos, por su interés y apoyo mostrado.

Resumen.

La tecnología aplicada al hogar, conocida como domótica, integra automatización, informática y nuevas tecnologías de comunicación. El vertiginoso avance tecnológico experimentado en los últimos años ha contribuido eficazmente al desarrollo de la domótica en aspectos tan cotidianos como la iluminación, climatización, comunicación, accesibilidad, seguridad, etc.

Este trabajo fin de máster se enmarca dentro del desarrollo del proyecto *Eldercare*, cuya finalidad es construir una aplicación robusta de asistencia y cuidado de mayores en su lugar de residencia utilizando visión artificial. Las cámaras son sensores que proporcionan gran cantidad de información sobre el entorno donde se utilizan. Extraer e interpretar esa información permite la creación de aplicaciones que realicen tareas útiles para nuestra vida cotidiana.

La finalidad de este trabajo es portar la aplicación *Eldercare* a la nueva plataforma software *Jderobot-5.0*, basada en el middleware orientado a objetos ICE (*Internet Communications Engine*). El sistema ahora se materializa mediante componentes que interactúan entre sí a través de sus interfaces. Esto aporta la flexibilidad necesaria para que el sistema pueda ejecutarse de manera distribuida.

Otro aporte significativo ha sido la creación de un cliente para móvil *Android* que recibe la información relevante que produce la aplicación *Eldercare*. Parte de esta información consiste en alarmas registradas en una base de datos y a las que se les asocia un vídeo de lo ocurrido tras detectar el peligro. Desde el cliente móvil pueden verse estos vídeos, una cámara en directo a través de *streaming* y la cámara virtual de la aplicación, en la que se representan las trayectorias en 3D de las personas que está siguiendo el sistema.

Una vez integrados todos los componentes del sistema, se hicieron pruebas para validar experimentalmente su correcto funcionamiento y caracterizarlo. Para valorar la robustez del sistema, se realizaron diferentes montajes experimentales en lugares distintos, incluyendo el salón de una casa.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este primer capítulo se describe el contexto en el que se encuadra este proyecto y la motivación para realizarlo. Este trabajo persigue la creación de un sistema de tele-asistencia basado en visión y que integra al teléfono móvil como interfaz natural. Esto implica la confluencia de tres campos diferentes: tele-asistencia, visión artificial y teléfonos inteligentes o *smartphones*.

A continuación se verá una breve introducción de cada uno de estos campos y por último se expondrán los antecedentes y el contexto particular en el que nació este trabajo fin de máster.

1.1. Visión artificial

La visión artificial o visión computacional es el área de la inteligencia artificial que se dedica a extraer información de las imágenes con el fin de comprender y asimilar lo que en ellas está sucediendo. La visión artificial tiene como objetivo interpretar las imágenes analizadas para obtener información relevante sobre elementos que en ellas aparecen.

La vista es el sentido que más utilizamos para captar información de nuestro entorno y consiste en la habilidad de detectar la luz y de interpretarla. Tanto los humanos como los animales poseemos un sistema visual que nos permite crear un esquema de nuestro entorno y conocer detalles de los objetos que nos rodean. Gracias a estos detalles somos capaces de reconocer dichos objetos por su color, por su forma, detectar movimiento en ellos o incluso estimar aproximadamente la distancia que nos separa.

En los últimos años un área activa de investigación es la reproducción de estas habilidades mediante sistemas informáticos. Las cámaras son sensores de bajo coste

cada vez más comunes en nuestra vida cotidiana. Hoy en día es normal encontrar cámaras en la ciudad, en el metro, en las carreteras y túneles, etc. Además también las tenemos en nuestros móviles y ordenadores. Gracias a esta proliferación y al aumento en la potencia de procesamiento en los ordenadores actuales, la visión se ha convertido en un área en expansión que está dando lugar a un abanico enorme de aplicaciones.

Figura 1.1: Proliferación de las cámaras.

La ciudad de Londres está plagada de cámaras para controlar el acceso a su zona centro, ya que existe una prohibición de circular por dicha zona debido a problemas de congestión y de contaminación que sufría la ciudad en el pasado. El elevado número de vehículos que siguen circulando por la ciudad hace imposible controlar el acceso manualmente. La solución más fácil y extensible es mediante la utilización de cámaras para obtener la matrícula de los vehículos y contrastarla con una base de datos que almacena la contabilidad de los conductores que han abonado las tasas.

Otra aplicación basada en visión consiste en la creación de interfaces para que las personas interactúen con su ordenador o videoconsola. Un buen ejemplo es el periférico *Eyetoy* (figura 1.2), creado por London Studio para la PlayStation 2, que tiene una cámara que permite que el jugador interactúe con lo que aparece en la pantalla. Otro ejemplo en esta línea es *Kinect* de *Microsoft* para la videoconsola *Xbox 360*, un nuevo sistema de control de juegos en el que usamos nuestro cuerpo y voz para jugar.

Figura 1.2: (a) EyeToy de PlayStation 2 y (b) Proyecto Kinect de Microsoft para Xbox 360

Los avances en geometría proyectiva, pares estéreo y autocalibración han abierto la puerta a la extracción de *información tridimensional* de las imágenes. El uso de estas nuevas técnicas permite alcanzar nuevas cotas en el área de la visión artificial, pues la información tridimensional permite conocer mejor y con una mayor precisión el entorno. Por ejemplo, hoy día es posible realizar *reconocimiento 3D* de rostros u objetos, que puede ser utilizado en el reconocimiento de personas mediante el estudio biométrico en áreas de seguridad, diseño infográfico para la industria del cine o de videojuegos, etc.

Un ejemplo de sistema basado en visión que recrea una escena tridimensional es *EyeVision*[1], que permite modelar en tiempo real una escena de manera fotorrealista con el fin de obtener nuevas imágenes virtualizadas de la misma. Este sistema ha sido empleado en la *Superbowl* para la repetición de jugadas desde cualquier ángulo. A través de la colocación estratégica de un determinado número de cámaras alrededor de un estadio es posible obtener la realidad 3D virtualizada de lo sucedido segundos antes. La escena tridimensional se obtiene de la composición de las imágenes 2D que consigue cada una de las cámaras.

[1] http://www.ri.cmu.edu/events/sb35/tksuperbowl.html

Figura 1.3: Sistema de repetición tridimensional Eye-Vision.

Una de las tareas comunes en visión artificial es el seguimiento 3D de objetos. Existen dos tipos de soluciones a este problema: sistemas de una única cámara en movimiento y sistemas que emplean varias cámaras fijas calibradas en común. El ejemplo más conocido de sistema que utiliza múltiples cámaras es el sistema *Vicon*[2] (figura 1.4), empleado en distintos ámbitos tales como medicina (laboratorio de análisis de marcha), deporte o en la industria cinematográfica. Mediante el uso de los marcadores especiales situados sobre el cuerpo de una persona y varias cámaras filmando desde diferentes puntos de vista, un ordenador analiza todas las trayectorias y extrae un esqueleto 3D animado.

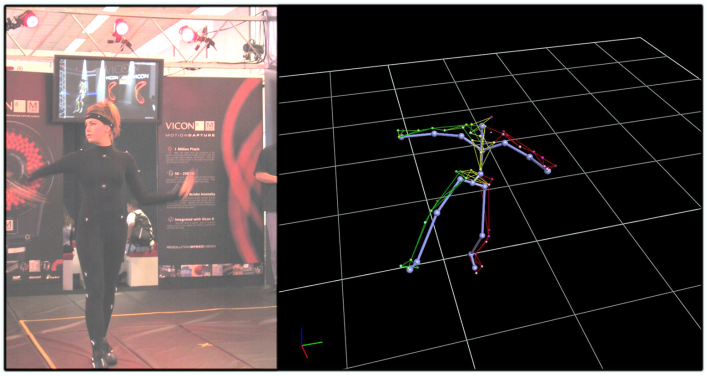

Figura 1.4: Sistema VICON.

1.2. Tele-asistencia

El cuidado de personas mayores o enfermas implica una monitorización continua de su quehacer diario. En muchos casos son los propios familiares o los servicios sociales los encargados de cuidarles en los domicilios de estas personas o en residencias especializadas. Pero aun contando con el personal necesario para el cuidado es imposible

[2]http://www.vicon.com/

observar a estos pacientes continuamente con el fin de detectar lo antes posible cualquier tipo de incidencias. El problema se agrava en el caso de personas que viven solas, las cuales necesitan aún más este tipo de asistencia.

La tele-asistencia es una tendencia creciente en los países desarrollados, donde las poblaciones están sufriendo un envejecimiento progresivo. Por ejemplo, según los datos que ha hecho públicos el Instituto Español de Estadística (INE) en la Proyección de Población a Corto Plazo de 2009-2049, en 2049 España tendrá 48 millones de habitantes, de los cuales el 31.9 % de ellos serán mayores de 64 años.

Cada vez son más los usuarios de este tipo de asistencia. El estudio más reciente (2008) de la consultora especializada DBK sobre "Servicios Asistenciales a Domicilio" confirma que por tipo de servicio, el de tele-asistencia ha venido creciendo a un ritmo más rápido que el de ayuda a domicilio, llegando a superarlo en número de usuarios desde 2007. También indica que en 2010 se espera que los usuarios de tele-asistencia alcancen la cifra de unos 647.000 usuarios. En cuanto a facturación, la tele-asistencia se incrementó en un 17.9 % hasta situarse en los 112 millones de euros.

En cuanto a los tipos de tele-asistencia, existen en la actualidad diferentes sistemas para ofrecer este servicio. Los sistemas de tele-asistencia tradicionales emplean terminales o dispositivos especiales que portan los propios pacientes, en forma de pulsera o collar, y que en un momento de necesidad son accionados por éstos para enviar la llamada de emergencia a los servicios de atención[3]. Esta necesidad de acción por parte de la persona representa una limitación, ya que en los casos en que se produce una pérdida de consciencia no será posible dicha acción, y por tanto el sistema queda inutilizado.

Una alternativa más evolucionada de tele-asistencia es la utilización de dispositivos de monitorización externos. Por ejemplo un conjunto de cámaras de vídeo se encarga de captar imágenes de las estancias y transmitirlas a un centro receptor de imágenes. Cuando el familiar u operador del servicio de asistencia observa en los monitores imágenes que representen un comportamiento que entrañe riesgo, inicia las actuaciones adecuadas. Este tipo de sistemas, que denominaremos "sistemas de tele-vigilancia" (ver imagen 1.6) representan frente a los anteriores algunas ventajas (como que actúan en

[3]http://www.teleasistencia.com/teleasist.sfw,
http://www.mapfre.com/seguros/es/particulares/soluciones/teleasistencia.shtml

Figura 1.5: Sistema tradicional de tele-asistencia - (a) Pulsador, (b) Servicio de asistencia.

todo momento, aun con pérdida de consciencia de la persona) y algunos inconvenientes. El primero es la exigencia de la atención continuada a las imágenes por parte de un familiar u operador. Se trata de una tarea ardua y poco eficiente cuando las situaciones de riesgo son escasas. El segundo inconveniente es la falta de privacidad para la persona monitorizada, ya que en todo momento alguien la observa.

Figura 1.6: Sistema de tele-vigilancia.

En general, resulta difícil y costoso el paso de los sistemas de tele-vigilancia a un nuevo, y más evolucionado, tipo de sistemas que analizan las imágenes de vídeo y detectan automáticamente cuándo la situación requiere atención a partir de dicho análisis. A éstos les denominaremos "sistemas de tele-vigilancia autónomos". El motivo principal de esta dificultad es el alto coste computacional que requiere la extracción de información útil de las cámaras, lo que encarece sobremanera tanto el proceso de desarrollo de la aplicación como la fabricación de los dispositivos. Sin embargo, este

tipo de sistemas resuelve inconvenientes de los sistemas tradicionales. Por ejemplo, en el caso de pérdida de consciencia, los sistemas autónomos no quedan inutilizados ya que no requieren ninguna acción por parte de la persona que necesita auxilio. Respecto a los sistemas de tele-vigilancia, no necesitan una persona que atienda de manera continuada las imágenes y esto además soluciona el problema de la falta de privacidad.

1.3. Teléfonos móviles inteligentes

El uso de teléfonos móviles es otra tendencia creciente en nuestra sociedad. Hace años que existen más teléfonos móviles que habitantes y que el número de hogares con móvil supera al de los que tienen teléfono fijo.

El *smartphone* o teléfono inteligente es un dispositivo electrónico que funciona como un teléfono móvil con características similares a las de un ordenador. Una característica importante de estos dispositivos es que permiten la instalación de programas para incrementar la funcionalidad y procesamiento de datos. Es común que dispongan de interfaces hasta hace poco impensables en los móviles, como puede ser un teclado *QWERTY*, pantalla táctil, o acceso a internet a través de tecnología 3G/4G o wifi.

No es casualidad que las compañías más poderosas del mundo hayan realizado un cambio de rumbo en sus estrategias ya que el *smartphone* es un dispositivo a explotar en los próximos años. *Google* y *Apple* son dos de las empresas que han visto el negocio y la necesidad tecnológica que tiene la sociedad de disponer de *smartphones* cada vez más rápidos, más modernos y con más utilidades. Apple se introdujo en el mercado a mediados del 2007, cuando presentó su *iPhone* (fig. 1.7(a)). Fue nombrado "invento del año" por la revista *Times* en el 2008 y hasta el día de hoy ha vendido cerca de 50 millones de unidades en todo el mundo.

Por su parte Google apostó por desarrollar un sistema operativo para móviles basado en software libre, *Android*. Este desarrollo está realizado en el marco de la *Open Handset Alliance (OHA)*, una alianza de 65 empresas del sector cuyo objetivo es desarrollar estándares abiertos y dispositivos móviles. El primer móvil Android que comercializó Google junto con HTC fue el *HTC Dream* en Octubre de 2008. En abril de 2009, 6 meses después, había vendido más de 1 millón de dispositivos sólo en Estados Unidos. Actualmente existen más de 100 modelos distintos de *smartphones*,

comercializados por más de 30 empresas distintas, que llevan Android instalado. Uno de los más potentes actualmente es el *Nexus One* (fig. 1.7(b)), comercializado por Google.

(a) (b)

Figura 1.7: iPhone de Apple (a) y Nexus One de Google (b)

La acogida de estos nuevos dispositivos es tan alta por parte de la sociedad que la industria ha visto la necesidad de portar a los *smartphone* aplicaciones que históricamente se han usado en el ordenador. Correo electrónico, acceso a las redes sociales y servicios web, reproductores multimedia (audio y vídeo) son entre otras, aplicaciones demandadas en estos dispositivos móviles.

1.4. Eldercare

Eldercare es un proyecto enfocado a la construcción de una aplicación robusta de tele-asistencia y cuidado de mayores en su lugar de residencia. La aplicación se apoya en técnicas de visión artificial para mantener localizados a una o más personas dentro de una estancia cubierta por dos o más cámaras. Esta localización se realiza en tres dimensiones, lo que permite detectar si una persona se ha caído al suelo y disparar algún tipo de alarma (sonora, visual, mensaje de texto, etc).

Sobre la base existente del sistema *Eldercare* arranca el punto de partida de este trabajo fin de máster.

El primer antecedente de este proyecto es el proyecto fin de carrera *Aplicación de seguridad basada en visión* [Pineda, 2006], que consistió en realizar una aplicación de seguridad que localiza tridimensionalmente dentro de una habitación a un *único sujeto*

del cuál se conoce su color y establece un umbral en la coordenada Z para disparar la alarma. Si el sujeto cae al suelo, el sistema avisa con una alarma sonora y visual (figura 1.8). Para el desarrollo de esa aplicación se emplearon dos técnicas distintas: un algoritmo evolutivo monomodal y una técnica probabilística denominada filtro de partículas, en ambos casos, utilizando una primitiva puntual (punto 3D).

Figura 1.8: Primera versión de Eldercare.

El segundo proyecto en esta línea fue el proyecto fin de carrera que realicé para el título de Ingeniero Técnico en Informática de Sistemas, *Seguimiento 3D visual de múltiples personas utilizando un algoritmo evolutivo multimodal*[Marugán, 2007], que conseguía el seguimiento de *múltiples* sujetos y aprender automáticamente el color de cada uno de ellos (figura 1.9). Este nuevo sistema también se aplicó a la detección de caídas [Cañas *et al.*, 2008] [Cañas *et al.*, 2009].

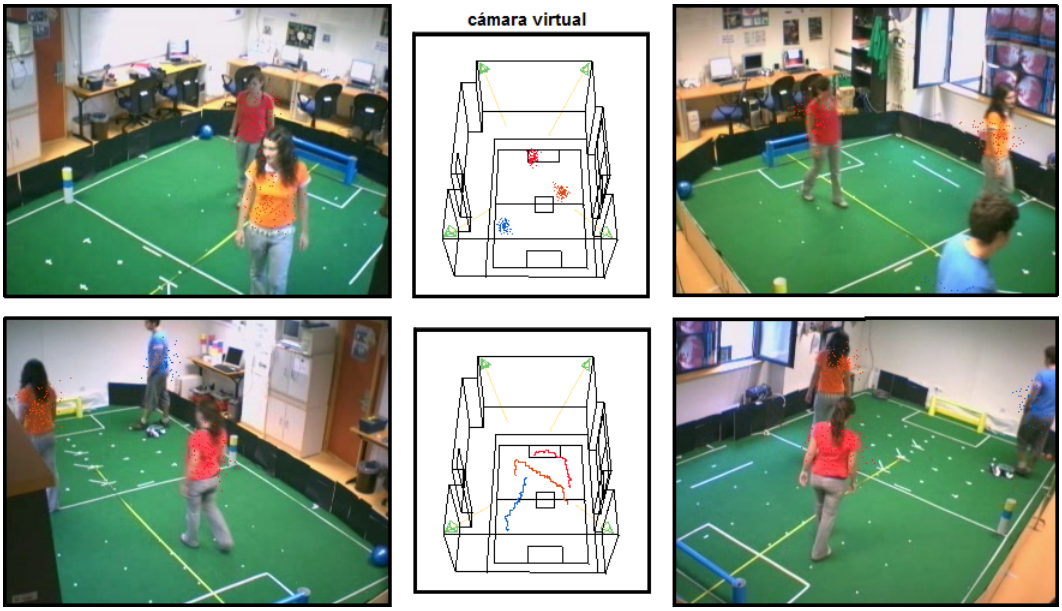

Figura 1.9: Segunda versión de Eldercare.

En la tercera generación del proyecto (*Seguimiento visual de personas mediante evolución de primitivas volumétricas*[Marugán, 2010]) se cumplieron nuevos objetivos como: utilizar una *primitiva con volumen* (ver figura 1.10), hacer más robusto el aprendizaje de color y explorar la alternativa de utilizar información bidimensional extraída del análisis individual de cada cámara para fusionarla y obtener así la localización tridimensional. Las primitivas volumétricas dotaron al sistema de mayor precisión y la mejora del aprendizaje de color hizo que el sistema mejorara notablemente su robustez, acercándolo a una aplicación real que pueda introducirse en el mercado como sistema automático de detección de caídas (figura 1.11).

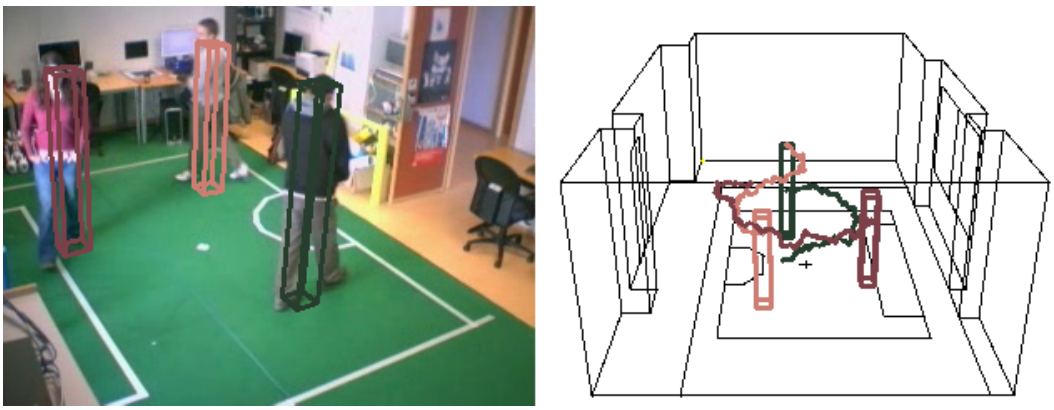

Figura 1.10: Tercera versión de Eldercare - seguimiento de tres personas.

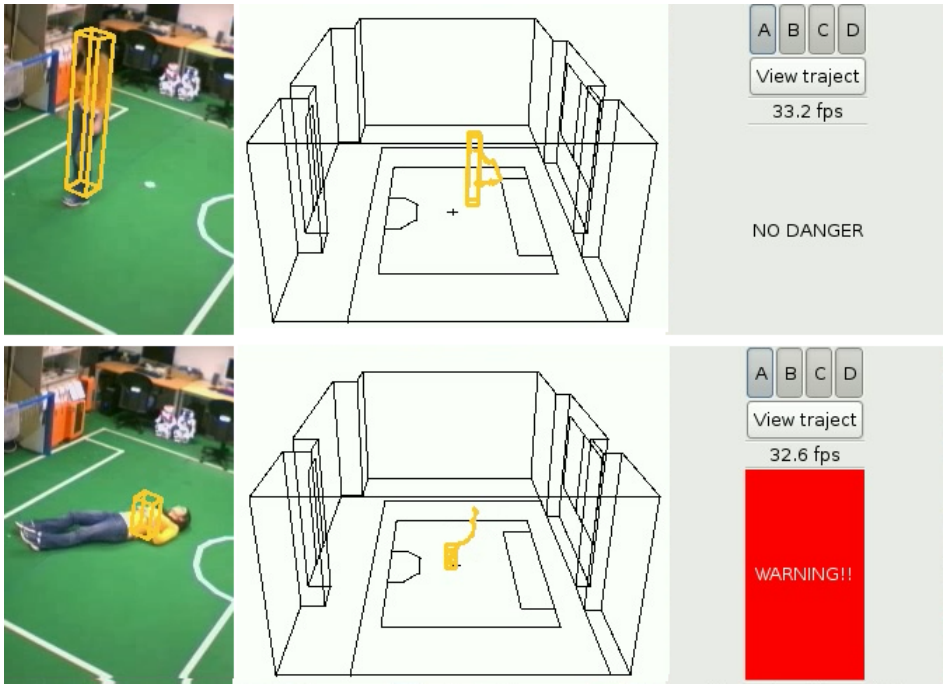

Figura 1.11: Tercera versión de Eldercare - detección de una caída.

Este trabajo supone la cuarta generación del proyecto *Eldercare*, en la que confluyen las tres áreas mencionadas: visión, tele-asistencia y telefonía móvil como interfaz natural. El objetivo principal del trabajo fin de máster es migrar la aplicación a la nueva plataforma *Jderobot-5.0*, que se basa en una arquitectura de componentes distribuidos y facilita la integración con un cliente para móvil Android como interfaz de usuario y para recibir las alarmas generadas. Esto aumentará en gran medida la utilidad de la aplicación para el usuario final. Además, la arquitectura de componentes distribuidos facilitará la monitorización de múltiples habitaciones en un futuro.

La plataforma está en una fase inicial de desarrollo, por lo que parte del trabajo consistirá en desarrollar la funcionalidad requerida en la plataforma para el funcionamiento de la aplicación. Para portar *Eldercare* a la nueva plataforma, la aplicación será refactorizada para utilizar el paradigma de programación orientado a objetos y utilizará el Modelo Vista Controlador (MVC).

En el siguiente capítulo se exponen con detalle los objetivos y la planificación de este proyecto. Después, se presenta el entorno de desarrollo y el hardware utilizados para su realización. En el capítulo 4 se describe el diseño e implementación de la aplicación. A continuación, en el capítulo 5, se detallan los experimentos realizados y por último, en el capítulo 6, se exponen las conclusiones y las vías futuras de continuación.

Capítulo 2
Objetivos

Una vez expuesto el contexto general y particular sobre el que se asienta este trabajo, estableceremos en este capítulo los objetivos concretos que se han planteado al igual que los requisitos que han condicionado el desarrollo del mismo.

2.1. Descripción del problema

Como se ha presentado en la introducción, *Eldercare* es un proyecto enfocado a la construcción de una aplicación robusta de asistencia y cuidado de mayores en su lugar de residencia. Este trabajo pretende evolucionar el proyecto *Eldercare* a una cuarta generación.

El objetivo principal de este trabajo fin de máster es portar la aplicación *Eldercare* a la nueva plataforma *Jderobot-5.0*. La finalidad es aprovechar su arquitectura distribuida basada en componentes e integrar en el sistema un teléfono móvil que reciba las alarmas y sirva como interfaz de usuario de la aplicación.

El objetivo final se ha articulado en varios subobjetivos más específicos:

1. *Reimplementación de la aplicación en la plataforma Jderobot-5.0*. En primer lugar se pretende refactorizar el proyecto fin de carrera *Seguimiento visual de personas mediante evolución de primitivas volumétricas*[Marugán, 2010] para utilizar el paradigma de programación orientado a objetos. La aplicación *Eldercare* pasa a ser un componente en la plataforma *Jderobot-5.0*.

2. *Construcción de un cliente (EldercareClient) para móvil Android desde el que se pueda consultar la información generada por Eldercare*. Este subobjetivo abarca el mecanismo de generación de alarmas del sistema. En el sistema anterior la alarma era simplemente visual y auditiva. Ahora el sistema será capaz de *enviar un email*

de aviso, *grabar vídeo* tras producirse una alarma y registrar esta información en una *base de datos*, que podrá ser consultada a través del cliente móvil. Además el cliente también permitirá visualizar las imágenes de las cámaras en tiempo real mediante *streaming* y ver las trayectorias en 3D en una cámara virtual con OpenGL.

3. *Experimentos*. El sistema se probará con todos los componentes integrados y se realizarán pruebas para caracterizar su robustez. La mayoría de las pruebas se realizarán en una *habitación de gran volumen y con condiciones de iluminación variables*, concretamente, en el Laboratorio de Robótica de esta universidad. Otras pruebas se llevarán a cabo en un entorno más real, como por ejemplo en el salón de una casa.

No es un objetivo primordial pero debido a que la plataforma se encuentra en un estado inicial de desarrollo, se pretende aportar nueva funcionalidad y mejorar los aspectos necesarios de los componentes de *Jderobot-5.0* que utiliza *Eldercare*.

2.2. Requisitos

El desarrollo del proyecto estará guiado por los objetivos comentados anteriormente y deberá ajustarse a los requisitos resumidos en los siguientes puntos:

1. Implementación del sistema apoyándose en la *plataforma software Jderobot-5.0*[1] sobre el sistema operativo *GNU/Linux*, concretamente la distribución Ubuntu.

2. Funcionamiento de la aplicación usando *hardware convencional*.

3. El software a desarrollar será *vivaz*, será capaz de seguir personas andando dentro de la habitación.

4. Seguimiento de varias personas con una *precisión centimétrica*.

5. El sistema será capaz de *avisar del peligro al usuario en tiempo real* vía email y grabará vídeo tras la detección del peligro. Estos vídeos podrán ser consultados a través de un *cliente móvil* con sistema operativo Android.

2.3. Metodología

En esta sección se presenta la metodología utilizada para la realización de este proyecto. El proyecto ha sido desarrollado siguiendo el modelo de desarrollo en

[1] http://jde.gsyc.es/

espiral basado en prototipos. La elección de este modelo se basa en la necesidad de separar el comportamiento final del sistema en varias subtareas que posteriormente serán integradas.

Cada subtarea se divide en un número determinado de ciclos. En cada uno de estos ciclos existen cuatro etapas: *Análisis de requisitos, Diseño e implementación, Pruebas y Planificación del próximo ciclo de desarrollo.*

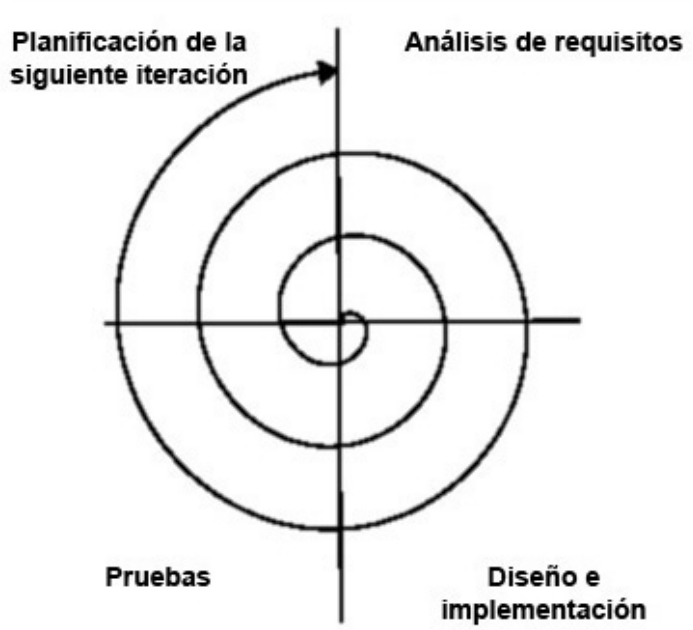

Figura 2.1: Modelo en espiral

1. *Análisis de requisitos.* Los objetivos de un ciclo de desarrollo deben ser identificados y especificados.

2. *Diseño e implementación.* Construcción del incremento de funcionalidad perteneciente al ciclo dado.

3. *Pruebas.* El sistema es validado con pruebas que verifican el correcto funcionamiento de acuerdo a los requisitos.

4. *Planificación del próximo ciclo de desarrollo.* Revisamos todo lo hecho, evaluándolo, y con ello decidimos si continuamos con las fases siguientes y planificamos el próximo ciclo.

Durante todo el desarrollo del proyecto se han mantenido reuniones semanales con el tutor para establecer y afinar los puntos que se han llevado a cabo en cada etapa,

así como para comentar los resultados de etapas anteriores. A modo de bitácora, se ha ido plasmando la evolución del proyecto en un *mediawiki*[2] donde se pueden encontrar fotografías y vídeos de los experimentos realizados a lo largo del trabajo.

2.3.1. Plan de trabajo

El plan de trabajo seguido se ha dividido en diferentes fases, cuyos productos han consistido en prototipos o conocimiento adquirido.
Un prototipo es una versión preliminar de un sistema con fines de demostración o evaluación de ciertos requisitos. Se suele asociar la finalización de una iteración en el modelo de desarrollo con la obtención de un nuevo prototipo. Sin embargo, la obtención de un prototipo no implica que se utilice para el producto final, se puede desechar dicho prototipo.

1. *Estudio del middleware ICE*[3]. En esta primera fase se estudió la arquitectura y el funcionamiento del middleware ICE, en el que se apoya la plataforma *Jderobot-5.0*.

2. *Estudio de Jderobot-5.0*. En esta fase se estudió el código fuente y el funcionamiento de los componentes de esta nueva plataforma.

3. *Mejora del componente CameraServer*. En esta fase se introdujeron dos mejoras en el componente que sirve las imágenes a la aplicación. La primera consistió en integrar el mecanismo de publicación/subscripción de ICE para transmitir las imágenes, denominado *IceStorm*. La segunda fue añadir el soporte de cámaras firewire, utilizadas en el laboratorio.

4. *Diseño e implementación del componente Calibrator*. En esta fase se creó un nuevo componente para la plataforma software *Jderobot-5.0*, necesario para la calibración de las cámaras de la aplicación *Eldercare*.

5. *Mejora del componente Recorder*. Para la grabación de vídeo tras activarse la alarma se utiliza el componente *RecordingManager* incluido en *Jderobot-5.0*. Éste maneja la base de datos que almacena la información de las alarmas producidas y se encarga de activar la grabación de vídeo a través del componente *Recorder*.

[2] http://jde.gsyc.es/index.php/Salons-ii
[3] http://www.zeroc.com/ice.html

En esta fase se construyó un nuevo tipo de grabador basado en la herramienta *mencoder* y se incluyó dentro del componente *Recorder*. Este nuevo grabador recoge las imágenes de un *CameraServer* y genera un fichero de vídeo.

6. *Diseño e implementación del componente Eldercare*. En esta fase se reimplementó la aplicación *Eldercare* como un nuevo componente en la plataforma *Jderobot-5.0*. Éste procesa las imágenes que recibe de *CameraServer* a través del algoritmo propuesto en [Marugán, 2010] para obtener la localización y seguimiento 3D visual de las personas en la estancia. El algoritmo se revisitó y se mejoró en algún aspecto.

7. *Diseño e implementación del componente EldercareClient*. En esta fase se estudió el sistema operativo Android y el entorno de desarrollo java *Eclipse IDE* para programación de móviles Android. Después se implementó *EldercareClient* como un componente que ejecuta en un móvil Android real e interactúa con otros componentes de la plataforma.

8. *Experimentos*. La última fase consistió en validar el funcionamiento del sistema con todos los componentes integrados y en la realización de experimentos más exhaustivos para caracterizarlo.

Capítulo 3
Entorno de desarrollo

En este capítulo se presenta la plataforma de desarrollo y la infraestructura utilizada para el proyecto. La plataforma software utilizada ha sido *Jderobot-5.0*, desarrollada en el propio Grupo de Robótica de esta universidad. En este proyecto la plataforma utilizada cobra protagonismo debido a que *Jderobot-5.0* se encuentra en un estado inicial de desarrollo y parte de las tareas del trabajo fin de máster han consistido en aportar funcionalidad nueva a la plataforma.

Por otra parte, se han usado cuatro librerías auxiliares: *GTK*, *OpenCV* y *Progeo*. *GTK* se ha utilizado para la generación de la interfaz gráfica, *OpenCV* para el tratamiento de imágenes y *Progeo* para las operaciones relacionadas con la información tridimensional en la aplicación principal.
Para la creación de la aplicación cliente para móvil se ha utilizado el entorno de desarrollo *Eclipse IDE* y el kit de desarrollo de Android para el mismo. Además, para visualizar la cámara virtual en el móvil se ha usado la librería *OpenGL ES* para Android.

3.1. Plataforma software Jderobot-5.0

Jderobot[1] es una plataforma de software libre para la programación de aplicaciones que generan comportamientos autónomos en robots y/o que están relacionadas con la visión artificial y con la domótica. Para ello resuelve los aspectos más generales de la programación de robots, como son el acceso a los sensores y actuadores, la multitarea, interfaces gráficas y las comunicaciones entre componentes. Añadiendo a todo esto software inteligente, se puede conseguir comportamientos avanzados.

Jderobot inició sus orígenes en el 2003 ([Cañas Plaza, 2003]) y ha sufrido varios cambios en su diseño con sucesivos proyectos destinados a tal fin ([Bravo, 2004] y

[1] http://jde.gsyc.es/

[Bravo, 2010]). La última versión (*Jderobot-5.0*) es aún experimental y está basada en el middleware orientado a objetos *ICE* (*Internet Communications Engine*) [Henning, 2004] de la compañía *ZeroC*[2]. Este middleware provee de una capa de abstracción sobre la red y sus conexiones, gracias a él se pueden distribuir aplicaciones sin necesidad de tener un experto en redes y comunicaciones. Además, provee de serie la interoperabilidad entre diferentes lenguajes (Java, C++ y Python entre otros) y existe una versión más ligera de *ICE*, llamada *Ice-e*, que puede ser ejecutada en terminales móviles.

El diseño de *ICE* está influido por el middleware CORBA, sin embargo es mucho menos complejo. *ICE* es un conjunto de componentes que interactúan entre sí a través de interfaces (ver figura 3.1). Éstas se definen con un lenguaje de definición de interfaces llamado *SLICE* (*Specification Language for ICE*), que es propietario de *ZeroC*.

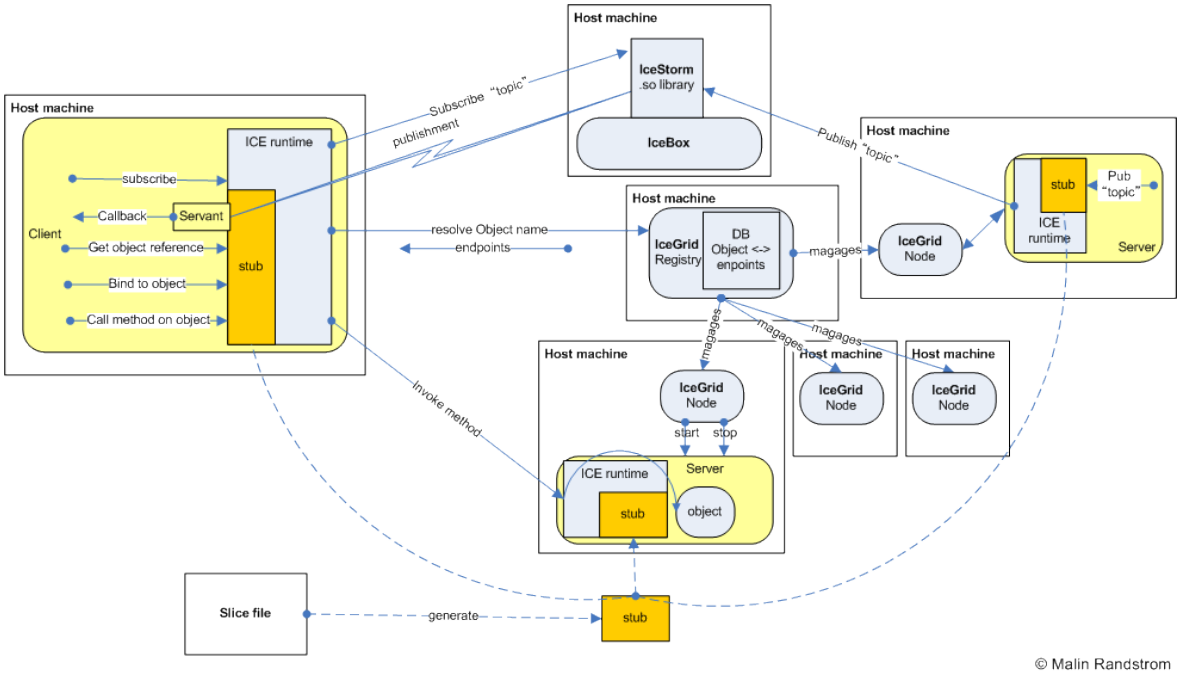

Figura 3.1: Componentes de *ICE*.

En este proyecto concretamente se han utilizado dos servicios proporcionados por *ICE*:

- *IceBox*: es un único servidor de aplicaciones que permite gestionar el arranque y parada de un determinado número de componentes de aplicación. Dichos

[2]http://www.zeroc.com/

componentes se pueden desplegar como una biblioteca dinámica en lugar de un proceso. Este hecho reduce la sobrecargar global del sistema.

- *IceStorm*: es un servicio de publicación/subscripción que actúa como un distribuidor de eventos entre servidores y clientes. Un único evento publicado se puede enviar a múltiples subscriptores.

Jderobot-5.0 es una colección de componentes que se comunican entre sí a través de *ICE*. Los componentes son flujos de ejecución independientes (modulable, iterativo, y que puede ser activado o desactivado a voluntad) con un determinado objetivo. En cada uno de estos componentes se encapsula diferente funcionalidad que podrá ser reutilizada en posteriores ocasiones. Para dicha reutilización es necesario que los componentes definan un interfaz para que otros componentes pueda interactuar con él, siendo el único mecanismo de acoplamiento entre ellos.

Las aplicaciones en *Jderobot-5.0* integran uno o más componentes, cada uno de ellos escrito en el lenguaje deseado y ejecutados en diferente hardware. En el caso de *Eldercare*, C++ y Java son los lenguajes utilizados y el hardware consiste en ordenadores personales y un teléfono móvil.

A continuación se presentan los diferentes componentes de *Jderobot-5.0* utilizados para el funcionamiento de *Eldercare* en esta plataforma. Para algunos ha sido necesario desarrollar nueva funcionalidad, que seguidamente se explica más en detalle.

3.1.1. Componente *CameraServer*

Este componente provee a la aplicación las imágenes que necesita. Para obtener las imágenes de los dispositivos utiliza la biblioteca *GearBox*[3], que permite acceder a cámaras usb a través del protocolo *video for linux* (v4l), a imágenes de fichero de vídeo y a flujos de vídeo servidos por la red.

Este componente utiliza las interfaces slice *ImageProvider* y *Camera*, que extiende a la anterior. A través de la interfaz *Camera* el componente recibe peticiones de imágenes por parte de las aplicaciones. A continuación vemos dichas interfaces:

[3]http://gearbox.sourceforge.net/index.html

```
interface ImageProvider
{
    /**
     * Returns the image source description.
     */
    idempotent ImageDescription getImageDescription();

    /**
     * Returns the latest data.
     */
    ["amd"] idempotent ImageData getImageData()
      throws DataNotExistException, HardwareFailedException;
};

interface Camera extends ImageProvider
{
    idempotent CameraDescription getCameraDescription();

    string startCameraStreaming();

    void stopCameraStreaming();

};
```

Las mejoras concretas que este trabajo fin de máster ha aportado al componente son la capacidad de soportar cámaras firewire y la inclusión del mecanismo de publicación/subscripción de *ICE* para servir las imágenes.

Para dar soporte a cámaras firewire en el componente *CameraServer* fue necesario utilizar la librería *libdc1394-22*. Ésta permite la inicialización, el acceso y la liberación de recursos de cámaras firewire. Este soporte era necesario debido a que las cámaras empleadas en las pruebas son cámaras web *Apple iSight*.
Para seguir utilizando la misma configuración de *CameraServer*, se adaptaron las opciones de configuración de las cámaras firewire. Aquí tenemos un ejemplo de configuración de *CameraServer* para una cámara firewire:

```
CameraSrv.NCameras=1
CameraSrv.Camera.0.Name=cameraA
CameraSrv.Camera.0.ShortDescription=Plugged to /dev/video1394/0.
CameraSrv.Camera.0.Uri=dv:///dev/video1394/0
CameraSrv.Camera.0.FramerateN=15
CameraSrv.Camera.0.FramerateD=1
CameraSrv.Camera.0.ImageWidth=640
```

```
CameraSrv.Camera.0.ImageHeight=480
CameraSrv.Camera.0.Format=RGB888
```

En cuanto a la segunda mejora, fue necesario estudiar el mecanismo de publicación/subscripción de *ICE* llamado *IceStorm*. *IceStorm* es un servicio que maneja los elementos publicadores y subscriptores a través de la definición de *topics*. Definir un *topic* es equivalente a definir una interfaz *slice*: las operaciones de la interfaz definen los tipos de mensajes soportados por el *topic*. Un publicador usa un proxy de la interfaz del *topic* para enviar los mensajes (en este caso imágenes) y un subscriptor implementa la interfaz del *topic* para recibir los mensajes. La interfaz definida para transmitir imágenes se ha llamado *ImageConsumer*:

```
//! Interface to the image consumer.
interface ImageConsumer
{
    //! Transmits the data to the consumer.
    void report( ImageData obj );
};
```

El método *report* es llamado por *CameraServer* para mandar una nueva imagen y *Eldercare* implementa el método para recibir la imagen y operar con ella.

Debido a que *Eldercare* utiliza diferentes cámaras ha sido necesario asignar un nombre de *topic* a cada una de ellas, que no es más que crearlos con la herramienta *icestormadmin* (proporcionada por *ICE*) con un nombre único. De este modo la aplicación puede subscribirse a cada cámara indicando el nombre único de cada una. Cada *topic* puede tener varios publicadores y subscriptores.
Para ejecutar el servicio *IceStorm* es necesario lanzar el contenedor de servicios de *ICE*, llamado *IceBox*.

Al usar el mecanismo de publicación/subscripción nos dimos cuenta de que aumentaba el rendimiento de la aplicación. Esto se detalla mejor en el capítulo de experimentos (capítulo 5).

Para hacerlo lo más genérico posible, *CameraServer* es capaz de mantener los dos modos de servicio (RPC y IceStorm) a la vez o deshabilitar uno de ellos a través de su

fichero de configuración:

```
// client/server mode
CameraSrv.DefaultMode=1

// publish/subscribe mode
CameraSrv.TopicManager=IceStorm/TopicManager:default -p 10000
```

3.1.2. Componente *RecordingManager*

RecordingManager es un componente que ofrece a las aplicaciones la funcionalidad de almacenar grabaciones de vídeo y asociarles alarmas [Palomino, 2010]. Guarda información de las grabaciones acerca de la fecha y la hora de la alarma, la duración y el nombre del vídeo en una *base de datos*. Este componente implementa la interfaz slice *RecordingManager*:

```
interface RecordingManager
{
    //! Returns the list of recordings
    idempotent RecordingSequence getRecordings(int from,
                    int elems);

    //! Returns the list of recordings by date
    // Format Date = "YYYY-MM-DD" - "2010-05-28"
    idempotent RecordingSequence getRecordingsByDate(string date,
                    int from, int elems);

    //! Return the event list of recording
    EventSequence getEventsOfRecording (int recordingId);

    //! Set an alarm event
    int setEvent (RecordingEvent recEvent, int recordingId);

    //! Get event
    RecordingEvent getEvent (int eventId);

    //! Start recording
    int startRecording (RecorderConfig recConfig);

    //! Returns: 0 if all was ok
    //           -1 in other case
    int stopRecording (int recordingId);

    //! Return: url to listen the streaming
```

```
    //null in other case
    string startStreaming (int id);

    //! Return: bytes of thumb
    //null in other case
    ByteSeq getThumbRecording (int id);
};
```

RecordingManager no realiza la tarea específica de grabar si no que se lo encarga al componente *Recorder* (sección 3.1.3). La figura 3.2 muestra la secuencia de interación entre estos componentes tras recibir una petición de grabación de otro componente, como por ejemplo *Eldercare*.

Figura 3.2: Secuencia de interacción entre *RecordingManager* y *Recorder*.

Esta funcionalidad para una aplicación como *Eldercare* es muy interesante. Al producirse una caída en la zona vigilada, el sistema, a parte de avisar a las personas interesadas, almacena un histórico de alarmas que puede ser consultado desde diferentes plataformas, incluida un teléfono móvil Android.

Base de datos

Como se ha comentado anteriormente, toda la gestión de grabaciones se lleva en una base de datos. Se ha utilizado el motor de *MySQL* para generar las tablas y relaciones necesarias. En la base de datos se guardan todos los datos relevantes de una grabación exceptuando los ficheros de vídeo e imágenes, que se guardan directamente en el sistema de ficheros (únicamente se guardan las referencias a los *paths* absolutos).

Como puede observarse en la figura 3.3 el diseño de la base de datos se compone de 4 tablas: en una se guarda toda la información referente a las grabaciones (*recording_recordings*), en otra se guarda toda la información referente a alertas (*recording_events*), en la tercera se guarda qué alertas pertenecen a qué grabaciones

(*event_recordings*) y por último en la tabla *recording_type_events* se define los tipos de alarmas existentes (de sistema y alarma generada por componente). Las alarmas generadas por el sistema son aquellas que hacen referencia a notificaciones generales como falta de espacio en disco duro, error de conexión de red, falta de memoria, etc. Las alarmas generadas por componentes, son alarmas específicas que puede lanzar cualquier componente del sistema como desconexión de la cámara, detección de movimiento, límite de conexiones permitidas por un componente, etc.

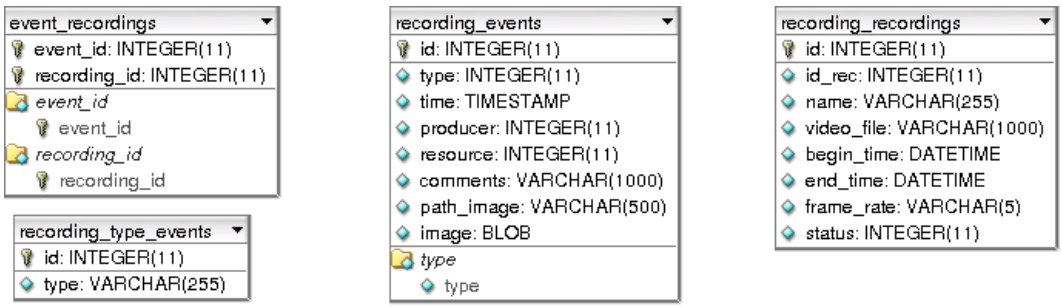

Figura 3.3: Diseño de la base de datos de grabaciones

3.1.3. Componente *Recorder*

La función de este componente es recibir peticiones para grabar vídeo a través de la interfaz slice *Recorder*. Estas grabaciones son configurables, se puede seleccionar la resolución, los fotogramas por segundo, la duración, el protocolo (por ejemplo v4l) y la cámara de la que se desea grabar.

```
interface Recorder
{
    //! Returns: ID of recording if all was ok
    // -1 in other case
    ["ami"] RecorderConfig startRecording(
    RecorderConfig recConfig);

    //! Returns: 0 if all was ok
    // -1 in other case
    int stopRecording (int idRecording);
};
```

Hasta ahora el componente *Recorder* sólo permitía los protocolos v4l y v4l2 y grabar de un dispositivo local a la máquina donde se ejecutara el componente. El aporte de

este trabajo fin de máster ha consistido en que este componente sea capaz de grabar a partir de un flujo de vídeo obtenido de un *CameraServer*. Para ello, este nuevo tipo de grabador llamado *ImageRecorder* implementa la interfaz *ImageConsumer* de modo que se subscribe a las imágenes de una cámara. Nada más comenzar, el grabador abre un fifo sobre el que escribe las imágenes según las recibe y por otra parte lanza la herramienta mencoder para que obtenga las imágenes del fifo y las convierta en un fichero de vídeo de tipo AVI.

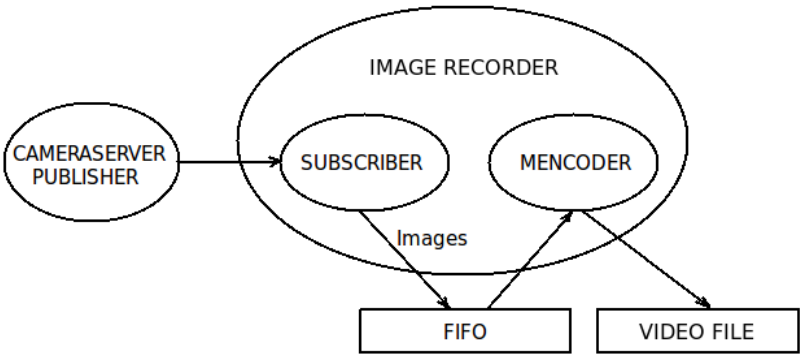

Figura 3.4: Funcionamiento del grabador *ImageRecorder*.

Con este sencillo mecanismo el componente *Recorder* mejora su usabilidad ofreciendo la posibilidad de grabar desde una nueva fuente de imágenes.

3.1.4. Componente *Calibrator*

Un aspecto necesario para este proyecto ha sido la calibración de las cámaras. El proceso de calibración consiste en obtener los *parámetros intrínsecos* y *parámetros extrínsecos* de las mismas. En el caso de los parámetros intrínsecos es necesario conocer la *distancia focal* en cada eje y el tamaño del píxel central del plano imagen. Para los parámetros extrínsecos se necesita la posición 3D de la cámara y la orientación.

El componente *Calibrator* parte del esquema *extrinsics* de la plataforma *Jderobot-4.3*, cuya funcionalidad consiste en ajustar los parámetros extrínsecos e intrínsecos de una cámara comparando la imagen real de la misma con una imagen virtual de lo que debería estar viendo según sus parámetros definidos en un fichero de configuración. El tratamiento de la información tridimensional se realiza mediante la biblioteca *Progeo* (ver sección 3.4).

Migrar esta funcionalidad a la nueva plataforma ha sido otro aporte específico de este trabajo fin de máster a la plataforma, ya que para calibrar las cámaras de *Eldercare* se ha utilizado este nuevo componente *Calibrator*.

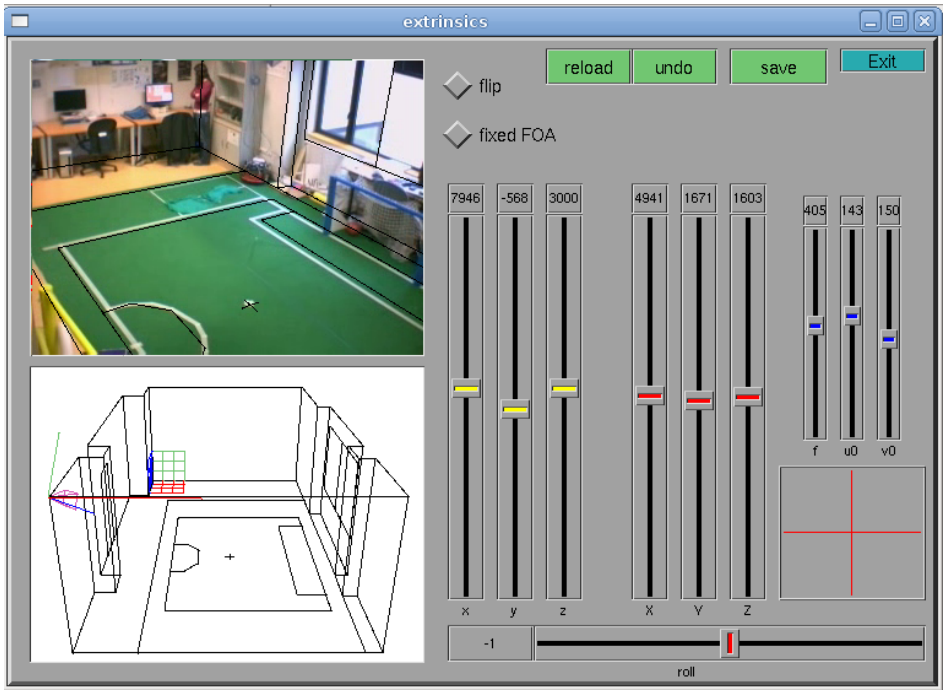

Figura 3.5: Interfaz gráfica del componente *Calibrator*.

En la figura 3.5 se puede ver la interfaz gráfica del componente. Ésta juega un papel fundamental ya que a través de ella se modifican los parámetros de la cámara, que son la información útil. En primer lugar se introducen unos parámetros de entrada aproximados y las líneas del mundo simulado a través de un fichero de configuración. Después se visualiza la imagen real y superpuesta en ella la imagen virtual. A través de los deslizadores y el joystick los parámetros pueden ajustarse hasta que coincidan ambas imágenes.

Una vez ajustada la imagen virtual, pulsando el botón de guardar el componente almacena un fichero de configuración de la cámara con los parámetros ajustados. Este mismo fichero se utiliza en la configuración de *Eldercare*.

3.2. Biblioteca GTK

La biblioteca *GTK*[4] (o Gimp Toolkit) es una de las que compone el conjunto *GTK+* y permite crear interfaces gráficas de usuario para los programas.

Proporciona una herramienta denominada *glade*[5] (figura 3.6), mediante la cuál de manera visual se pueden crear objetos gráficos en sistemas de ventanas. Estos objetos se denominan *widgets* y ejemplos son botones, etiquetas, barras de desplazamiento, etc.

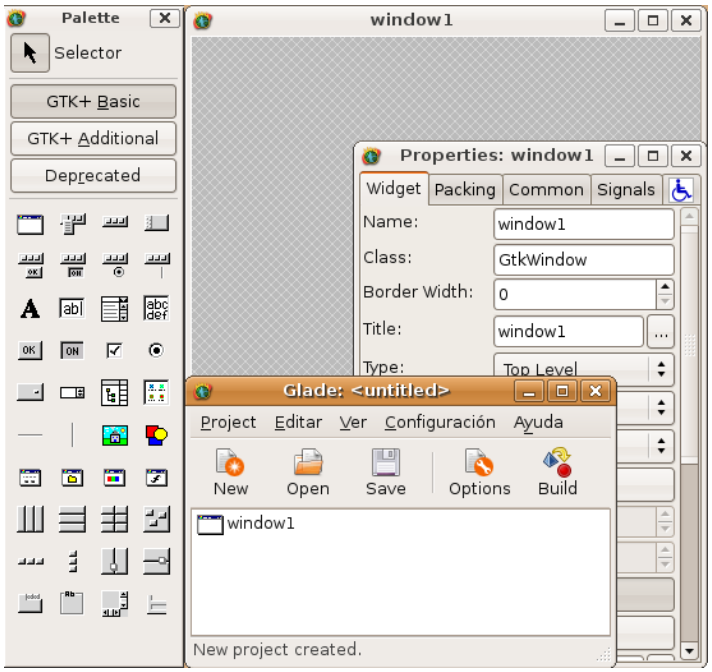

Figura 3.6: Glade.

Los eventos en *GTK* se materializan a través de señales asociadas a manejadores. *Glade* permite asociar las señales de determinado *widget* a sus funciones manejadoras o *callbacks*.

La interfaz de usuario de *Eldercare* se ha diseñado con esta librería y la propia aplicación carga el fichero *.glade*, que define la interfaz gráfica diseñada con *glade*.

[4] www.gtk.org/
[5] http://glade.gnome.org/

3.3. Librería OpenCV

OpenCV[6] es una biblioteca originalmente desarrollada por Intel para visión artificial. La biblioteca es de código abierto y multiplataforma. Su versión actual y la utilizada para este proyecto es la 1.1.0.

La biblioteca ofrece algunos algoritmos habituales en la visión computacional y además una colección de tipos de datos de alto nivel como listas, matrices, árboles, imágenes, etc. Las funciones de *OpenCV* para el tratamiento de imágenes pueden clasificarse en:

- Procesado de imágenes: operaciones morfológicas, bordes, filtros, conversión, histogramas, emparejamiento, etc.
- Análisis estructural: geometría computacional, subdivisiones planares, etc.
- Detección de movimiento y seguimiento de objetos: acumulación de fondo, plantillas de movimiento, flujo óptico, estimadores, etc.
- Reconocimiento de patrones y detección de objetos.
- Calibración de cámaras y reconstrucción 3D.

Concretamente, en este proyecto se han utilizado funciones de conversión, de manejo de histogramas para el aprendizaje de color y operaciones morfológicas sobre imágenes, como la dilatación o la resta de imágenes.

3.4. Biblioteca Progeo

La biblioteca de geometría proyectiva Progeo es utilizada en la aplicación para relacionar el mundo de las imágenes (2D) con el mundo real (3D). Esta relación se consigue gracias a dos funciones principales:

- *Proyectar*. Esta función permite realizar la proyección óptica de un punto 3D del espacio en el plano imagen de una de las cámaras, obteniendo así un punto 2D perteneciente a ese plano. Con ese punto 2D es posible obtener el píxel de la imagen correspondiente.

[6] https://code.ros.org/svn/opencv/trunk/opencv

- Función *Retroproyectar*. Esta función permite obtener la recta de proyección que une el centro óptico de una cámara con el punto 3D que representa un punto 2D de su plano imagen. Ese punto 2D se corresponde con un píxel en la imagen de esa cámara y mediante esta función se puede llegar a obtener el punto 3D del que es proyección en el plano.

Progeo se basa en el *modelo de cámara Pinhole* para realizar estas transformaciones, que se puede observar en la figura 3.7.

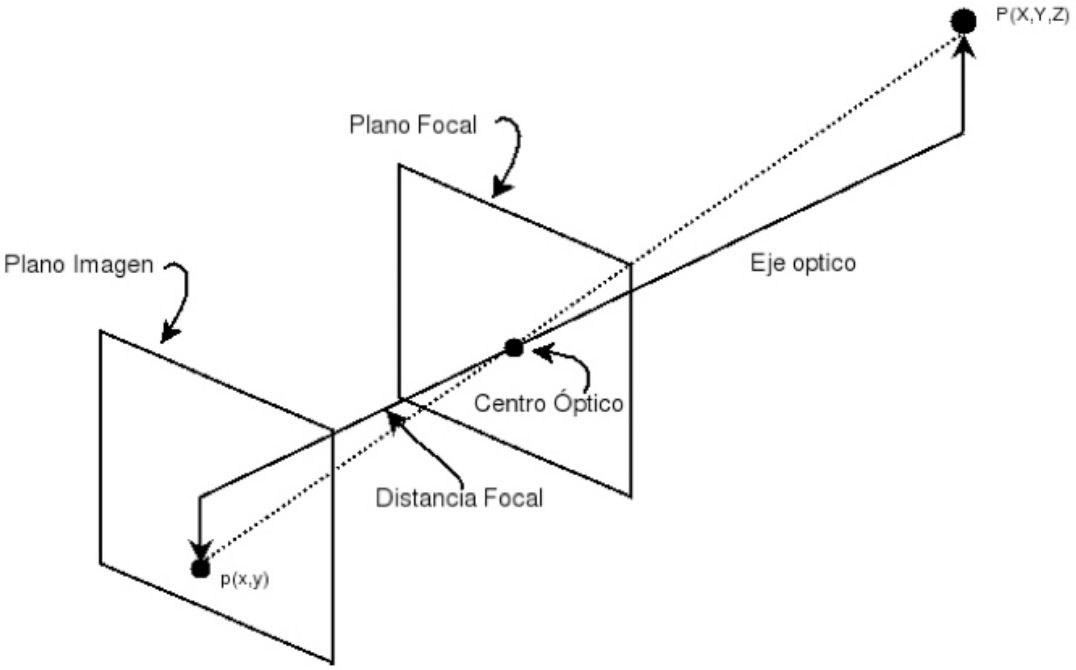

Figura 3.7: Modelo de cámara Pinhole

En este modelo se asume que cualquier punto $P(x, y, z)$ se proyecta en el plano de imagen a través de otro único punto llamado *Centro óptico*. La recta que une el punto P y el centro óptico se denomina *Línea de proyección* e intersecta al plano imagen justo en el píxel $p(x, y)$, que es la proyección de $P(X, Y, Z)$. El centro óptico está situado a la *Distancia focal* del plano imagen. Este modelo lo completan el *Eje óptico*, que es una línea perpendicular al plano imagen y que atraviesa al centro óptico, y también el *Plano focal*, que es el plano perpendicular al eje óptico cuyos puntos no se proyectan en el plano imagen, incluyendo al centro óptico.

Progeo proporciona además los tipos de datos *Punto2D* y *Punto3D* para la representación de puntos en el plano y en el espacio. También proporciona los tipos *CamaraPinhole* y *CamaraStereoPinhole*. Ambos tipos se utilizan para simular el uso de cámaras y pares estéreo en el entorno tridimensional.

Esta librería se ha utilizado para la generación de imágenes virtuales del mundo 3D y las operaciones necesarias para relacionar el mundo 3D con las imágenes 2D.

3.5. Android

Android es un sistema operativo orientado a dispositivos móviles basado en una versión modificada del núcleo Linux. Inicialmente fue desarrollado por Android Inc., compañía que fue comprada después por Google, y en la actualidad lo desarrollan los miembros de la Open Handset Alliance (liderada por Google).

A día de hoy existen más de 200 móviles diferentes con este sistema operativo. Desde luego, parece una buena plataforma para desarrollar (gracias a que es software libre) y para distribuir, ya que poco a poco va comiendo mercado a sus directos competidores (*iPhone* y *BlackBerry*). Por tanto, no es casualidad que se haya optado por este sistema operativo. A principios del año 2010 los responsables de Google-Android hablaban de las ventas: más de 60.000 unidades vendidas cada día en todo el mundo.

Figura 3.8: Logotipo de Android.

Los componentes principales del sistema operativo de Android son:

- *Aplicaciones*: las aplicaciones base incluirán un cliente de email, programa de SMS, calendario, mapas, navegador, contactos, y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

- *Framework de aplicaciones*: los desarrolladores tienen acceso completo a los mismos APIs del *framework* usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

- *Bibliotecas*: Android incluye un conjunto de bibliotecas C/C++ usadas por varios componentes del sistema Android. Estas características se exponen a los desarrolladores a través del *framework* de aplicaciones de Android; algunas son: System C library (implementación biblioteca C standard), bibliotecas de medios, bibliotecas de gráficos, 3d, SQLite, entre otras.

- *Runtime de Android*: Android incluye un conjunto de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".

- *Núcleo Linux*: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, stack de red, y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto del stack de software [Meier, 2010].

Fundamentos de una aplicación Android

La plataforma de Android proporciona diferentes componentes a la hora de programar en función del objetivo de tu aplicación. Android provee cuatro tipos diferentes de componentes:

- *Activity*: Una actividad es el componente más usado en las aplicaciones Android. Típicamente una actividad representa una pantalla individual en el terminal y presenta una interfaz gráfica al usuario. Por ejemplo, en una aplicación de listado de teléfonos utilizaríamos dos actividades. Una para mostrar el listado de nombres y teléfonos y la segunda para mostrar la información detallada

del contacto seleccionado. La navegación entre las pantallas se realiza iniciando nuevas actividades. Cuando una actividad es abierta, la actividad previa es puesta en pausa y agregada el *history stack* y no volverá al estado de ejecución hasta que vuelva a ser invocada.

- *Services*: Un servicio no tiene interfaz gráfica, pero puede ejecutarse en segundo plano por un tiempo indefinido (se asemeja mucho al demonio de los sistemas Linux). Por ejemplo, podemos utilizar un servicio para que vaya capturando cada cierto tiempo la posición GPS y nos avise cuando estemos cerca de algún amigo. Mientras tanto el usuario puede seguir realizando otras tareas.

- *Broadcast receivers*: Este tipo de componentes se utilizan para recibir y reaccionar ante ciertas notificaciones *broadcast*. Este tipo de componentes no tienen interfaz gráfica y pueden reaccionar ante eventos como cambio de zona horarias, llamadas, nivel de batería ... Todos los *receivers* heredan de la clase base *BroadcastReceiver*.

- *Intent*: Este tipo de componentes es una clase especial que usa Android para moverse de una pantalla a otra. Un *Intent* describe lo que una aplicación desea hacer. Cualquiera *activity* puede reutilizar funcionalidades de otros componentes con solo hacer una solicitud en la forma de *Intent*.

La aplicación Android de este proyecto se basa en un conjunto de actividades que se comunican a través de *Intents*, mecanismo que permite realizar el paso de parámetros correspondiente.

3.5.1. Librería OpenGL ES

OpenGL ES[7] (OpenGL for Embedded Systems) es una variante simplificada de la API gráfica OpenGL diseñada para dispositivos integrados tales como teléfonos móviles, PDAs y consolas de videojuegos. La define y promueve el Grupo Khronos, un consorcio de empresas dedicadas a hardware y software gráfico interesadas en APIs gráficas y multimedia.

Esta librería es utilizada por la mayoría de los dispositivos más populares de hoy en día:

- OpenGL ES ha sido seleccionada como la API para gráficos 3D oficial en el sistema operativo Symbian OS y la plataforma para dispositivos móviles Android.

[7] http://www.khronos.org/opengles/

- OpenGL ES 2.0 es la librería gráfica 3D para el dispositivo Nokia N900 con sistema operativo Maemo basado en Linux.
- OpenGL ES es la librería gráfica 3D en el SDK del iPhone.
- OpenGL ES 1.0 más algunas extensiones y con soporte de Cg está disponible para la PlayStation 3 como API gráfica oficial.

Con el objetivo de visualizar la cámara virtual y los resultados del seguimiento 3D de la aplicación *Eldercare* en el cliente móvil se ha utilizado esta librería.

3.6. Eclipse IDE

El entorno de desarrollo integrado (IDE) Eclipse[8] es un entorno de desarrollo de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

[8]http://www.eclipse.org/

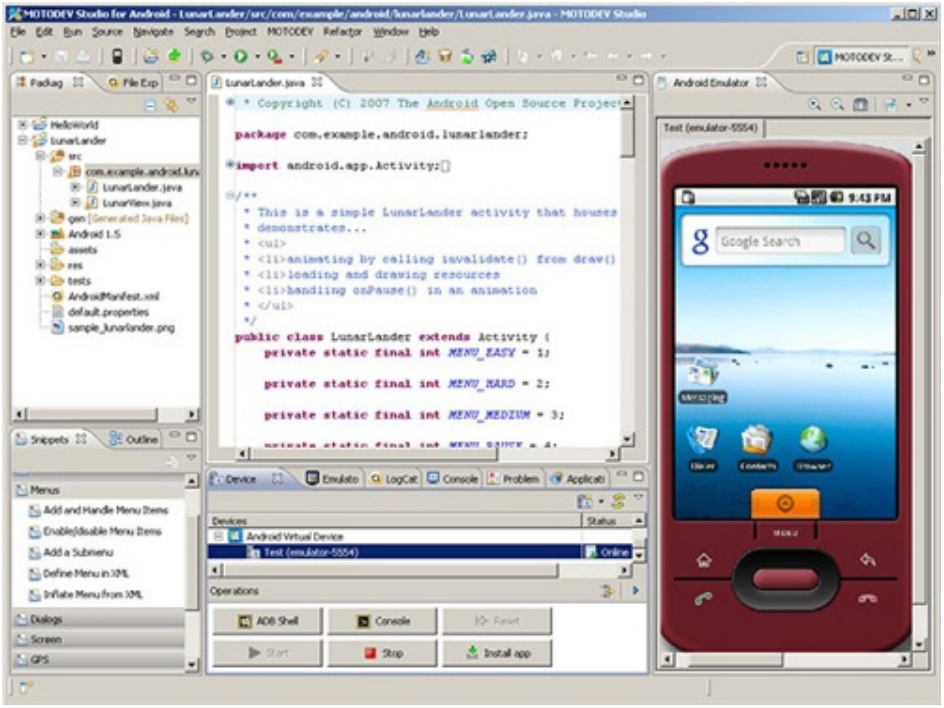

Figura 3.9: Android SDK en Eclipse

Android ofrece un conjunto de herramientas a disposición del desarrollador que hacen más sencilla la programación en dispositivos móviles. Para este proyecto hemos utilizado la versión 1.6 del SDK de desarrollo[9]. Además, *Android* ofrece una extensión muy completa para el entorno de desarrollo de *Eclipse* que facilita el acceso al *API*, al emulador y al depurador. En la figura 3.9 se puede observar el entorno de desarrollo de *Eclipse* con la extensión para *Android* y el emulador arrancado.

El emulador de *Android* es una herramienta muy potente, ya que es muy fiable y con un alto porcentaje de robustez. Aunque es algo lento, como la mayoría de los emuladores, tiene dos grandes puntos a su favor. El primero es que puedes emular casi todas las características reales del móvil: fallos en la red, cámara de fotos, SMS, llamadas o posicionamiento *GPS*. El segundo es su fiabilidad, aplicación que pruebas y funciona correctamente en el emulador, lo hace igual de bien en el dispositivo móvil real.

[9]http://developer.android.com/

Capítulo 4
Descripción técnica del sistema

En el capítulo anterior se presentó la plataforma en desarrollo *Jderobot-5.0* sobre la que se apoya este proyecto y los aportes concretos que este trabajo ha supuesto para ella. En este capítulo se describen en detalle los componentes principales: *Eldercare*, que engloba el corazón de la aplicación: el algoritmo de seguimiento 3D visual y las reglas de generación de alarmas; y el componente *EldercareClient*, que consiste en una aplicación para un teléfono móvil Android que recibe la información generada por el primer componente.

4.1. Diseño global

El sistema está compuesto por varios componentes que interactúan entre sí a través de los interfaces definidos por la plataforma *Jderobot-5.0* y un interfaz definido para esta aplicación llamado *EldercareProvider*. Todos los componentes de *Jderobot-5.0* heredan de la clase *Component* proporcionada por la librería *jderobotice* de la plataforma. En la figura 4.1 se puede ver un diagrama de componentes del sistema.

El componente *Eldercare* recibe las imágenes del componente *CameraServer* a través de la interfaz *ImageConsumer*, definida para recibir las imágenes mediante publicación/subscripción a través del servicio *IceStorm* de *ICE*. La tarea de *Eldercare* es analizar de forma continuada y en tiempo real las imágenes que recibe con la finalidad de extraer información tridimensional de la escena, concretamente la posición 3D de las personas que se encuentren en ella. Una vez obtenida esta información comprueba si alguna posición 3D está muy cerca del suelo, lo que significa que hay una persona caída en él. Si esto sucede, *Eldercare* envía un aviso, como puede ser un email, y comienza a grabar vídeo que quedará almacenado junto a la fecha y hora en que se ha producido. Como vemos en el diagrama 4.1, *Eldercare* utiliza el componente *RecordingManager* para grabar vídeo. Éste almacena la información de la grabación en una base de datos

y a su vez encarga la tarea específica de grabar el vídeo al componente *Recorder*.

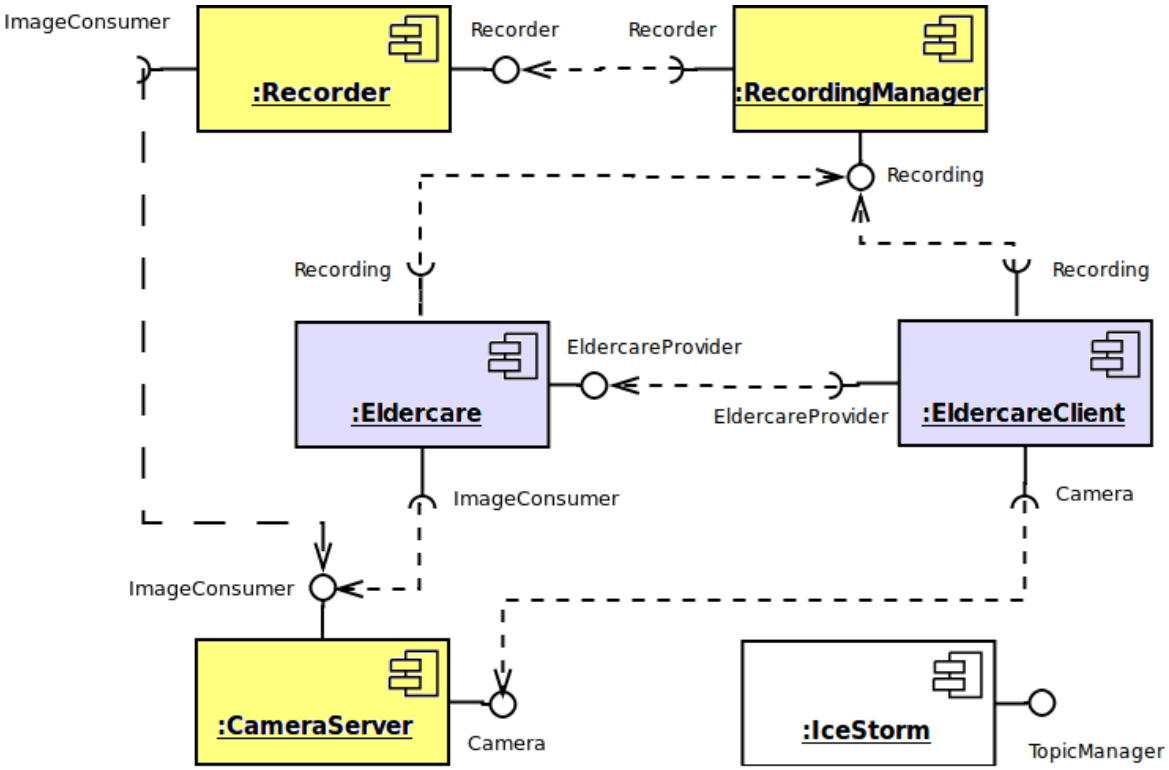

Figura 4.1: Diagrama de componentes del sistema.

El usuario, al ser alertado por el aviso, puede entrar en la aplicación *EldercareClient* desde un móvil para ver lo sucedido. Concretamente puede visualizar el vídeo grabado o incluso ver en directo qué está pasando en la zona monitorizada. Además, esta aplicación cliente también permite ver la cámara virtual del sistema a través de un visor OpenGL.

La comunicación entre el software que corre en el ordenador y la aplicación del teléfono es fluida, iteroperan suavemente gracias al middleware *ICE*.

Una vez comentada la funcionalidad completa del sistema, este capítulo se va a centrar en describir los componentes *Eldercare* y *EldercareClient*.

4.2. Componente Eldercare

Eldercare es el componente principal de la aplicación ya que implementa una gran parte de la funcionalidad del sistema. Su diseño utiliza el Modelo Vista Controlador

(MCV), de modo que mantiene muy desacoplada su interfaz gráfica de la lógica de la aplicación. Esto es interesante puesto que *el sistema está pensado para ejecutar sin interfaz gráfica*, ya que ésta es un elemento de ayuda a la depuración de la aplicación.

4.2.1. Diseño: Modelo Vista Controlador

En la figura 4.2 vemos el diagrama clases del componente. Las clases en amarillo están definidas por la plataforma y las azules pertenecen a la aplicación.

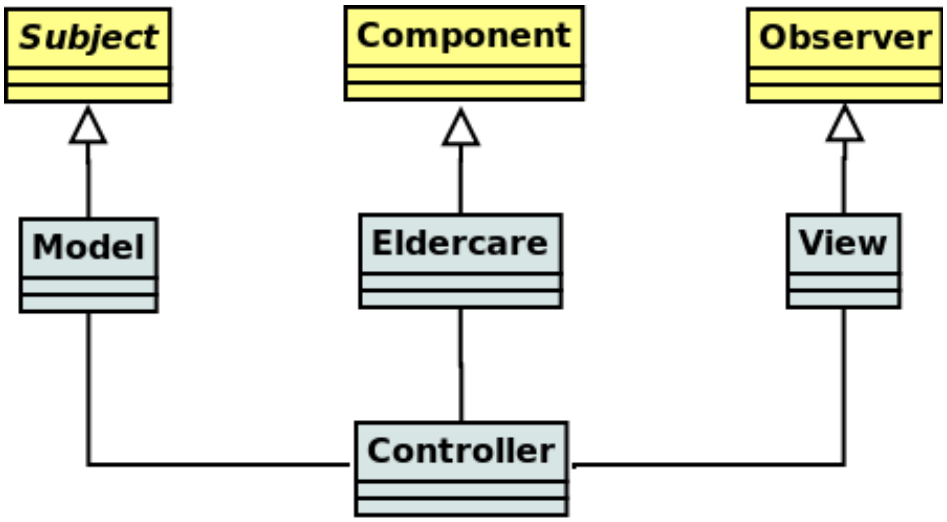

Figura 4.2: Diagrama de clases del componente Eldercare.

El modelo es la clase que recubre el algoritmo evolutivo de seguimiento tridimensional. En el modelo se definen todas las constantes y parámetros de configuración. Esta clase ofrece una serie de métodos para cambiar dichos parámetros de configuración y ejecutar el algoritmo.

La vista es la clase que se encarga de actualizar la interfaz gráfica en cada iteración. Desde ella se controlan los cambios de estado de los distintos elementos de la interfaz gráfica, como por ejemplo los botones y los deslizadores o sliders.

La clase que conecta el modelo con la vista es el controlador. A través de él los cambios producidos en el modelo se transmiten a la vista y viceversa. Una funcionalidad añadida al controlador es manejar cada cuánto tiempo se generan las alarmas si el algoritmo está detectando una situación de peligro de manera continuada.

La clase *Eldercare* es la principal, ya que crea el componente y se encarga de conectarse a los demás componentes del sistema. Obtiene las imágenes, manda ejecutar el algoritmo tras recibir imágenes nuevas y si se ha activado la alarma, ordena grabar vídeo.

4.2.2. Modelo: algoritmo de seguimiento visual 3D

El modelo alberga el algoritmo de seguimiento visual 3D de personas, que es el corazón de la aplicación *Eldercare*. Gracias a él el sistema obtiene la información tridimensional necesaria para saber si alguna persona de la zona monitorizada se encuentra en peligro.

Este algoritmo tiene varios antecedentes como vimos en el capítulo de introducción (sección 1.4). La última versión es la propuesta en mi proyecto fin de carrera para el título de Ingeniero en Informática *Seguimiento visual de personas mediante evolución de primitivas volumétricas*[Marugán, 2010]. Esta versión es mucho más robusta que las anteriores debido a que introduce primitivas volumétricas para estimar a las personas y se genera un aprendizaje de color por cada cámara, resolviendo además la carencia que había para aprender colores claros y oscuros.

Al reimplementar el algoritmo en la plataforma *Jderobot-5.0* se ha revisitado reajustando algunos parámetros y añadiendo algunas mejoras. Las mejoras más destacables son:

- *Parametrización del número de cámaras* que puede utilizar el algoritmo. Antes el número de cámaras del sistema estaba fijado en cuatro y, tras esta mejora, al sistema se le pueden agregar tantas cámaras como el rendimiento temporal permita sin perder el tiempo real.

- *Cambio al modelo HSV de color para realizar el filtro de movimiento.* El modelo RGB presentaba algunas limitaciones para tolerar los cambios de iluminación producidos en breve período de tiempo. Definiendo la tolerancia a los cambios de iluminación sobre el modelo HSV el filtro de movimiento se comporta de manera más robusta ante dichos cambios. Esta mejora se describe más en detalle en la sección 5.5.1 del capítulo de experimentos.

- *Paralelización del filtro de movimiento* dentro del mismo componente pero en un hilo de ejecución separado. Esta paralelización mejora notablemente el rendimiento temporal del algoritmo (ver sección 5.4.3).
- *Paralelización de las acciones para generar las alarmas.* Estas acciones tenían asociadas un ligero retardo que hacían que el algoritmo no ejecutara fluidamente cuando se detectaba el peligro y se accionaba la alarma. Al ejecutarlas en un hilo separado ese retardo no afecta al algoritmo.

En cuanto a nueva funcionalidad, ahora es capaz de generar alarmas de mayor utilidad para un uso del sistema en situaciones reales. Hasta ahora el sistema sólo activaba una alarma visual en la interfaz gráfica de la aplicación y una alarma auditiva en la zona vigilada. Ahora se ha incorporado la capacidad de avisar al usuario de la situación de peligro en su propio teléfono móvil y que se pueda ver lo ocurrido a través de grabaciones o en tiempo real.

Otra funcionalidad añadida a *Eldercare* es la de servir las posiciones 3D de las personas a través de un interfaz slice definido a propósito para esta aplicación. Esto permite que cualquier sistema que conozca este interfaz pueda pedirle a *Eldercare* el estado en que se encuentra el seguimiento tridimensional.

A continuación se presenta el algoritmo de seguimiento, descrito con más detalle en [Marugán, 2010].

Algoritmo de seguimiento

Los llamados algoritmos genéticos o evolutivos son algoritmos de búsqueda que tratan de hallar una solución óptima al problema planteado. Estos algoritmos combinan las propiedades de las hipótesis en el espacio de soluciones para obtener nuevas generaciones más robustas. Son algoritmos iterativos que guardan cierta similitud con la evolución natural y se utilizan a menudo en tareas de optimización, diseño de controladores borrosos o sistemas de aprendizaje automático.

Estos algoritmos cuentan con un conjunto o población de N individuos donde cada individuo n_i tiene asociado un determinado peso o salud h_i, que indica cómo de buena es esa solución y se actualiza en cada iteración del proceso. Dependiendo del propósito del algoritmo, un determinado individuo puede ser en si mismo una posible solución

(enfoque *Pittsburgh*) o bien sólo una contribuyente a la misma (enfoque *Michigan*).

Los algoritmos genéticos se dividen principalmente en dos fases:

1. *Generación de la próxima población.* Se aplican operadores genéticos para obtener la nueva población de individuos.

2. *Computación de la salud.* Se calcula la salud para cada uno de los individuos en función de las observaciones realizadas.

El algoritmo de esta aplicación es un algoritmo de búsqueda en el espacio 3D que genera hipótesis y las valida hasta encontrar las soluciones óptimas. Las hipótesis en este algoritmo son prismas con cuatro grados de libertad (ver imagen 4.3).

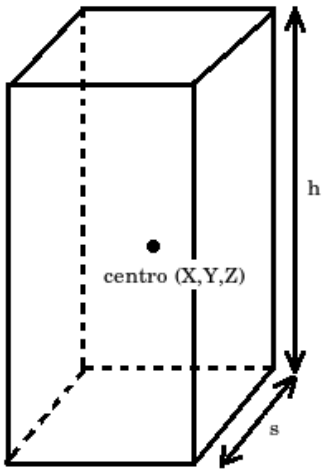

Figura 4.3: Prisma y grados de libertad.

Estos prismas son generados a partir de las regiones de interés (ROI, figura 4.4 (b)) extraídas de las imágenes basándose en un filtro de movimiento. Por cada región de interés de una imagen se generan mediante abducción múltiples prismas en el espacio 3D en diferentes posiciones y tamaños cuyas proyecciones coinciden con la región (ver figura 4.5). Después, proyectando estos prismas en las otras cámaras se puede valorar cuáles verdaderamente se encuentran en la zona 3D donde está la persona que genera ese movimiento y descartar aquellos que no lo están. A partir de los prismas que no hayan sido descartados se generan las razas, que son las encargadas del seguimiento de las personas.

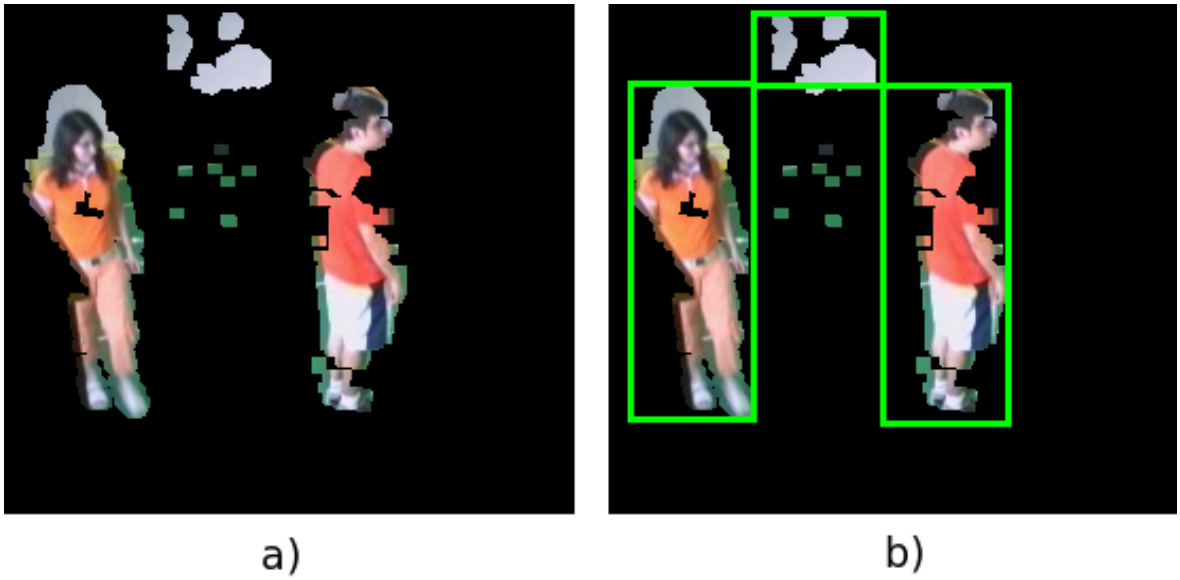

Figura 4.4: (a) Filtro de movimiento, (b) Regiones de interés.

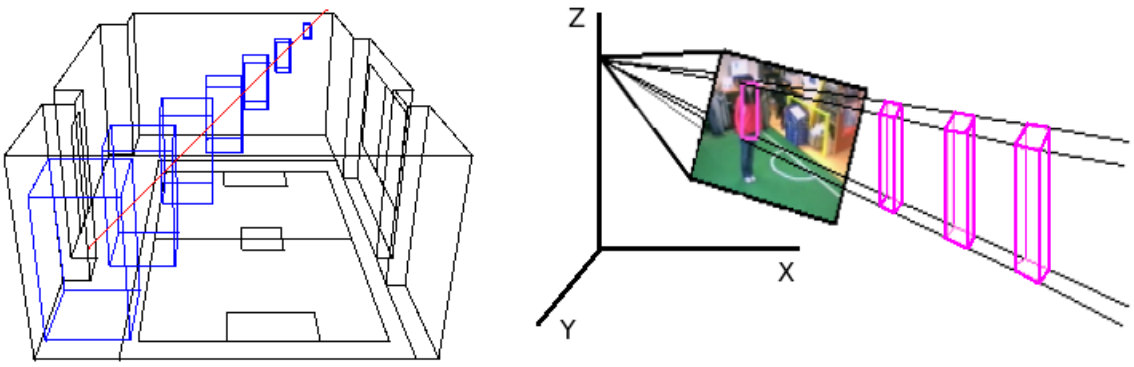

Figura 4.5: Abducción.

En este contexto una raza es un conjunto de individuos (prismas) que mantienen el seguimiento 3D de una persona, estiman su tamaño y almacenan información sobre ella. Cada individuo representa una posible solución del problema y es calificado con un valor de salud para indicar cómo de buena es esa solución.

A continuación se presenta un diagrama de flujo del algoritmo de seguimiento.

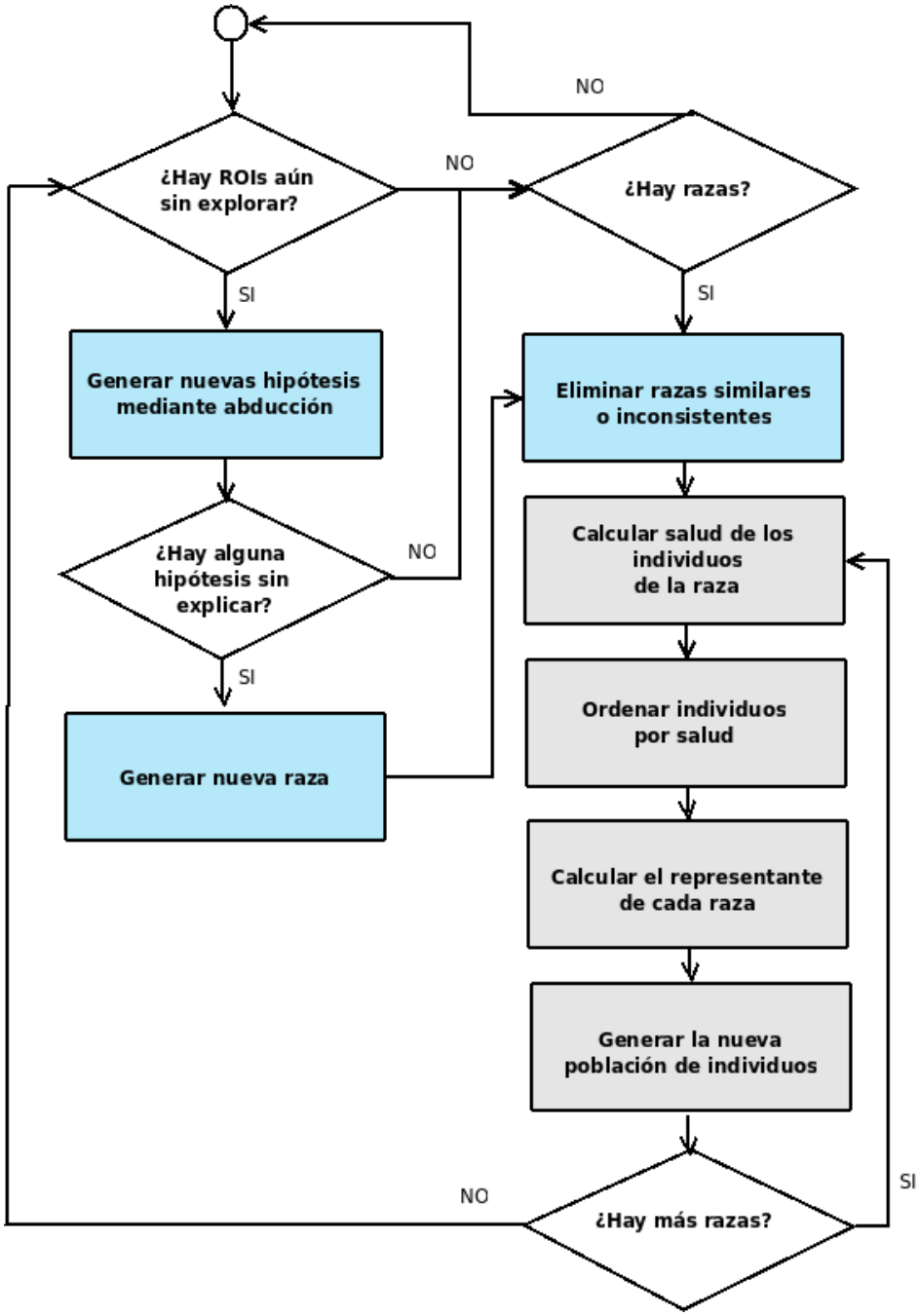

Figura 4.6: Diagrama de flujo del algoritmo.

Las cajas del diagrama de flujo (figura 4.6) sombreadas en azul corresponden con los pasos de generación de hipótesis 3D (prismas exploradores), creación y eliminación de razas. Las sombreadas en gris corresponden a los pasos de evaluación y evolución de los individuos una raza (población de explotadores), lo que permite su adaptación al movimiento de la persona.

La creación de una raza se produce a partir de un movimiento no explicado en la

zona monitorizada, esto es, si de los prismas generados por abducción a partir de las regiones de movimiento alguno es corroborado en al menos dos cámaras, el sistema considera que ha aparecido una nueva persona en la escena. En el caso contrario, una raza es eliminada cuando su salud es baja repetidamente, es decir, cuando las imágenes no coinciden o no corroboran la estimación que dicha raza está realizando.

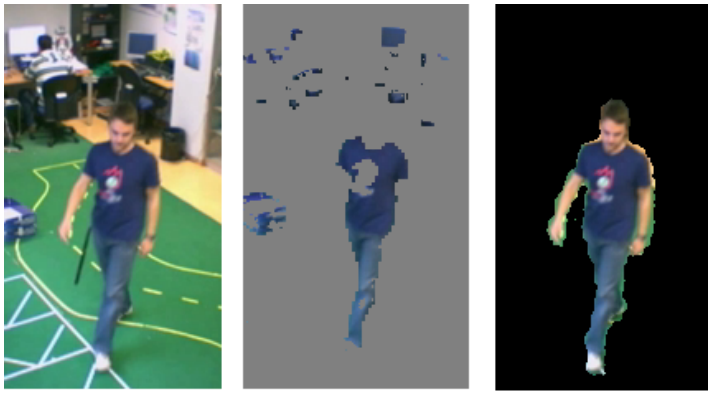

Figura 4.7: Filtros - (a) Imagen original, (b) Filtro de color, (c) Filtro de movimiento.

El cálculo de salud o evaluación de una raza es algo crucial en este tipo de algoritmos ya que determina qué hipótesis son acertadas y cuáles no lo son. En este caso, la única información que se tiene de la realidad son las imágenes de las cámaras, por lo que el cálculo de salud se apoya en ellas. Concretamente extrae información de movimiento y de color generando los filtros de movimiento y color que permiten valorar cuantitativamente a una raza. El filtro de movimiento es genérico para todas las razas, mientras que en el caso del color se asocia uno diferente a cada raza. Por tanto, cada raza que sigue a una persona tiene un color asociado.

Para evaluar los prismas que componen una raza, se proyectan en cada cámara y se observa la cantidad de píxeles de la proyección que pasan el filtro de movimiento y el de color específico de la raza para indicar cómo de buena es su estimación (ver imagen 4.8). La salud total de un individuo combina el resultado del filtrado por movimiento y color en las cuatro imágenes.

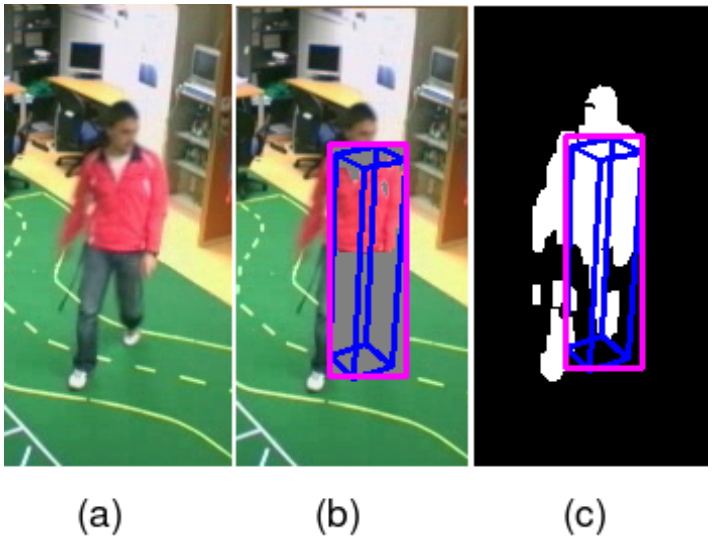

Figura 4.8: Cálculo de salud - (a) Imagen original, (b) Prisma proyectado y filtro de color, (c) Prisma proyectado y filtro de movimiento.

En cuanto a la evolución de las razas, se utilizan operadores genéticos que transforman la población de la iteración anterior en una nueva que se adapte a la nueva posición de la persona. Estos operadores son elitismo y ruido térmico:

- *Elitismo*: consiste en mantener sin cambios los mejores individuos de la raza en la siguiente población. Esto permite que se tomen como referencia para la solución a la localización de la persona y para la generación de nuevas hipótesis cercanas a ellos. Además, el prisma representante de la raza se calcula como media de los prismas elitistas.

- *Ruido térmico*: escoge aleatoriamente individuos elitistas y les aplica una pequeña modificación también aleatoria. La modificación se hace en cada uno de los grados de libertad del individuo, en este caso cinco (X,Y,Z,h,s). Este operador es necesario para la adaptación al movimiento 3D de la persona, ya que explora el tamaño adecuado y la posición adecuada de ésta.

De este modo, las poblaciones o razas evolucionan continuamente hasta converger a la solución óptima en cada instante.

Una vez que se obtiene la información relevante de la escena, que son las posiciones 3D de las personas que aparecen en ella, la regla para activar la alarma es sencilla. En cada iteración el algoritmo comprueba que si alguna de las posiciones 3D tiene una

coordenada Z inferior a 30 centímetros. Si esto es así, se dispara la alarma. Este umbral de 30 centímetros permite diferenciar entre una persona sentada en el suelo y tumbada en él (esto se muestra en el capítulo de experimentos 5).

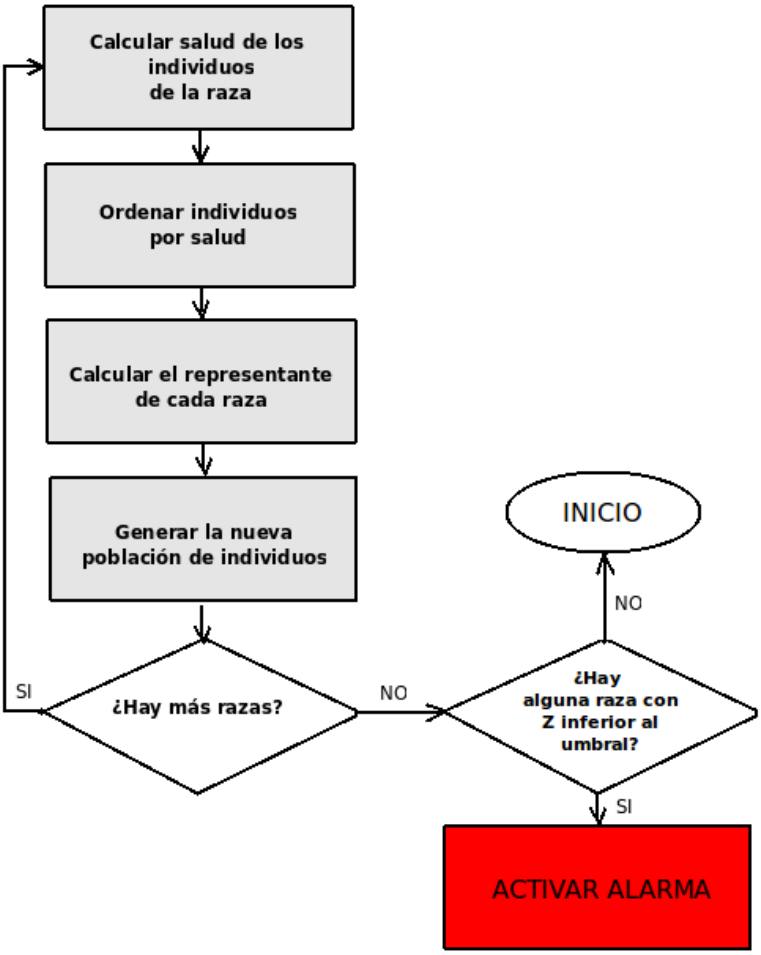

Figura 4.9: Diagrama de flujo del algoritmo para la generación de la alarma.

El diagrama 4.9 vemos los pasos de evaluación y evolución de las razas presentados anteriormente. Tras evaluar todas las razas, el algoritmo hace la comprobación con la coordenada Z de cada raza para activar la alarma o no.

Explicados los pasos principales del algoritmo, la siguiente sección se va a centrar en describir los aspectos técnicos de la nueva funcionalidad del sistema.

4.2.3. Clase Eldercare

La primera tarea de esta clase es obtener las imágenes que necesita el algoritmo para poder ser ejecutado. Para ello implementa la interfaz *ImageConsumer* que

permitirá suscribirse a cada cámara y recibir nuevas imágenes en cuanto sean publicadas. Este servicio de publicación/subscripción se llama *IceStorm* y lo proporciona el middleware ICE. La interfaz *ImageConsumer* únicamente posee un método que debe implementar la clase *Eldercare* y que es llamado por el publicador, que en este caso es el componente *CameraServer*. A través de este método recibe una imagen para proporcionársela al algoritmo. Aquí tenemos un ejemplo de la configuración de una cámara para *Eldercare*.

```
EldercareApp.CameraA.Proxy=IceStorm/TopicManager:default -p 10000
EldercareApp.CameraA.TopicName=cameraA
```

Eldercare debe conocer dónde se encuentra ejecutando el servicio *IceStorm* para mandarle la petición de subscripción de la cámara que desee, en este caso la cámara A. Es necesario obtener un *proxy* por cada cámara y suscribirse por separado. Esta es la parte del código que realiza la subscripción:

```
for(int k=0; k<Model::NCAMS; k++){
    ...
    proxy[k] = adapter[k]->addWithUUID(
     imageConsumers[k])->ice_oneway();
    try {
        topic[k] = topicManager[k]->retrieve(topicName);
        IceStorm::QoS qos;
        topic[k]->subscribeAndGetPublisher(qos, proxy[k]);
    }
    catch (const IceStorm::NoSuchTopic& ex) {
        std::cerr << ex << std::endl;
    }
    adapter[k]->activate();
}
```

Tras recibir nuevas imágenes el algoritmo puede ejecutar una nueva iteración y realimentar el estado del seguimiento en ese instante. Otra de las tareas de esta clase es servir las posiciones 3D del seguimiento que recibe por parte del Modelo. Para realizar esto ha sido necesario definir una interfaz slice a la que hemos llamado *EldercareProvider*. Aquí tenemos parte de la definición:

```
//! Description of the race state
class RaceState
{
   //! Identifier of race
   string id;
   //! Position
   Point3D position;
   //! Color
   ColorRGB color;
};

//! A sequence of race states
sequence<RaceState> RaceSequence;

interface EldercareProvider
{
   //! Returns the list of race states
   idempotent RaceSequence getRaceStates();
};
```

A través de este interfaz cualquier componente puede obtener la lista de razas del algoritmo ejecutando en tiempo real. A cada estado de una raza se le asocia un identificador único, el color característico de la raza y su posición 3D. Esta información podría ser de utilidad para su análisis, como por ejemplo el análisis de las trayectorias.

Como hemos visto anteriormente, la alarma se genera en el modelo y a través del controlador llega a la clase *Eldercare*. Esta clase es la encargada de manifestar la situación de peligro. Actualmente es capaz de generar tres tipos de alarmas: *visual*, *auditiva* y *envío de correo electrónico*. La alarma visual se establece en la interfaz gráfica de la aplicación, lo cual no es de gran utilidad para un uso real del sistema, sino para la depuración del mismo. La alarma auditiva es útil para alertar de la situación de peligro al entorno cercano de la zona monitorizada. Por ejemplo, si una persona vive sola en un bloque de pisos y cae al suelo, la alarma auditiva alertaría de la situación a los vecinos cercanos a la vivienda, que podrían ofrecer al accidentado unos primeros auxilios. La alarma vía email está pensada para avisar de forma telemática tanto a un servicio de asistencia como a un pariente. Otro tipo de alarmas como envío de sms podrían ser incorporadas al sistema de manera inmediata, si se desea ese servicio.

Para enviar un email se utiliza el comando de Unix *mailx*, un sencillo programa *Mail User Agent* para mandar y enviar correos electrónicos. *Eldercare* posee su propia

cuenta en el servidor GSYC de esta universidad. Esto permite que automáticamente entre en el servidor y envíe un correo electrónico como alarma. Para poder entrar automáticamente se ha necesitado un mecanismo de clave pública y privada (Public Key Infrastructure) que evite la petición de contraseña por parte del servidor. Este mecanismo consiste en generar un par de claves (pública y privada) con el comando *ssh-keygen*. La clave pública se introduce en el fichero *known_hosts* del servidor y la privada se deja en el directorio *.ssh* del *home* del usuario que ejecuta la aplicación *Eldercare*. Aquí tenemos el comando que ejecuta la aplicación para enviar un email a través de la llamada al sistema *system()*:

```
ssh eldercare@gsyc.es 'echo "Alguien está en peligro."
 | mailx -s "Alarma Eldercare" user@email'
```

Además de estos mecanismos de aviso, el sistema es capaz de grabar vídeo tras detectar el peligro. Para ello el componente *Eldercare* realiza una petición para grabar vídeo al componente *RecordingManager* (sección 3.1.2) y éste almacena la información de la grabación en una base de datos. Esto se realiza desde la clase *Eldercare*, que lanza un hilo de ejecución para ejecutar esta tarea. Únicamente es necesario crear un objeto *proxy* para conectar con *RecordingManager* y realizar la petición. Las siguientes líneas muestran cómo se genera el *proxy* a partir de las propiedades configuradas a través del fichero de configuración del componente *Eldercare*:

```
Ice::ObjectPrx base =
communicator->propertyToProxy(
    "EldercareApp.RecordingManager.Proxy");

if (base==0)
    throw "Could not create proxy (recordingManager)";

jderobot::RecordingManagerPrx rm_prx =
jderobot::RecordingManagerPrx::checkedCast(base);
...
recId = rm_prx->startRecording(recConfig);
```

Esta funcionalidad que implica el envío de un email y la grabación de vídeo se lleva a cabo en un hilo de ejecución separado del proceso principal. Cuando el algoritmo detecta peligro, la clase *Eldercare* crea un nuevo proceso que se encarga de las tareas

de alarma y seguidamente termina. Esto se hace así para no interrumpir la ejecución del algoritmo.

Por otra parte, la aplicación cliente para móvil *EldercareClient* está pensada principalmente para ver qué ha ocurrido tras recibir una alarma del sistema. Sin embargo, también podría usarse como alarma pasiva, puesto que en cualquier momento se puede entrar en la aplicación y ver el listado de alarmas que se han producido.

4.2.4. Vista: interfaz gráfica

La interfaz gráfica es un elemento importante de la aplicación, ya que permite visualizar los resultados del seguimiento 3D y es una herramienta de depuración. En ella fácilmente se puede mostrar en ella por ejemplo los filtros tanto de color como de movimiento. Además, permite cambiar en tiempo de ejecución ciertos parámetros del algoritmo y ver su impacto en el funcionamiento. Sin embargo, el sistema está pensado para ejecutarse sin la interfaz gráfica debido a que es un sistema autónomo y no necesita la supervisión de una persona.

En la interfaz podemos ver las imágenes que obtienen las cuatro cámaras reales y en el centro la imagen de la cámara virtual, en la que se visualiza un esqueleto de la habitación y el resultado del seguimiento en tres dimensiones.

Los botones *A, B, C y D* permiten seleccionar qué cámaras se tienen en cuenta para el seguimiento. Por defecto, las cuatro cámaras se toman en cuenta. El botón *Flip* permite dar la vuelta a las imágenes, ya que las cámaras reales están instaladas boca abajo.

Debajo de estos botones hay una etiqueta de texto que por defecto muestra 'NO DANGER', lo que significa que ninguna de las personas seguidas está caída en el suelo. Si una persona se encuentra tirada en el suelo, el sistema lo detecta y lo indica modificando esta etiqueta para poner 'DANGER' y cambiar su color a rojo.

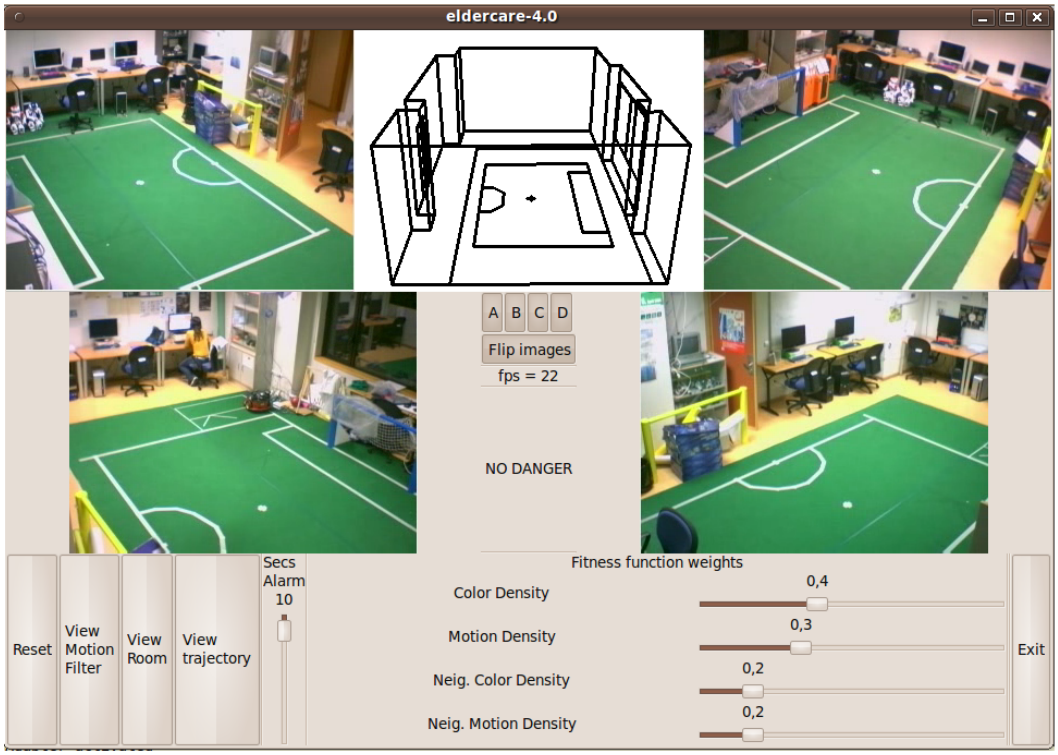

Figura 4.10: Interfaz gráfica de la aplicación.

El botón de *Reset* sirve para resetear el seguimiento actual, eliminando la información que el sistema había guardado hasta el momento. Pulsando el botón *View Motion Filter* se dibujan en blanco aquellos píxeles que superan el filtro de movimiento, además de pintarse en rojo las regiones rectangulares resultado de la segmentación de movimiento. El botón *View Room* activa la visualización de las imágenes de las cámaras virtuales superpuestas sobre las imágenes reales, de este modo es muy fácil comprobar si las cámaras están bien calibradas. El siguiente es el botón *View trajectory*, que sirve para comenzar a ver en la cámara virtual las trayectorias que realizan las personas.

El deslizador *Secs Alarm* permite modificar los segundos entre generación de alarmas cuando se detecta un peligro de manera continuada.

Los cuatro deslizadores que se encuentran al lado del botón *Exit* sirven para ajustar los pesos de la función de cálculo de salud del algoritmo.

Por último, el botón de *Exit* sirve para salir de la aplicación.

4.3. Componente EldercareClient

Uno de los principales objetivos de este trabajo fin de máster era crear un cliente que ejecutara en un móvil y que tuviera acceso a los resultados o eventos que genera la aplicación *Eldercare*. Esto se ha materializado en un nuevo componente enmarcado en la plataforma *Jderobot-5.0* llamado *EldercareClient*. Este componente es una aplicación escrita en Java para un teléfono móvil con sistema operativo Android (ver sección 3.5). Para desarrollarla, se ha utilizado el entorno de desarrollo Eclipse IDE junto con las herramientas del SDK de Android y la librería ICE para Android, todo ello descrito en la sección 3.6.

Esta nueva funcionalidad supone un gran paso en la utilidad del sistema y acercarlo a una aplicación real en el mercado.

4.3.1. Diseño

Las aplicaciones Android normalmente se descomponen en varias actividades. La actividad principal, *Main*, es la primera que se lanza al entrar en la aplicación y consiste en un menú para acceder a las otras actividades, que son las que realizan tareas relevantes.

EldercareClient es un componente más en la plataforma *Jderobot-5.0* y la mayoría de sus actividades están dedicadas a recibir información de otros componentes de *Jderobot-5.0* para ofrecérsela al usuario. Concretamente, esta aplicación cliente recibe datos de tres componentes diferentes, que se indicaron en el diagrama 4.1 presentado al comienzo del capítulo. Éstos son: *CameraServer*, *RecordingManager* y *Eldercare*.

Las actividades se relacionan entre sí ya que unas pueden lanzar a otras. En el diagrama 4.11 vemos las conexiones entre las distintas actividades. Las flechas negras simbolizan la capacidad de lanzar la actividad en la que termina la flecha y las grises la posibilidad de retornar de dicha actividad a la misma que la lanzó.

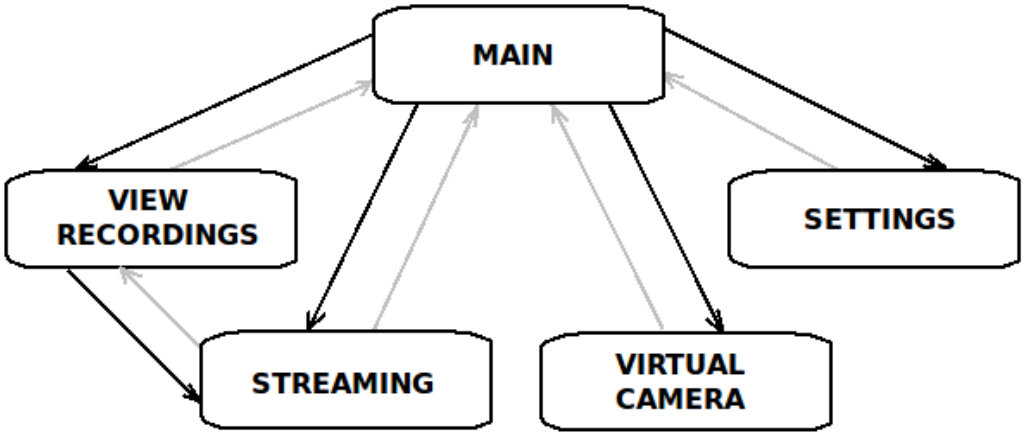

Figura 4.11: Diagrama de interacción entre actividades.

Aquí tenemos el manifiesto del proyecto en Eclipse:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
    "http://schemas.android.com/apk/res/android"
      package="com.urjc.robotica.eldercare"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
         android:label="@string/app_name">
        <activity android:name=".Main"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name=
            "android.intent.action.MAIN" />
                <category android:name=
            "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    <activity android:name="Video"></activity>
    <activity android:name="VirtualCamera"></activity>
    <activity android:name="RecordingList"></activity>
    <activity android:name="RecordingEventList"></activity>
    <activity android:name="PhotoShowing"></activity>
    <activity android:name="Settings"></activity>

</application>

<uses-permission android:name="android.permission.INTERNET">
</uses-permission>

</manifest>
```

Es necesario darle el permiso específico para que la aplicación tenga acceso a internet.

La aplicación se compone de una serie de clases que interactúan entre sí. En la figura 4.12 tenemos el diagrama de clases principales de la aplicación. Las clases color verde pertenecen a las librerías de Android y las azules componen el código fuente de la aplicación *EldercareClient*.

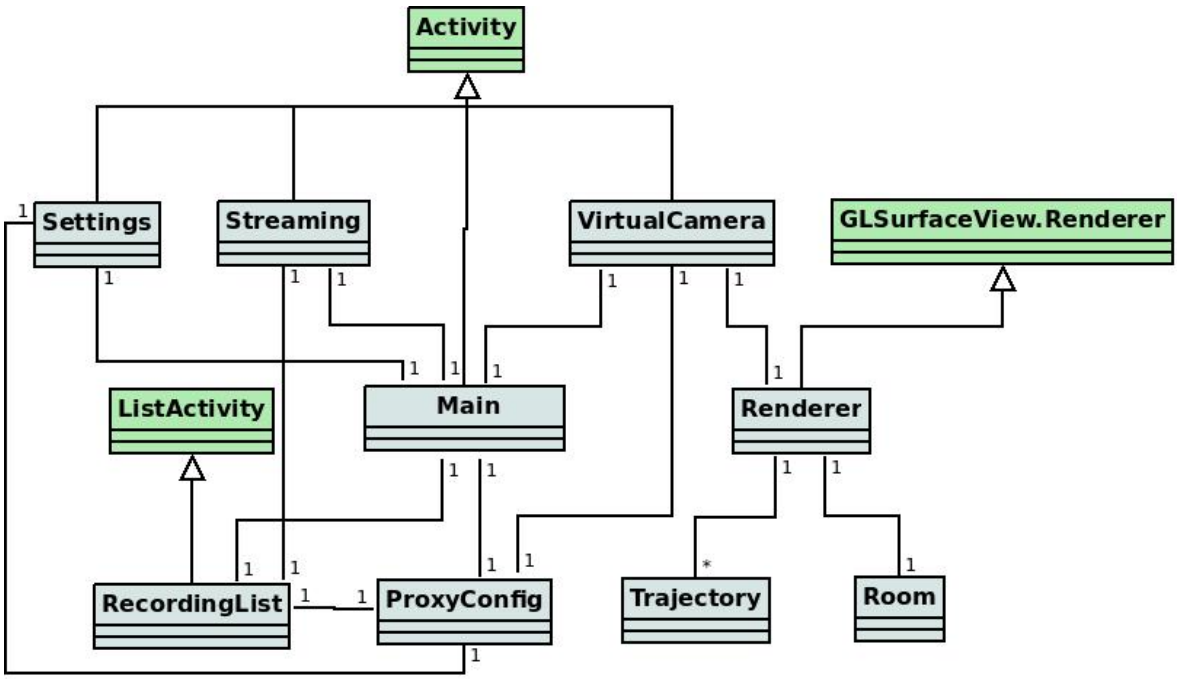

Figura 4.12: Diagrama de clases del componente *EldercareClient*.

Este diseño consiste en una clase por cada actividad y además una clase que almacena la configuración de la aplicación a la que se ha llamado *ProxyConfig*. A través de esta clase todas las demás pueden acceder a estos parámetros configurables. En cuanto a la actividad *VirtualCamera*, encargada de dibujar la habitación y las trayectorias de las personas, necesita un objeto que haga la función de *render*, materializado en la clase *Renderer*. Por último, las clases *Trajectory* y *Room* guardan la información que el *render* consulta para dibujarla. Como indica el diagrama, puede haber más de una instancia de la clase *Trajectory*.

A continuación se explica detenidamente en qué consisten las distintas actividades.

4.3.2. Actividad Main

Esta actividad además de proporcionar el menú para acceder a las demás actividades se encarga de otras tres tareas más: inicializar los parámetros de configuración, comprobar que éstos son adecuados, comprobar que hay conexión a internet y a través de un menú desplegable dar a elegir al usuario qué cámara del sistema quiere visualizar.

Los parámetros de configuración consisten en la dirección de los distintos componentes a los que se conecta la aplicación y pueden modificarse mediante la actividad *Settings*, por lo que serán descritos en detalle en la siguiente sección.

Antes de lanzar una actividad, se comprueba que se tiene acceso a todo lo necesario para llevarse a cabo, si no hay conexión a internet o alguno de los proxys que necesita no está disponible, la aplicación muestra un aviso y no lanza la actividad requerida por el usuario.

4.3.3. Actividad Settings

La finalidad de la actividad *Settings* es poder modificar los parámetros de configuración de la aplicación en tiempo de ejecución para no tener que recompilar la aplicación y guardar los cambios de manera permanente.

Como ya se ha comentado, estos parámetros de configuración son las direcciones de los proxys donde reciben peticiones los componentes que proporcionan los datos relevantes para el usuario. Básicamente se componen de un nombre identificativo, una dirección IP y un puerto especificados de la siguiente manera:

```
<nombre>: tcp -h <ip> -p <puerto>
```

El middleware ICE sólo necesita que le proporcionen una cadena de texto de este estilo para obtener el objeto remoto o *proxy* con el que trabajar. Por lo que esta actividad da la posibilidad de modificar estas cadenas de texto. Concretamente la aplicación necesita que le sean especificados seis proxys distintos: uno por cada cámara del sistema (en total son cuatro), uno para conectar con *RecordingManager* y otro con *Eldercare*.

Si el usuario introduce mal uno de los proxys, será avisado de ello desde la actividad principal, como se ha indicado antes.

El mecanismo utilizado para guardar los cambios de manera permanente es una utilidad que facilita Android llamada *Preference Manager*. A través de la interfaz *SharedPreferences.Editor* se pueden almacenar cualquier par clave-valor de manera que a pesar de abandonar la aplicación estos datos queden guardados. Las siguientes líneas muestran un pequeño ejemplo del código que materializa esto, en primer lugar para almacenar los datos y después para consultarlos:

```
SharedPreferences prefs =
    PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor spe = prefs.edit();
spe.putString(PROXY_A, string_proxyA);
...
spe.commit();

String camAproxy = prefs.getString(PROXY_A,
  this.getResources().getString(R.string.uri_camA));
```

De esta simple manera la aplicación mejora su usabilidad y flexibilidad.

4.3.4. Actividad Streaming

El *streaming* consiste en hacer una distribución de vídeo y/o audio. Esta distribución puede ser de 1 a 1, o de 1 a muchos. La palabra *streaming* se refiere a emitir un flujo de vídeo y/o audio en continuo, no se puede ir hacia atrás, ni hacia delante. Sólo es posible visualizar lo que se emite en ese momento. Es muy útil para realizar conferencias o visualizar contenido multimedia sin tener que descargar el archivo completo.

La actividad *Streaming* tiene una finalidad muy concreta, que es la de visualizar vídeo servido por otros componentes del sistema mediante streaming.

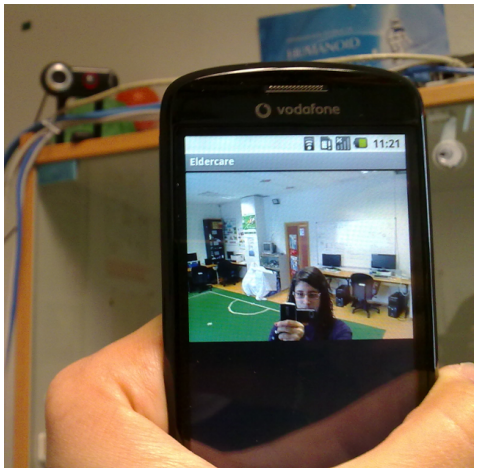

Figura 4.13: Actividad Streaming.

Para materializar esta funcionalidad se ha utilizado la clase *VideoView* de Android, que permite cargar imágenes desde diferentes fuentes. En este caso, la fuente es la uri proporcionada. Además, esta clase permite asociarle un *MediaController*, que ofrece las posibilidades de rebobinar, pausar e ir hacia delante en el vídeo.

Esta actividad es lanzada en dos ocasiones: desde la actividad *Main* para ver las imágenes de una cámara en directo y desde la actividad *ViewRecordings*, que se describe en la siguiente sección. En el caso de que se desee ver una cámara la actividad *Main* se conecta con el componente *CameraServer*, del que recibe la uri necesaria y a continuación invocar esta actividad.

4.3.5. Actividad ViewRecordings

En la sección 4.2 se describió la funcionalidad del componente *Eldercare*. La funcionalidad concreta que tiene que ver con esta actividad es la grabación de vídeo tras detectar una situación de peligro. Como se explicó, estas grabaciones quedan registradas en una base de datos. A estos datos la actividad *ViewRecordings* tiene acceso mediante el componente *RecordingManager*, al que realiza peticiones y éste devuelve la lista de grabaciones registradas hasta el momento.

Para obtener la lista de grabaciones consulta la configuración del *proxy* de dicho componente y crea el objeto *proxy*. Una vez creado, a través de él puede invocar los métodos definidos en la interfaz slice *Recording* implementada por *RecordingManager*. Concretamente el método invocado es *getRecordings()*.

Como esta actividad necesita mostrar la lista de grabaciones, hereda directamente de la actividad *ListActivity* que proporciona Android. Además de poder ver toda la lista de grabaciones, también es posible seleccionar aquellas correspondientes a un día especificado. Pulsando la opción correspondiente del menú de la actividad, aparece un cuadro de diálogo para indicar la fecha y tras introducirla la actividad invoca el método *getRecordingsByDate(date)* que devuelve únicamente las grabaciones realizadas ese día.

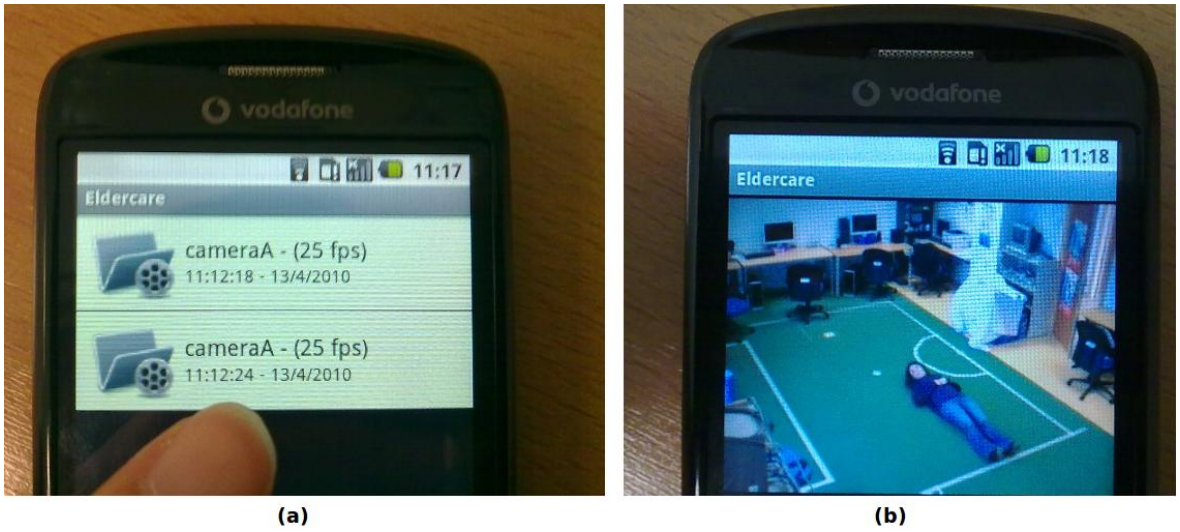

Figura 4.14: Actividad ViewRecordings - (a) Lista de grabaciones, (b) Vídeo.

Para ver una grabación se necesita pulsar sobre el elemento de la lista deseado y esto invoca el método *startStreaming()* de *RecordingManager* para que comience a servir el vídeo seleccionado mediante streaming y devuelva la uri donde se sirve. Para ver la grabación, esta actividad lanza la actividad *Streaming*, detallada en la sección anterior, indicándole la uri.

4.3.6. Actividad VirtualCamera

Como su nombre indica, esta actividad consiste en visualizar la escena monitorizada desde una cámara virtual. Esta cámara es la misma que la del componente *Eldercare*, utilizada para ver los resultados del seguimiento 3D de las personas dentro del laboratorio de Robótica.

Para implementarla se ha utilizado la librería OpenGL ES (ver sección 3.5.1), que

proporciona a esta aplicación la capacidad de ver en 3D lo que está ocurriendo en la habitación monitorizada. Esta librería proporciona la clase *GLSurfaceView* a la que se le debe asociar una clase que haga la función de *render*. Esta clase es *Renderer*, que hereda de *GLSurfaceView.Renderer* e implementa el método *onDrawFrame()*. En este método se dibujan los elementos que van a aparecer en la escena, concretamente el laboratorio y las trayectorias de las personas. El laboratorio se ha definido en la clase *Room* y la clase *Trajectory* representa una trayectoria. Ambos implementan el método *draw()* que es invocado desde la clase *Renderer* para dibujar estos elementos en la cámara virtual.

Figura 4.15: Actividad VirtualCamera.

Como se describió en la sección 4.2, el componente *Eldercare* sirve los resultados de su seguimiento visual de personas a través de la interfaz slice *EldercareProvider*. Esto permite que desde esta actividad se pueda recibir el estado del seguimiento en cada instante y construir las trayectorias asociadas a cada persona. Por tanto, es esta misma actividad la que guarda la estela de posiciones de las distintas personas.

4.3.7. Interfaz gráfica

La interfaz gráfica en este tipo de aplicación es un elemento necesario ya que es la única manera que tiene el usuario para interactuar con la aplicación. En la imagen 4.16 podemos ver la interfaz de la actividad *Main*.

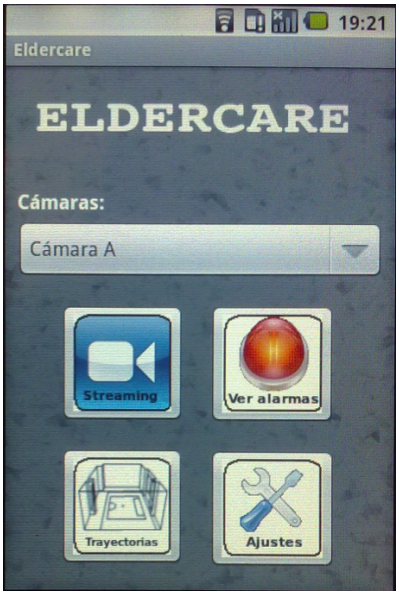

Figura 4.16: Interfaz de la actividad Main.

La interfaz gráfica se compone de diferentes escenarios o *layouts* descritos en formato XML. Normalmente cada actividad necesita su propio *layout* y en él se pueden introducir etiquetas de texto, botones, *checkbox*, menús desplegables, etc. Todo ello se puede configurar de manera muy flexible. En las siguientes líneas vemos una parte del XML que define el *layout* de la actividad *Main*, concretamente la parte del menú desplegable para seleccionar la cámara deseada:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   android:id="@+id/widget28"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical"
   android:background="@drawable/fondo"
   xmlns:android="http://schemas.android.com/apk/res/android"
   >
   <RelativeLayout
         android:id="@+id/layout01"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:paddingTop="100px"
         android:paddingLeft="10dip">
         <TextView
                  android:id="@+id/tvCamera"
                  android:text="@string/main_cameras"
                  android:layout_width="fill_parent"
                  android:layout_height="28dip"
```

```
                android:singleLine="true"
                android:textStyle="bold"
                android:textColor="#ffffff"
                android:textSize="16sp">
        </TextView>
        <Spinner android:id="@+id/spinner_camera"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_below = "@+id/tvCamera"
                android:drawSelectorOnTop="true">
        </Spinner>
    </RelativeLayout>
...
</RelativeLayout>
```

En general se ha utilizado el tipo de escenario *RelativeLayout*, ya que permite ubicar los elementos que contiene en las posiciones deseadas de forma relativa al principio del *layout* o relativa a otros elementos, como en el caso del *spinner* en este ejemplo a través de la propiedad *android:layout_below*.

Otro aspecto importante es el idioma en el que se muestra la interfaz. Android permite definir las cadenas de texto de una aplicación en distintos idiomas y, según la configuración de las locales del terminal que abra la aplicación, aparecerá en un idioma o en otro. Para esta aplicación se han utilizado dos idiomas: inglés y español.

Capítulo 5
Experimentos

Los experimentos, ajustes y observaciones realizados durante todo el proceso de creación de este proyecto se presentan en este capítulo. La misión principal de las pruebas fue validar el buen funcionamiento del sistema final. Además, una buena cantidad de experimentos fueron necesarios para ajustar el sistema así como para comparar las distintas implementaciones alternativas realizadas.

5.1. Escenarios de pruebas

Para validar el sistema se han realizado diferentes montajes, variando tanto el hardware como el lugar de pruebas. El escenario real en el que se han realizado la gran mayoría de experimentos ha sido el laboratorio de Robótica del Edificio Departamental II de esta universidad, que es una habitación de gran volumen (unos 40 metros cúbicos) con una ventana grande. El segundo lugar en el que se probó el sistema fue el salón de una casa. En las siguientes secciones se detallan estos montajes.

En cuanto al hardware, las características del equipo donde ejecuta *Eldercare* se han mantenido: Pentium Quad-Core a 2.4 Ghz cada núcleo, 4 GB de RAM y sistema operativo GNU/Linux, concretamente la distribución Ubuntu. Por otra parte, la aplicación cliente del sistema, *EldercareClient*, se ha probado en un móvil Android (versión 1.6) HTC Magic con procesador Qualcomm MSM7200A 528 MHz y memoria RAM: 288 MB.

5.1.1. Escenario I

El primer montaje se realizó con cuatro cámaras web *Apple iSight* (figura 5.2 (a)) a resolución 320x240 adheridas en el techo de las esquinas de la habitación.

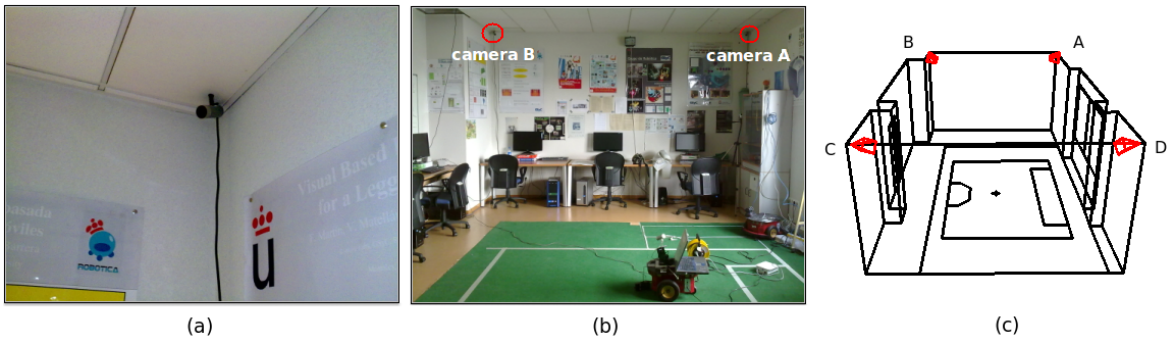

Figura 5.1: Laboratorio de Robótica.

Cada cámara está conectada a un equipo y sus imágenes son servidas a través de *CameraServer* por la red. El equipo principal es donde se reciben esas imágenes y se ejecuta la aplicación. En el mismo equipo o en otros diferentes se ejecutan los componentes *RecordingManager* y *Recorder*.

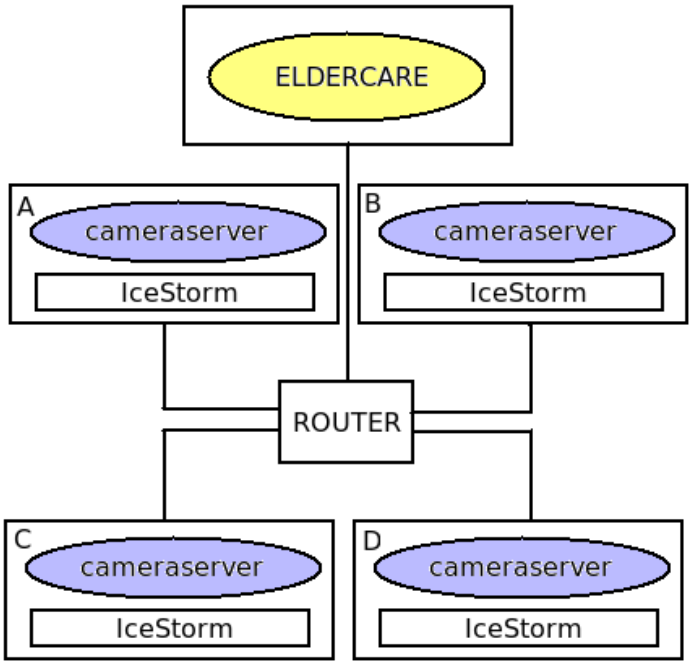

Figura 5.2: Primer montaje del sistema en el laboratorio - procesadores y componentes.

5.1.2. Escenario II

Para llevar a cabo las pruebas de robustez se realizó otro montaje con *cámaras IP Axis 207MW* (imagen 5.3 (a)) en el mismo laboratorio de Robótica. Se ubicaron en lugares distintos a las cámaras originales y a alturas y distancias distintas entre

ellas (imagen 5.3 (b)). El objetivo de probar otro montaje distinto es comprobar que el sistema continúa funcionando correctamente, es decir, que es robusto a cambios en el hardware.

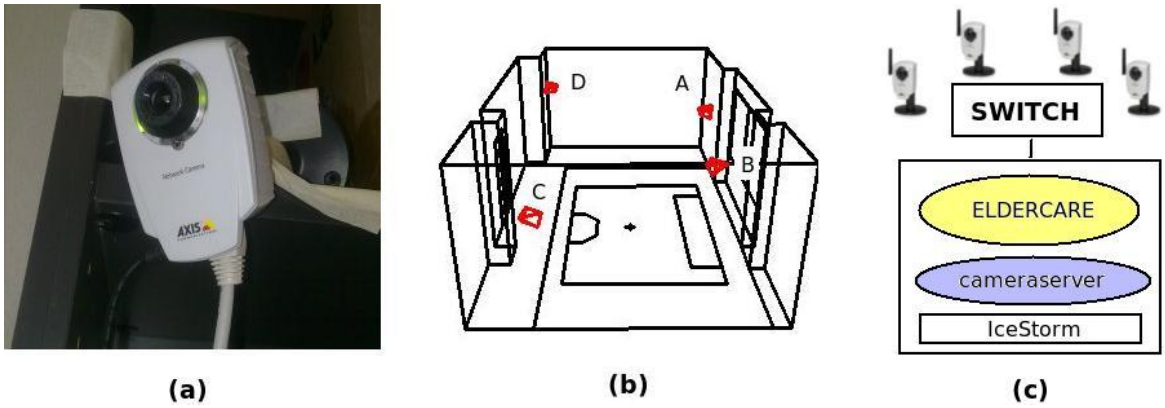

Figura 5.3: Segundo montaje del sistema en el laboratorio.

Mediante cables *ethernet* se conectó cada cámara a un *switch* y éste al equipo donde se ejecuta el sistema, formando una red privada (imagen 5.3 (c)). A cada cámara se le asignó una IP privada distinta. En el equipo se lanza tanto el componente *CameraServer*, que accede a las imágenes de las cámaras, como la aplicación *Eldercare*.

Con este nuevo montaje descubrimos que las imágenes llegaban con un retardo de un segundo aproximadamente. Posiblemente esto se debe a que el protocolo utilizado por *CameraServer* para obtener las imágenes es *http* y que algunos de los cables utilizados superan los 5 metros de longitud. Sin embargo, esto no entorpece el funcionamiento del sistema. Sí sería dañino diferentes retardos entre las cámaras.

Las cámaras se han configurado de tal modo que ofrezcan las imágenes tal como las reciben, sin ajustar el brillo, el contraste o el balance de blancos. Por este motivo las cámaras IP muestran mayor sensibilidad a los cambios de luz. Esto provocó que se tuviera que ajustar mejor el filtrado por movimiento de las imágenes, lo que se detalla en la sección 5.5.1.

5.1.3. Escenario III

En este escenario se utilizó el mismo hardware que en el escenario II y el montaje se realizó en la casa de un señor mayor que se prestó voluntario. La instalación se llevó a

cabo en el salón de la casa, ya que es la estancia en la que el señor pasa más tiempo (ver figura 5.4).

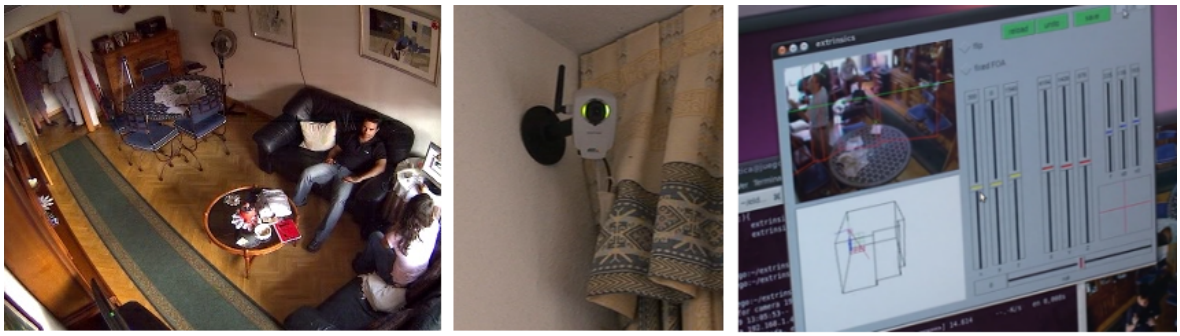

Figura 5.4: Montaje en el salón de una casa - (a) Imagen de una cámara, (b) Cámara Axis, (c) Calibrador.

Tras instalar las cámaras, la primera tarea consistió en medir la habitación para generar un modelo 3D de la misma. Una vez obtenido el modelo, se pudieron calibrar las cámaras con el componente *Calibrator* (imagen 5.4 (c)).

En este montaje se han utilizado cinco cámaras y esto hace que el rendimiento temporal descienda levemente, sin embargo, como se vio en [Marugán, 2010], a medida que aumenta el número de cámaras utilizadas el error de estimación del sistema disminuye y por tanto compensa su utilización.

5.1.4. Escenario IV

El componente *CameraServer* también es capaz de servir imágenes desde fichero de vídeo. Esto permite utilizar una configuración del sistema complementaria a las anteriores y que consiste en ejecutar el sistema sobre imágenes de vídeo previamente grabado. Esta configuración ha permitido agilizar los experimentos y ajustes del sistema durante el desarrollo del mismo. Otra utilidad de esta configuración es la de probar distintas implementaciones del algoritmo sobre exactamente las mismas imágenes de entrada, lo que facilita compararlas.

Además, gracias a esto se ha podido utilizar una base de datos de caídas para el cálculo de la tasa de aciertos, presentado en la sección 5.3. La base de datos se compone de varios vídeos sobre los que se ejecutó el sistema.

5.2. Ejecución típica

El sistema desarrollado se ha validado experimentalmente a través de múltiples pruebas en los dos montajes reales. En esta sección se presenta una ejecución representativa del sistema, a la que llamamos ejecución típica.

La ejecución típica del sistema muestra el funcionamiento habitual del sistema, que cumple con los objetivos planteados al comienzo de este trabajo. En primer lugar se arrancan los componentes *RecordingManager* y *Recorder* a los que el componente *Eldercare* se conectará después. También se lanzan los componentes *CameraServer* para servir las imágenes de cada una de las cámaras. Por último, se ejecuta el componente *Eldercare*, que comienza a monitorizar la habitación.

Al entrar una persona en la habitación, el sistema detecta el movimiento y comienza su seguimiento. Durante su seguimiento, el sistema aprende la apariencia cromática de la persona, es decir, el color característico de su vestimenta. Esto ayuda a su seguimiento. En todo momento, la aplicación está comprobando que la posición de la persona no está muy cerca del suelo.

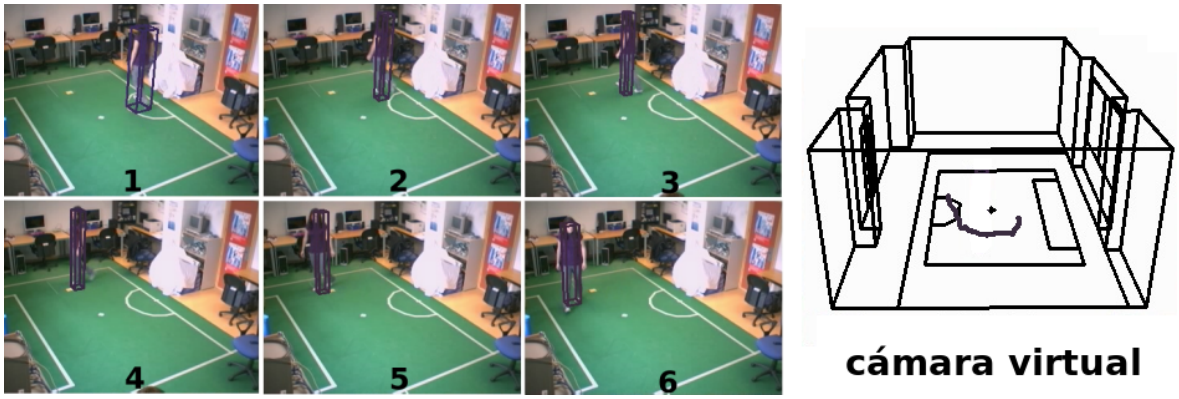

Figura 5.5: Eldercare siguiendo a una persona. Se muestran fotogramas de la cámara A y la trayectoria en la cámara virtual.

En el caso de que se produzca una caída por parte de la persona, el sistema pone en marcha distintos mecanismos de alarma, entre ellos una alarma visual en la interfaz, una alarma auditiva, envía un email a la cuenta de correo configurada y ordena grabar vídeo según las opciones también previamente configuradas.

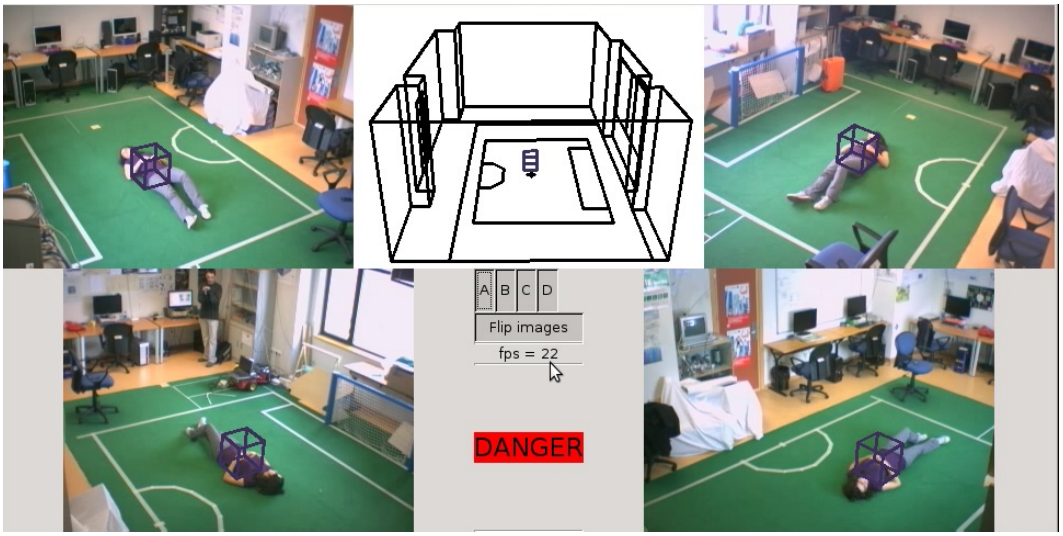

Figura 5.6: Eldercare detectando una caída (en escenario I).

Como se aprecia en la imagen, el sistema detecta correctamente la caída de la persona. El email (imagen 5.7 (a)) llega pocos minutos después al teléfono móvil y el vídeo grabado (imagen 5.7 (b)) puede verse entrando en la aplicación *EldercareClient*. Desde esta misma aplicación también se puede visualizar en tiempo real lo que está ocurriendo ya que *CameraServer* permite servir en *streaming* las imágenes de una cámara. Por último, si se quiere visualizar en 3D la trayectoria seguida por la persona, *EldercareClient* lo permite.

Figura 5.7: Alarma - (a) Email de alarma, (b) Visualizando el vídeo grabado desde *EldercareClient*.

De este modo se comprobó que todo el sistema integrado funcionaba correctamente para detectar la caída de una persona. Las siguientes pruebas incluían dos y tres

personas dentro de la habitación y pudimos ver que la aplicación continuaba detectando las caídas.

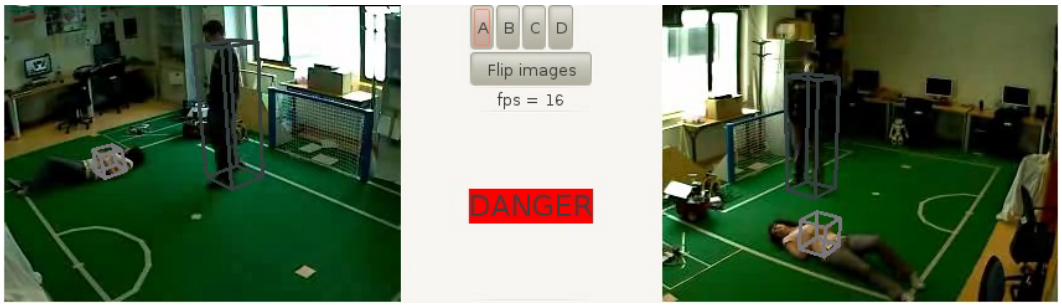

Figura 5.8: Eldercare siguiendo a dos personas y avisando de la caída de una de ellas (en escenario II).

Las últimas pruebas se realizaron en *la casa de una persona mayor* con un montaje de cinco cámaras. Las pruebas resultaron sastisfactorias ya que vimos que el sistema funcionaba igual que en el laboratorio. En la figura 5.9 vemos las imágenes de las cinco cámaras empleadas y cómo el sistema detecta que la persona mayor está caída en el suelo.

Figura 5.9: Prueba del sistema en la casa de una persona mayor (en escenario III).

Por tanto, las pruebas de validación corroboraron que el sistema funciona

correctamente en tiempo real. Los vídeos de estas pruebas pueden verse en el wiki[1] del trabajo fin de máster.

5.3. Porcentaje de caídas detectadas

Con la finalidad de caracterizar lo mejor posible el sistema se recopiló una "base de datos" de caídas para poder analizar el comportamiento del sistema utilizando ese banco de pruebas. El objetivo era hallar el porcentaje de caídas detectadas correctamente. En primer lugar vamos a definir los términos que se van a utilizar para indicar los resultados de las pruebas:

- *Éxito*: se refiere a que el sistema ha detectado correctamente una caída.
- *Falso negativo*: se refiere a que el sistema no ha detectado una caída.
- *Falso positivo*: se refiere a que el sistema ha detectado una caída que no se ha producido.

La base de datos contiene 5 vídeos utilizando las cuatro cámaras del sistema .en los que diferentes personas entran en el laboratorio de Robótica y se dejan caer al suelo. Las caídas se producen en distintas zonas del laboratorio y con diferentes condiciones de iluminación. Las personas siempre entran de uno en uno y salen de la habitación, por lo que cada caída se trata individualmente y sin relacionarse con las demás. El número total de caídas es 20. La siguiente figura muestra fotogramas de los diferentes vídeos.

Figura 5.10: Fotogramas de la base de datos de caídas.

[1] http://jde.gsyc.es/index.php/Salons-tfm

Como el algoritmo de seguimiento tiene un factor aleatorio en su funcionamiento, se decidió ejecutar el sistema 10 veces diferentes sobre el banco de pruebas presentado.

Nº de ejecución	Nº éxitos	Nº falsos negativos	Nº falsos positivos
1	17	3	0
2	16	4	1
3	19	1	2
4	18	2	0
5	16	4	0
6	18	2	1
7	18	2	0
8	18	2	0
9	18	2	0
10	17	3	1

Sobre estos resultados se hallaron la media y la moda de los éxitos, falsos negativos y falsos positivos obtenidos. En la siguiente tabla se muestran estos números:

	% Éxitos	% Falsos negativos	% Falsos positivos
Media	87.5	12.5	2.5
Moda	90	10	0

Los resultados obtenidos muestran que en la gran mayoría de los casos el sistema detecta las caídas correctamente y que el número de falsos positivos es mínimo.

Análisis de errores

Durante las distintas ejecuciones sobre el banco de pruebas, los éxitos normalmente se han producido en las mismas caídas, esto es, que si en una caída durante una de las ejecuciones el sistema ha tenido éxito, normalmente ha tenido éxito también en las demás. En el caso de los falsos negativos ha ocurrido lo mismo. Esto nos indica que cuando el sistema da un falso negativo tiene mucho que ver con las condiciones en las que sucede la caída de la persona. En estas condiciones sobre todo ha influido el número de cámaras que cubría la zona donde se ha caído la persona, siendo más propensas al fallo aquellas que sólo eran cubiertas por dos cámaras. Experimentos realizados en [Marugán, 2010] indicaron que el error del algoritmo en la estimación de la posición utilizando sólo dos cámaras podía llegar a los 20 centímetros.

Por otra parte, las pruebas han mostrado que ante distinto tipo de iluminación el funcionamiento del sistema es el mismo.

En cuanto a los falsos positivos, éstos suelen producirse cuando una persona sale de las escena y su seguimiento trata de redefinirse durante unos instantes hasta que es eliminado. En ese proceso puede que el seguimiento explore zonas cercanas al suelo y por ese motivo aparezca el falso positivo. Este experimento sobre caídas individuales nos ha mostrado que el porcentaje de falsos positivos es mínimo.

5.4. Rendimiento temporal

El rendimiento temporal es un factor importante en cualquier sistema software, pero en el caso de las aplicaciones en tiempo real es un factor clave. La aplicación desarrollada en este proyecto se encuentra dentro de ese tipo de sistemas, por lo que es imprescindible tener muy en cuenta esta medida y optimizar lo máximo posible los procesamientos. Por ejemplo, para convertir este sistema en un producto real de mercado, sería interesante que pudiera ser ejecutado en placas del tipo *Pico ITX* o *Mini ITX*, ya que se conseguiría un sistema de tiempo real ejecutando en un microprocesador barato.

En un principio el componente *Eldercare* se materializó con un único proceso que se encargaba de toda la funcionalidad, sin embargo pronto descubrimos que ejecutar ciertas tareas en hilos de ejecución separados del proceso principal podía mejorar el rendimiento del componente *Eldercare* y por tanto del sistema en general. Estas mejoras se describen en los apartados 5.4.2 y 5.4.3.

La medida utilizada para caracterizar el rendimiento temporal del sistema ha sido *iteraciones por segundo*. Esto es debido a que es un sistema que ejecuta en tiempo real y de manera iterativa.

Las pruebas indicaron que *la aplicación es capaz de ejecutar a 20 iteraciones por segundo* siguiendo a una persona.

5.4.1. Influencia del número de personas seguidas

En esta sección se analiza un factor importante que influye determinantemente sobre el rendimiento temporal del sistema. Este factor es el número de personas que es

capaz de seguir la aplicación.

En la sección 4.2.2 se explicó el algoritmo de seguimiento 3D visual de personas. Se detalló que a cada persona se le asocia una raza, que es un conjunto de prismas que se adaptan en cada iteración al movimiento y al tamaño de la persona para estimar su posición. Una raza necesita unos recursos asociados y que requieren ser actualizados en cada iteración, como por ejemplo la propia población de prismas o el filtro de color asociado a la raza por cada una de las cámaras. Además, el algoritmo comprueba repetidamente si las razas son similares comparándolas por su posición y color. Todo esto provoca que seguir a una nueva persona conlleve un coste computacional añadido.

En este experimento se tomaron datos del tiempo que tarda en ejecutar el algoritmo con distinto número de personas seguidas. Se obtuvieron cuatro tipos de tiempos: tiempo medio en ejecutar una iteración completa (TC), tiempo medio en realizar el filtrado por movimiento de las imágenes (TM), tiempo medio en ejecutar una iteración del algoritmo de seguimiento (TA) y por último las iteraciones por segundo (IPS) en media. Todos los tiempos se han medido en *milisegundos*.

Una iteración completa comprende la recepción de imágenes, su filtrado por movimiento, una iteración del algoritmo de seguimiento y la actualización de la interfaz gráfica.

Acerca del tiempo medio en realizar el filtrado por movimiento de las imágenes (TM), hay que aclarar que este proceso se ejecuta en paralelo al algoritmo de seguimiento, como se explicó en la sección anterior. Esto hace que cuanto mayor sea el tiempo que tarda una iteración del algoritmo, el tiempo del filtrado por movimiento acaba siendo despreciable puesto que al acabar el algoritmo ya está listo el resultado del filtrado de las imágenes.

Tarea	0 personas	1 persona	2 personas	3 personas
TC	45	50	65	94
TM	30	0	0	0
TA	10	38	58	87
IPS	23	20	17	10

La conclusión principal obtenida tras analizar los datos es que el sistema posee

un límite de personas seguidas. Valorando el buen funcionamiento del algoritmo este límite ronda las tres personas. El rendimiento temporal del sistema siguiendo a cuatro personas está por debajo de las 10 iteraciones por segundo, lo que hace empeorar notablemente el funcionamiento del algoritmo.

Cuando no hay personas en la escena, el sistema invierte la mayor parte del tiempo en el filtrado por movimiento de las imágenes. Siguiendo a una persona, el tiempo del algoritmo de seguimiento ya supera el tiempo del filtrado y por tanto éste se hace despreciable debido a que se ejecutan en paralelo. Las imágenes se reciben a 30 fotogramas por segundo y el sistema itera a 20, lo que hace que el seguimiento sea muy vivaz. Con dos y tres personas la aplicación emplea mucho más tiempo en el algoritmo de seguimiento y baja el rendimiento temporal pero no se pierde el tiempo real blando.

5.4.2. Optimización de la recepción de imágenes

Con el objetivo de seleccionar el modo de obtener las imágenes de *CameraServer* y cuántos hilos de ejecución eran necesarios para el mejor rendimiento temporal de la aplicación, se realizaron pruebas con distintas implementaciones.

En primer lugar se probó el modo cliente/servidor (C/S-1hilo) de *CameraServer* para recibir las imágenes, sin embargo se comprobó que la latencia de la llamada RPC para obtener una imagen era demasiada y más si se multiplica por cuatro cámaras que se utilizan. Por lo que se pensó en lanzar un hilo por cada cámara (C/S-5hilos) para obtener las imágenes. Esta alternativa de implementación mejoraba el rendimiento pero requería cinco procesos.

Tras estas pruebas se decidió probar el modo publicación/subscripción de *CameraServer*, implementado a propósito de este trabajo y utilizando el servicio IceStorm de ICE. Los resultados fueron muy favorables. Con un único thread (IceStorm-1hilo) incluso se mejoraba algo el rendimiento temporal que daba la implementación con cinco threads (IceStorm-5hilos). También se probó a recoger las imágenes con publicación/subscripción en un hilo separado (IceStorm-2hilos) y comprobamos que mantenía el rendimiento.

Aquí se muestran los números concretos de iteraciones por segundo a las que la

aplicación ejecutaba en cada caso. Los datos se obtuvieron durante el *seguimiento de dos personas, considerando esto el caso típico*. Además se probó en dos procesadores distintos para ver la influencia del número de núcleos.

Procesador	C/S - 1 hilo	C/S - 5 hilos	IceStorm - 1 hilo	IceStorm - 2 hilos
Core Duo	4	12	12	12
Quad Core	7	16	17	17

Como muestran los datos, utilizando el modo cliente/servidor, la diferencia entre ejecutar un único proceso o utilizar un proceso para cada cámara además del principal es muy significativa. La primera alternativa es simplemente inviable, mientras que la segunda sí lo es. Comparando este resultado con la utilización del modo publicación/subscripción, el rendimiento temporal es muy similar, por lo que la comparación pasa a ser sobre el uso de recursos. Con publicación/subscripción se necesitan menos recursos, lo que es un buen motivo para seleccionar este modo de obtener imágenes.

Con publicación/subscripción no hay diferencia entre usar un único proceso o utilizar además de éste un hilo separado que obtenga las imágenes, se escogió mantener en un hilo separado para la recepción de imágenes de *CameraServer* como la mejor opción.

5.4.3. Optimización del filtrado de movimiento

El filtrado de movimiento de las imágenes es un preproceso necesario para el algoritmo de seguimiento. Al igual que la tarea de recepción de imágenes, este filtrado puede hacerse en un hilo de ejecución separado del principal. Esto aporta la ventaja de ejecutar en paralelo el filtrado y el algoritmo de seguimiento. El resultado de filtrar las imágenes de determinada iteración se puede utilizar en la iteración siguiente para el algoritmo de seguimiento sin perder eficacia. Gracias a esto es posible la paralelización del filtrado de movimiento.

Para dar una idea cuantitativa de la mejora en el rendimiento temporal de la aplicación, se midió en milisegundos el tiempo que se tarda en filtrar las cuatro imágenes que recibe la aplicación en determinada iteración. En media el resultado fue 30 milisegundos. Al paralelizar este proceso, el algoritmo de seguimiento no tiene

que esperar ese tiempo de preprocesar las imágenes, sino que coge el resultado de la iteración anterior y mientras se ejecuta el algoritmo se filtran las imágenes nuevas. Esto *aumenta el rendimiento temporal del sistema en 3 iteraciones por segundo.*

En la siguiente sección se detalla mejor el tiempo invertido en cada procesamiento y se evalúa el rendimiento en distintos casos.

5.5. Pruebas de robustez

En este trabajo fin de máster se ha dado importancia no sólo a que el sistema integrando todos los componentes funcione, si no que la aplicación funcione en el mayor número casos posibles, que sea robusta. Las pruebas que se presentan en las siguientes secciones son experimentos encaminados a conocer y mejorar la robustez del sistema.

5.5.1. Robustez frente a cambios de iluminación

Una característica muy deseada para el sistema es que sea en gran medida robusto a los cambios de iluminación. El objetivo es acercarlo lo máximo posible a una aplicación real disponible en el mercado y superar la fase de prototipo en el laboratorio.

Haciendo pruebas con cámaras IP se observó que estas cámaras son más sensibles a los cambios de luz que las cámaras *iSight* que utilizábamos hasta ahora. Esto quiere decir que ante un cambio de luz en el ambiente o simplemente las sombras de las personas hacen que una imagen cambie bastante. Debido a esta situación, detectamos que el filtro de movimiento no era lo suficientemente robusto a estos cambios de iluminación. En la figura 5.11 podemos ver dos situaciones conflictivas que provocan que el filtro de movimiento empeore el seguimiento. La primera es la sombra de la persona y la segunda es al encender las luces del laboratorio. Los píxeles dibujados en blanco son los que superan el filtro de movimiento y las cajas rojas son las regiones que contienen el movimiento.

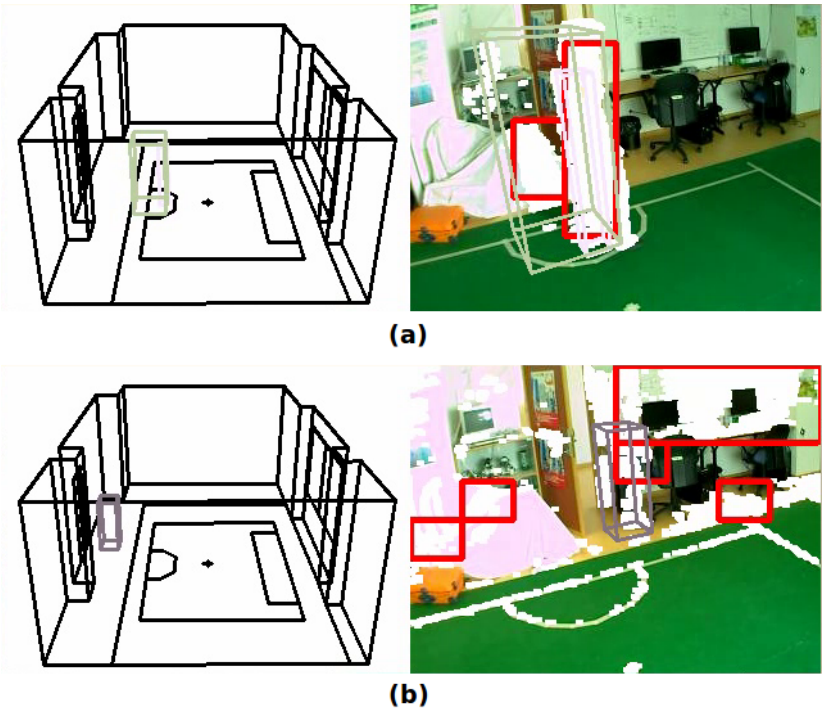

Figura 5.11: Filtro RGB de movimiento - (a) Sombras, (b) Cambio brusco de iluminación.

El filtro de movimiento se basaba en la diferencia de píxeles RGB respecto al fondo aprendido y la imagen previa en cada instante. En primer lugar se intentaron ajustar los umbrales de diferencia para solucionar el problema, sin embargo no se obtenía el resultado esperado. El siguiente paso fue probar a construir un filtro de movimiento basado en el modelo de color HSV (Hue Saturation Value). Este modelo permite distinguir muy bien el valor de iluminación asociado a un píxel, ya que la propia coordenada V lo indica.

Para generar el filtro de movimiento en HSV únicamente fue necesario convertir las imágenes al modelo de color HSV, restar píxel a píxel para obtener la diferencia y ajustar las tolerancias. En un primer momento se puso un umbral a la diferencia en cada canal y si era superada en los tres canales se consideraba movimiento. Este primer acercamiento dio resultados similares a los ofrecidos por el filtro RGB, por lo que se experimentó con varias ecuaciones para definir el filtro y las tolerancias hasta se encontró la que mejor resultado daba. Las siguientes imágenes muestran la mejora de robustez del filtro de movimiento frente a los cambios de iluminación.

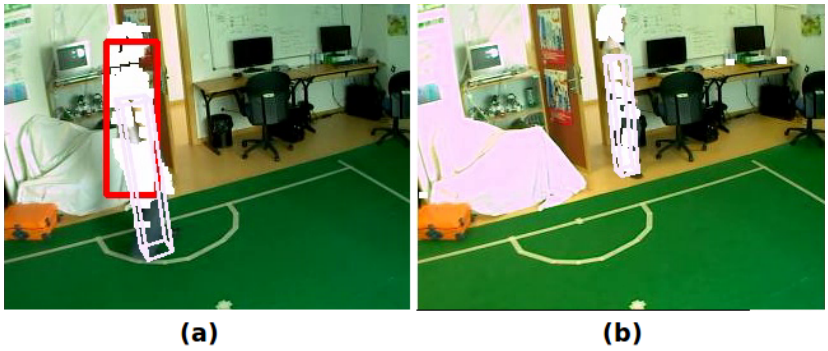

Figura 5.12: Filtro HSV de movimiento - (a) Sombras, (b) Cambio brusco de iluminación.

La fórmula aplicada para el cálculo de la diferencia entre dos píxeles p1(h,s,v) y p2(h,s,v) es:

$$D(p1, p2) = (diffH > Htol\ AND\ diffS > Stol\ AND\ diffV > Vtol)\ OR\ (diffV > Vtol_{max}) \tag{5.1}$$

con diffH, diffS y diffV las diferencias entre los distintos canales y las tolerancias del filtro Htol=10, Stol=58, Vtol=58 y $Vtol_{max}$=170. Para hallar la resta en el canal H se ha tenido en cuenta que los valores son ángulos.

La segunda parte de la fórmula, que únicamente considera la coordenada V, está pensada para aquellos píxeles cercanos al color blanco o negro. Esto es porque las coordenadas H y S de dichos píxeles no cobran importancia si la iluminación es muy baja o por el contrario muy alta. Por tanto, como puede haber o no diferencia en las coordenadas H y S para este tipo de píxeles, la diferencia para ellos se basa en la iluminación únicamente.

5.5.2. Robustez en la activación de la alarma

Un experimento muy simple y que nos indica el grado de ajuste del sistema para detectar caídas es diferenciar entre una persona sentada en el suelo y tumbada en él. Como vemos en las imágenes una persona sentada en el suelo no hace que el sistema active la alarma, sin embargo cuando se tumba en el suelo sí se dispara.

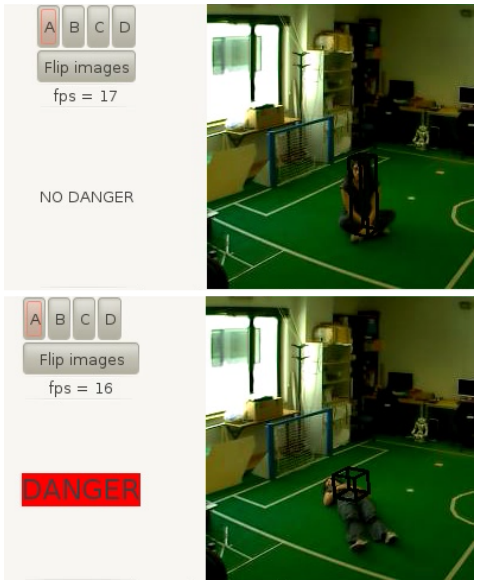

Figura 5.13: Distinción entre posición sentado y tumbado.

Esta es una prueba de la robustez del sistema en cuanto a la detección de la situación de peligro. Además, también se comprobó que al sentarse en las sillas el seguimiento continuaba correctamente y no se disparaba la alarma.

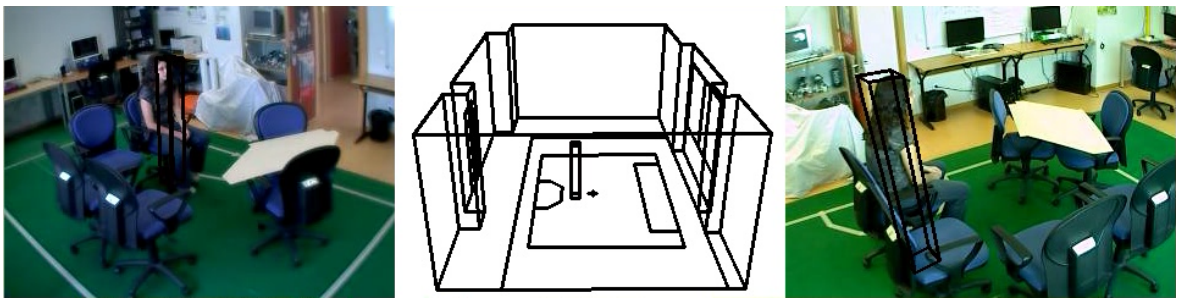

Figura 5.14: Seguimiento en la posición sentado.

Otro aspecto a analizar de la robustez del sistema a la hora de activar la alarma es la influencia del mobiliario en la sala monitorizada. Los muebles generan oclusiones y esto dificulta el seguimiento de las personas. En el propio laboratorio de Robótica se colocaron las sillas de tal modo que simularan un sofá y una mesa con el objetivo de ver qué ocurría cuando una persona se cae entre ellas. El resultado del experimento fue satisfactorio puesto que el sistema detecta las caídas a pesar de las oclusiones que provocaban las sillas (ver imagen 5.15).

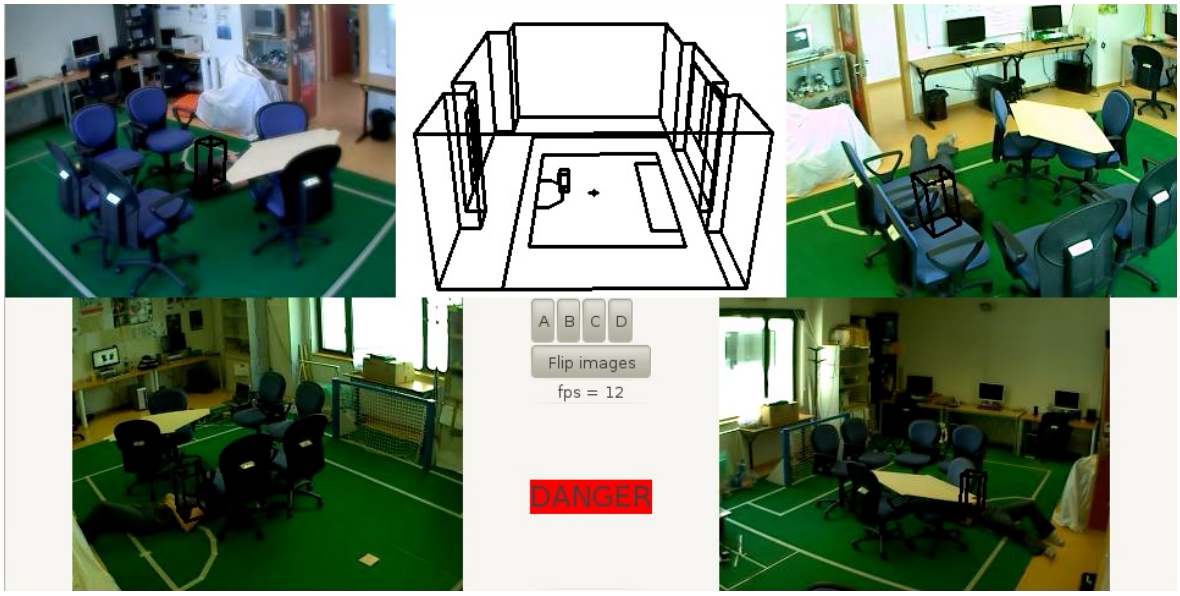

Figura 5.15: Detección de una caída con oclusión por el mobiliario (en escenario II).

Al llevar el sistema a un salón real de una casa también pudimos comprobar que el sistema funcionaba correctamente a pesar de las oclusiones del mobiliario (imagen 5.16).

Figura 5.16: Detección de una caída con oclusión por el mobiliario (en escenario III).

Capítulo 6
Conclusiones y trabajos futuros

En los capítulos anteriores se presentó la nueva plataforma de desarrollo en la que se apoya este trabajo, a continuación se describió el algoritmo y la tecnología utilizada en el sistema de detección de caídas y por último se expusieron los experimentos realizados para comprobar el correcto funcionamiento y la robustez del sistema. En este capítulo se resumen las conclusiones obtenidas y las posibles vías de continuación para la investigación.

6.1. Conclusiones

El objetivo principal de este trabajo era portar el sistema de detección de caídas *Eldercare* a la nueva plataforma *Jderobot-5.0*. La finalidad era aprovechar su arquitectura distribuida de componentes e integrar en el sistema un cliente móvil que reciba alarmas además de otros tipos de información.

En primer lugar es necesario destacar que *los tres subobjetivos en los que se divide este proyecto se han alcanzado con éxito*. El primero consistía en migrar la aplicación *Eldercare* a la plataforma *Jderobot-5.0*, agregando la funcionalidad necesaria tanto a la plataforma como a la propia aplicación. El segundo se trataba de crear una aplicación cliente que ejecutara en un teléfono móvil y que pudiera acceder a la información generada por la aplicación principal, producto del primer subobjetivo. El último consistía en realizar pruebas con todos los componentes integrados y experimentos para caracterizar la robustez del sistema.

En el capítulo 3 se presentaron los componentes de la plataforma *Jderobot-5.0* utilizados en el sistema. Como parte del primer subobjetivo, a algunos componentes fue necesario añadirles nueva funcionalidad, concretamente a *CameraServer* y a *Recorder*. A *CameraServer* se le dotó de la capacidad para transmitir imágenes mediante

publicación/suscripción, utilizando el servicio *IceStorm* de ICE, y además la capacidad de acceder a las imágenes de cámaras firewire. En cuanto al componente *Recorder*, se le agregó un nuevo tipo de grabador, que permite grabar vídeo no sólo de un dispositivo local a la máquina donde ejecute sino de cualquier *CameraServer* al que se conecte. Esto también resuelve el problema de conectar varios *Recorder* a la misma fuente de imágenes.
Por último, también fue parte de este trabajo generar el componente *Calibrator* para la calibración de las cámaras del sistema. Esta funcionalidad ya se tenía en la versión anterior de la plataforma, por lo que la tarea consistió en refactorizar el código fuente al paradigma orientado a objetos y crear el nuevo componente que proporcionara esa funcionalidad.

El componente *Eldercare*, que es el principal del sistema, fue creado para ejecutar el algoritmo de seguimiento visual de personas y detección de caídas. Se diseñó mediante el Modelo Vista Controlador (MVC). El modelo alberga todo lo relacionado con el algoritmo, la vista se encarga de los eventos que se producen en la interfaz gráfica y el controlador conecta a los dos anteriores. Una vez refactorizado el código fuente del algoritmo se añadió la capacidad de generar alarmas utilizando la funcionalidad ofrecida por los componentes de *Jderobot-5.0*. Esta nueva funcionalidad comprende el nuevo mecanismo de alarma, la capacidad de servir el estado del seguimiento a otros componentes a través de un interfaz slice y la mejora del filtro de movimiento.
En cuanto al mecanismo de alarma, se mantuvo la alarma auditiva y visual en la interfaz gráfica, pero además se programó el envío de un email de aviso a una o más cuentas de correo electrónico configurables. De este modo la alarma puede transmitirse a las personas que estén interesadas. Además, tras detectar una caída se ordena grabar vídeo a través del componente *RecordingManager*. La información acerca de estos vídeos queda almacenada en una *base de datos*, por lo que puede ser recuperada posteriormente.

El segundo subobjetivo consistía en generar una aplicación cliente que se integrara en el sistema a través de la plataforma *Jderobot-5.0* y que ejecutara en un teléfono móvil con sistema operativo Android. Esta aplicación debía dar acceso a distintos tipos de información, que son: los vídeos grabados al activarse la alarma, el estado del seguimiento de las personas que se encuentren en la zona monitorizada y las imágenes en tiempo real de las cámaras del sistema.
Esta aplicación para móvil, a la que hemos llamado *EldercareClient*, se dividió en

distintas actividades. Una de ellas se conecta al componente *RecordingManager* para obtener la lista de grabaciones, que pueden consultarse por fecha y pueden visualizarse desde el propio móvil para ver lo ocurrido. Otra se conecta al componente *Eldercare* y consiste en un visor OpenGL de la cámara virtual del sistema, en la que se dibujan en tiempo real las trayectorias de las personas seguidas según se van recibiendo las posiciones 3D de las mismas. Por último, otra actividad envía una petición al componente *CameraServer* para que comience a servir las imágenes de una cámara a través de *streaming* y poderlas ver en el móvil.
Por tanto, desde esta aplicación se tiene acceso a conocer todo lo que está ocurriendo en la zona monitorizada en cualquier momento, sin embargo es especialmente interesante tras recibir un email de aviso del peligro en el propio teléfono.

Evaluar el sistema con todos los componentes integrados era la finalidad del tercer subobjetivo. En primer lugar el sistema se probó con el montaje utilizado en los anteriores proyectos (ver sección 5.1.1), con el que se pudo comprobar su correcto funcionamiento. Además, se realizaron distintas implementaciones del componente *Eldercare* para averiguar cuál de ellas era la mejor en cuanto al rendimiento temporal, que es de gran importancia para el sistema.
Una vez ajustado el funcionamiento del sistema, se quiso dar un paso más y caracterizar lo mejor posible su robustez realizando distintos tipos de experimentos. El primer paso fue montar otro entorno de pruebas en el mismo laboratorio. Esta vez con cámaras IP conectadas mediante un switch al equipo principal (ver sección 5.1.2). Estas cámaras eran más sensibles a los cambios de luz y esto afectaba al filtrado de movimiento de las imágenes, por lo que se tuvo que mejorar utilizando el modelo HSV de color para solventar el problema. Otros experimentos consistieron en valorar el comportamiento del sistema frente a cambios en el mobiliario de la sala y las diferentes posiciones de las personas tales como sentado en una silla, sentado en el suelo o tumbado.
Para terminar de caracterizar el sistema se quiso averiguar el porcentaje de caídas detectadas. Para ello se recopilaron caídas de personas mediante varios vídeos en diferentes zonas y condiciones de luz, obteniendo un banco de pruebas. El sistema se ejecutó 10 veces distintas sobre el banco de pruebas de un total de 20 caídas. Los resultados fueron satisfactorios, puesto que *en un 90 % de los casos el sistema tuvo éxito*. Los casos de fallo estaban relacionados con que el número de cámaras que cubría la zona de la caída era mínimo, lo que es solucionable introduciendo más cámaras en el sistema.
Las últimas pruebas se realizaron en la casa de una persona mayor que se

prestó voluntaria. Para ello se instaló el sistema en el salón con un montaje de cinco cámaras IP (ver sección 5.1.3). Los resultados fueron satisfactorios, ya que a pesar del cambio de entorno el sistema funcionaba igual que en el laboratorio.

En cuanto a los requisitos, como se vio en los experimentos, *la aplicación cumple todos los planteados* en el capítulo 2. Utiliza únicamente *hardware convencional*, concretamente cámaras de videoconferencia y ordenadores personales. Funciona correctamente en una *habitación de gran volumen*, que como se ha visto en los experimentos es el Laboratorio de Robótica del Edificio Departamental II de esta universidad. Por último, es capaz de *seguir personas andando* y detectar cuando una de ellas se ha caído al suelo, avisando al usuario vía email y pudiendo observar en tiempo real lo ocurrido desde un teléfono móvil.

Para terminar, conviene recordar que *este trabajo fin de máster se enmarca dentro del proyecto Eldercare*, un sistema para la asistencia y cuidado de mayores en su lugar de residencia. Este trabajo *constituye la cuarta generación*, que ha portado el sistema a una nueva plataforma basada en componentes distribuidos. Esto ha facilitado la integración de un teléfono móvil en el sistema para recibir las alarmas generadas o simplemente ver la zona monitorizada en cualquier momento, lo que ha aumentado en gran medida la usabilidad del sistema.

6.2. Trabajos futuros

La primera línea futura de trabajo consiste en seguir probando el sistema de detección de caídas en entornos reales, como casas o residencias de ancianos. Esto permitiría valorar mejor la validez del sistema.

Una segunda línea de trabajo consistiría en dar el paso hacia monitorizar una casa entera, no sólo una habitación. Este paso obligaría a distribuir el procesamiento mucho más. Posiblemente la primera solución a probar sería replicar el sistema en cada habitación de la casa y conectar sus salidas a un equipo principal, encargado de analizar la información y activar la alarma en caso necesario.

Por último, con el objetivo de paliar la limitación del sistema cuando las luces están apagadas, se podría estudiar la inclusión de cámaras infrarrojas en el sistema.

Bibliografía

[Barrera *et al.*, 2005] Pablo Barrera, José María Cañas, y Vicente Matellán. Visual object tracking in 3d with color based particle filter. *Int. Journal of Information Technology*, 2005.

[Barrera y Cañas, 2004] Pablo Barrera y José María Cañas. Seguimiento tridimensional usando dos cámaras. 2004.

[Bravo, 2004] David Lobato Bravo. jde+: an object oriented implementation of jde. *Proyecto Fin de Carrera, URJC*, 2004.

[Bravo, 2010] David Lobato Bravo. jderobot 5.0 : Entorno de desarrollo basado en componentes para aplicaciones robóticas. Trabajo fin de máster, Universidad Rey Juan Carlos, 2010.

[Burnette, 2009] Ed Burnette. *Hello, Android. Introducing Google's Mobile Development Platform.* Ed. The Pragmatic Bookshelf, 1 edition, 2009. Libro en ingles.

[Cañas *et al.*, 2008] José María Cañas, Sara Marugán, Carlos Agüero, y Teodoro González. Detección visual de caídas para ambientes inteligentes. *Proceedings of RoboCity2030 4th Workshop, Robots personales y asistenciales*, 2008.

[Cañas *et al.*, 2009] José María Cañas, Sara Marugán, Marta Marrón, y Juan C. García. Visual fall detection for intelligent spaces. *Proceedings of the 6th IEEE International Symposium on Intelligent Signal Processing*, 2009.

[Cañas Plaza, 2003] Jose María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis Doctoral - Universidad Politécnica*, Diciembre 2003.

[D. Margaritis, 1998] S. Thrun D. Margaritis. Learning to locate an object in 3d space from a sequence of images. 1998.

[DeMenthon y Davis, 1995] D. DeMenthon y L. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 1995.

[Henning y Spruiell, 2010] Michi Henning y Mark Spruiell. Distributed programming with ice. Website, 2010. `http://www.zeroc.com/doc/Ice-3.4.1/manual`.

[Henning, 2004] M. Henning. A new approach to object-oriented middleware. *Internet Computing, IEEE*, 8(1):66–75, 2004.

[J.Davison *et al.*, 2007] Andrew J.Davison, Ian D. Reid, Nicholas D. Molton, y Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[López, 2010] Luis Miguel López. Autolocalización en tiempo real mediante seguiminto visual monocular. *Proyecto Fin de Carrera, URJC*, 2010.

[Marugán, 2007] Sara Marugán. Seguimiento 3d visual de múltiples personas utilizando un algoritmo evolutivo multimodal. *Proyecto Fin de Carrera, URJC*, 2007.

[Marugán, 2010] Sara Marugán. Seguimiento visual de personas mediante evolución de primitivas volumétricas. *Proyecto Fin de Carrera, URJC*, 2010.

[Meier, 2010] Reto Meier. *Professional Android 2 Application Development.* Ed. Wrox, 2010. Libro en ingles.

[Palomino, 2010] Roberto Calvo Palomino. Sistema distribuido de vídeo-vigilancia basado en android. Trabajo fin de máster, Universidad Rey Juan Carlos, 2010.

[Pineda, 2006] Antonio Pineda. Aplicación de seguridad basada en visión. *Proyecto Fin de Carrera, URJC*, 2006.

[Team, 2010] Android Developer Team. The developer's guide. Website, 2010. `http://developer.android.com/guide/index.html`.