MÁSTER EN VISIÓN ARTIFICIAL

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2013-2014

Proyecto Fin de Máster

Algoritmo evolutivo para detección y seguimiento de personas en 3D con sensores RGB-D

Autor: Francisco Miguel Rivas Montero
Tutor: José María Cañas Plaza

Lo único que necesita el mal para triunfar en el mundo, es que los buenos no hagan nada

Edmundo Burke

Agradecimientos

Quisiera dar las gracias a todas las personas que me han apoyado durante toda mi trayectoria académica que desemboca en este trabajo fin de máster. A todos mis profesores y compañeros con los que tanto he aprendido.

Antes de nada me gustaría mencionar al tutor de este trabajo, José María Cañas Plaza. Me brindaste la posibilidad de trabajar contigo desde el primer momento con los Naos de nuestro trabajo fin de carrera y durante todos estos años me has enseñado infinidad de cosas. Gracias por tu buen hacer y por tu gran calidad tanto como profesor como persona.

Como no, agradecer al magnífico grupo de robótica de la Universidad Rey Juan Carlos, sobre todo a mi compañero Álex que tanto me ha ayudado durante el máster y a Roberto Calvo por su inestimable ayuda.

Además, quiero dar las gracias a mis compañeros del Departamento de Fisioterapia, Terapia Ocupacional, Rehabilitación y Medicina Física de la Universidad Rey Juan Carlos por todo vuestro apoyo y por la paciencia que habéis tenido conmigo durante la realización de este trabajo. En especial quiero destacar a Paco, Alicia, María, Esther, Lola y Javi, que habéis sufrido la fase experimental del proyecto, dando vueltas sin parar por el laboratorio para comprobar que todo funcionaba correctamente. Mil gracias.

También quiero reconocer el apoyo de mi novia Noelia, eres mi motivación y contigo todo siempre es más fácil. Gracias por todo y por saber siempre estar ahí.

No me quiero olvidar de mi familia, en especial de mis tíos y de mis padres. Me habéis dado todo lo que tengo y espero compensaros parte de ese sacrificio con este trabajo fin de máster. Gracias por todo lo que hacéis y por estar siempre tan pendientes.

¡Gracias a todos!

Resumen

El auge del mundo de los videojuegos en los últimos años ha hecho que se inviertan muchos recursos en el desarrollo de nuevos interfaces con el fin de mejorar la experiencia del usuario. Uno de los dispositivos más revolucionarios de la nueva generación en este ámbito son los sensores RGB-D tipo Kinect. Debido a las grandes ventajas que ofrece este sensor, se ha venido utilizando no sólo para interactuar con consolas sino también en otros campos como la robótica, la visión artificial o incluso en el ámbito clínico orientado a la rehabilitación.

En este trabajo fin de máster, se presenta un sistema de teleasistencia basado en este tipo de sensores. El sistema desarrollado está basado en un algoritmo evolutivo que es capaz de detectar y seguir personas en 3D en interiores. Este algoritmo utiliza por una parte, individuos exploradores encargados de estimar zonas donde pueden haber personas, y por otra parte individuos explotadores que se encargan de realizar la búsqueda local, afinando la posición exacta de cada individuo. Este algoritmo está acompañado de un módulo de alarmas que, en función de las posiciones de personas estimadas por el algoritmo, es capaz de detectar una serie de situaciones de riesgo. En el caso de haber detectado alguna circunstancia que conlleve peligro, se pueden disparar varias acciones como emitir un sonido o mandar un correo electrónico. El algoritmo desarrollado es capaz de realizar el seguimiento de múltiples personas en tiempo real, utilizando únicamente la información tridimensional ofrecida por el sensor. De esta forma, no necesita datos de color, lo que permite que el sistema funcione de manera ininterrumpida las 24 horas del día, tanto con luz como en total oscuridad.

Complementariamente a la descripción de las aplicaciones y algoritmos desarrollados se incluyen una serie de resultados obtenidos tras evaluar el sistema en dos entornos diferentes, un salón de un domicilio particular y un dormitorio de una residencia de ancianos.

Para el desarrollo del sistema se ha utilizado el lenguaje de programación C++ bajo la plataforma *JdeRobot* 5.2 en Debian Wheezy de 64 bits. El uso de *JdeRobot* permite que los componentes desarrollados para este trabajo puedan ejecutarse de manera distribuida, facilitando el acceso remoto a los datos del entorno, utilizando prácticamente cualquier dispositivo desde el exterior.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

En este primer capítulo se explicará, de forma general, el contexto en el que se encuadra este proyecto. Se desarrollarán los conceptos básicos de teleasistencia y visión artificial, las dificultades a la hora de desarrollar un sistema de seguimiento tridimensional de personas y se describirán los sensores RGB-D sobre los que se ha basado este proyecto.

1.1. Teleasistencia

La teleasistencia es un servicio principalmente orientado a personas mayores o con algún tipo de discapacidad o afectación cognitiva, que ofrece al sujeto en cuestión, la posibilidad de pedir ayuda desde su propio hogar de forma rápida y semiautomática.

El sistema de teleasistencia más empleado en la actualidad es el basado en pulsador. Este sistema consta de un teléfono fijo colocado en la casa del sujeto susceptible de necesitar algún tipo de asistencia y un pulsador que debe llevar encima, ya sea en forma de pulsera o de colgante (figura 1.1).

Estos sistemas han tenido gran acogida debido a que, a pesar de no ser un sistema de prevención, es capaz de reducir significativamente los daños sufridos en caso de accidente, minimizando el tiempo que tarda la persona en recibir asistencia. Si una

Figura 1.1: Sistema de teleasistencia con pulsador

caída o desvanecimiento se debe a afectaciones graves, tales como un ataque al corazón o un ictus, se estima que por cada hora de retraso en producirse la intervención sanitaria se pierde alrededor de un 10 % de autonomía. Según datos de la Organización Mundial de la Salud (OMS), cada año se producen 37.7 millones de caídas cuya gravedad requiere de atención médica, siendo los mayores de 65 años los más propensos a este tipo de accidentes[1]. Es por ello, que este tipo de dispositivos de teleasistencia son un recurso incluido en Ley de Promoción de la Autonomía Personal y Atención a las Personas en Situación de Dependencia[2].

En los últimos años se ha tratado de complementar la teleasistencia convencional con sensores novedosos para incrementar la funcionalidad prestada. Una evolución de estos sistemas consiste en la incorporación en el propio pulsador de un acelerómetro [3]. Con este sensor adicional, el sistema es capaz de comprobar automáticamente la verticalidad de la persona que lo lleva. Determinando si ha sufrido una caída cuando se registran cambios bruscos en los datos del sensor.

El principal problema que tienen estos sistemas es la necesidad, por parte del sujeto monitorizado, de llevar puesto el pulsador para que funcione correctamente. Estos sistemas van orientados principalmente a personas mayores o con algún tipo de afectación cognitiva, como pueden ser sujetos que padecen de Alzheimer. Que estas personas recuerden ponerse el colgante o que simplemente quieran llevar un aparato todo el día encima es complicado, aun siendo muchas las ventajas que ofrece el sistema para su salud en caso de algún tipo de incidente. Por ello, la teleasistencia ha estado en el punto de mira de grupos de investigación y empresas, con el fin de desarrollar sistemas remotos que sean totalmente autónomos y que no requieran llevar puesto ningún tipo de prenda o de dispositivo especial.

Uno de los medios más habituales, es añadir aplicaciones basadas en visión para complementar los dispositivos ya existentes (pulsador y acelerómetros), con los que se consigue un 94 % de aciertos en detección de caídas [Zhang *et al.*, 2013]. Este sistema no sólo detecta situaciones de riesgo, sino que es capaz de predecir qué está haciendo el sujeto en todo momento mediante técnicas de clasificación. Una evolución más radical, es no utilizar elementos que tenga que llevar consigo el sujeto, utilizando información aportada únicamente por sensores externos. En este sentido, una empresa que ha apostado por esta idea y que ha presentado una solución comercial es *Edao* [4]. Este sistema, desarrollado en Francia, utiliza cámaras de visión infrarroja con el fin de detectar situaciones de riesgo para la salud de las personas monitorizadas. Estas detecciones se realizan siguiendo un proceso semiautomático. En un primer paso, mediante el análisis de las imágenes captadas por las cámaras infrarrojas, es capaz de estimar un estado de prealarma cuando ha detectado alguna situación peligrosa. Una vez detectada esta situación se avisa, de forma automática, a un centro de gestión

[1] http://www.who.int/mediacentre/factsheets/fs344/es/
[2] https://www.boe.es/buscar/act.php?id=BOE-A-2006-21990
[3] http://www.gruponeat.com/docs/Product_iATOM.pdf
[4] www.edao.com

donde una persona se encargará de rechazar la alarma en caso de producirse un falso positivo o de tomar las medidas necesarias para solventar la situación correspondiente. Para aceptar o rechazar una alerta automática, el operador se apoya en imágenes de color capturadas por el propio sistema.

A pesar de suponer una mejora de los sistemas convencionales de teleasistencia, siguen teniendo una serie de desventajas. La evolución presentada por Zhang mantiene el requerimiento de portar un acelerómetro y, aunque el sistema de Edao no tiene esta necesidad, tiene el inconveniente de utilizar cámaras de visión infrarrojas. Este tipo de cámaras funcionan mejor que las cámaras convencionales en condiciones de baja luminosidad, pero necesitan un mínimo de iluminación para trabajar de forma adecuada, por lo que, en condiciones de total oscuridad, no son capaces de captar información alguna.

1.2. Visión artificial

Se entiende como Visión artificial o visión computacional, aquel campo de la inteligencia artificial que trata de comprender una escena extrayendo características de imágenes. Nada más aparecer en 1960, muchos grupos de investigación se centraron en este nuevo área y por ello se crearon revistas científicas como *Pattern Recognition* en 1968 o *Computer Vision and Pattern Recognition* en 1977, así como asociaciones internacionales como la *International Association for Pattern Recognition* en 1978. Aunque ya en 1969, se había publicado un libro centrado en esta materia titulado *Picture Processing by Computer* [Rosenfeld, 1969], no fue hasta la década de los 90, gracias a la evolución de los ordenadores, cuando se empezaron a generalizar aplicaciones reales basadas en visión.

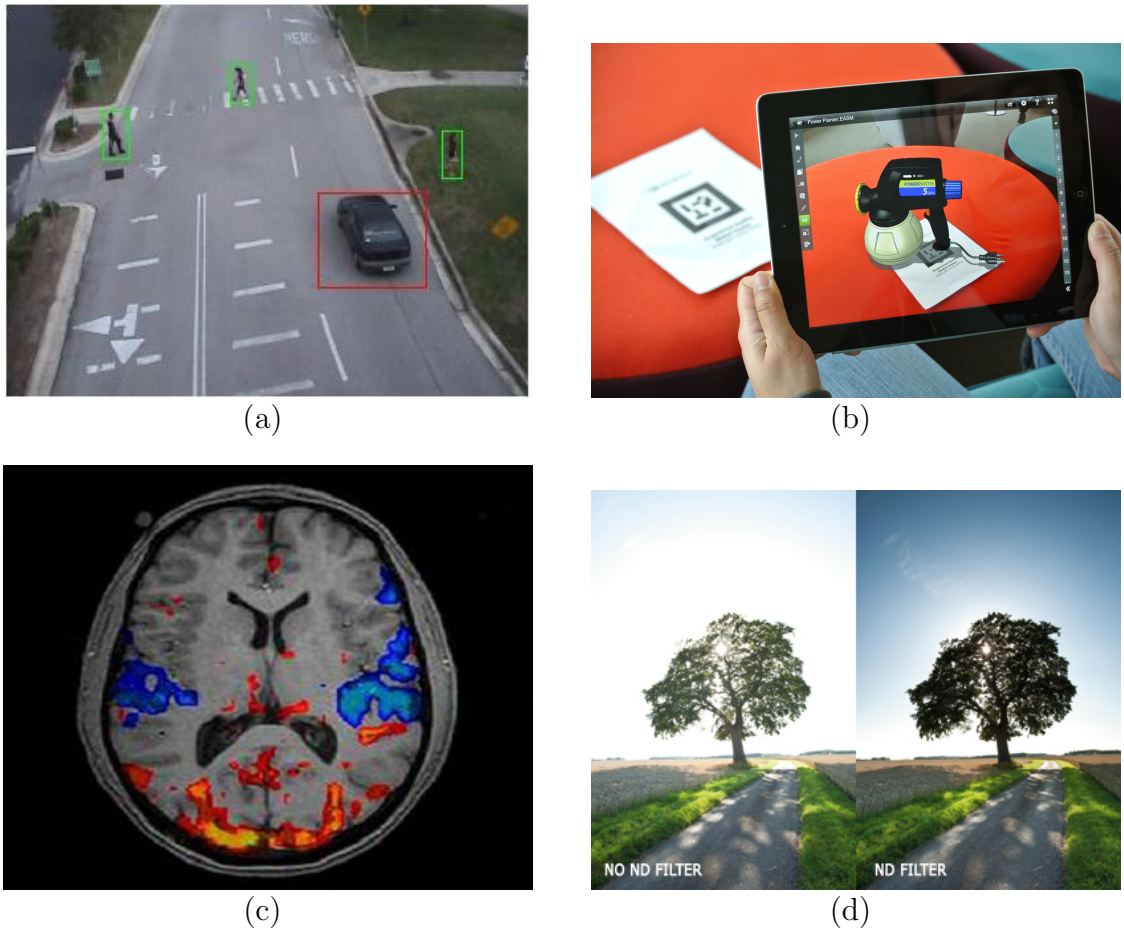
Algunos de los problemas concretos que trata de resolver la visión artificial son:

- Captura de imágenes en diferentes rangos del espectro electromagnético (imágenes infrarrojas, térmicas...).
- Procesado de imágenes con el fin de conseguir reducción de ruido, mejora de contraste, realzado...
- Detección y clasificación de objetos.
- Analizar o seguir objetos en movimiento.
- Registro de imágenes.

La visión computacional ha conseguido dar solución a estos problemas con muy buenos resultados, aplicando técnicas muy diversas y por ello, se ha expandido a prácticamente todos los ámbitos de la industria:

- Inspección en aplicaciones industriales (detección de efectos en líneas de fabricación).

- Videovigilancia.
- Banca (reconocimiento de textos manuscritos).
- Televisión (creación de escenas 3D, Ojo de Halcón en deportes...).
- Biometría.
- Medicina (diagnóstico por imagen, equipos fotogramétricos de medición biomecánica...).
- Videojuegos.
- Tráfico (detección y lectura de matrículas).



(a) (b) (c) (d)

Figura 1.2: (a) Clasificación de objetos (b) Realidad aumentada (c) Imagen de una FRMi (d) Filtrado de imágenes

Precisamente en este último ámbito, se ha avanzado mucho en los interfaces basados en visión para la interacción del usuario con la consola. Desde la aparición del Wiimote (figura 1.3(a)) en 2006, ha habido una revolución en el mundo de los mandos para consolas y se empezó a apostar por la visión computacional para revolucionar este sector. El primero que apostó por este tipo de soluciones fue Sony, con su

(a) (b) (c)

Figura 1.3: (a) Wiimote de Nintendo (b)Eyetoy de Sony (c) Kinect de Microsoft

dispositivo Eyetoy (figura 1.3(b)), el cual básicamente incorporaba una cámara web y un micrófono. A través de la imágenes capturadas con este periférico, los juegos basados en Eyetoy eran capaces de reconocer una serie de gestos, con los que el usuario podía interactuar con la consola. Finalmente, a finales de 2010, Microsoft empezó a comercializar un sensor totalmente revolucionario para su consola Xbox 360. Este sensor, llamado Kinect (figura 1.3(c)), incorpora una cámara de color, un sensor de profundidad y un micrófono matricial. Con Kinect no sólo se pueden reconocer gestos preestablecidos, sino que además se puede calcular la posición de cada una de las articulaciones del jugador. Esto permite que las posibilidades de interacción con la consola aumenten enormemente.

Aunque este sensor irrumpió en el mercado de los videojuegos de la mano de Microsoft, rápidamente el sector de la investigación, sobre todo orientada a visión, se interesó por él. Disponer de un sensor de profundidad con el que poder reconstruir escenas 3D prácticamente de inmediato y a un precio muy reducido fue el principal motivo de este movimiento.

1.3. Sensores y sistemas de seguimiento 3D

Dentro del campo de la robótica y la visión computacional existe un gran interés en desarrollar sistemas que sean capaces de generar información 3D para su uso en aplicaciones, fundamentalmente de creación de mapas, localización o seguimiento.

Los sistemas utilizados para reconstruir escenas o seguir objetos en 3D son variados: la triangulación, el uso de técnicas de tiempo de vuelo (TOF), sistemas de proyección de luz estructurada, aplicaciones basadas en detección de silueta y sombreado, técnicas de flujo óptico o, en los últimos años, *visual SLAM*.

La técnica de triangulación es una de las más habituales para el cálculo de información tridimensional. Esta técnica se basa en encontrar el punto de intersección de las rectas de retroproyección de un punto real observado por dos o más cámaras (figura 1.4). Para ello, es necesario calcular las correspondencias entre las imágenes

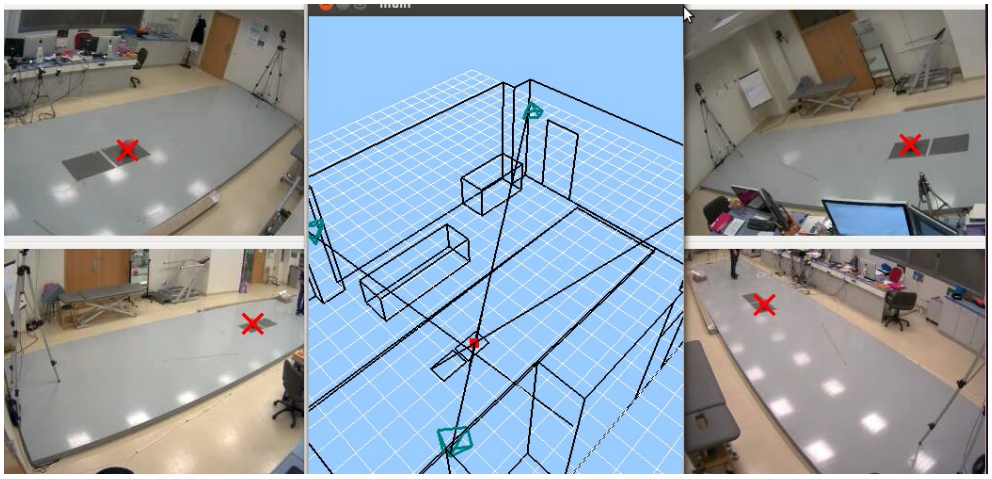
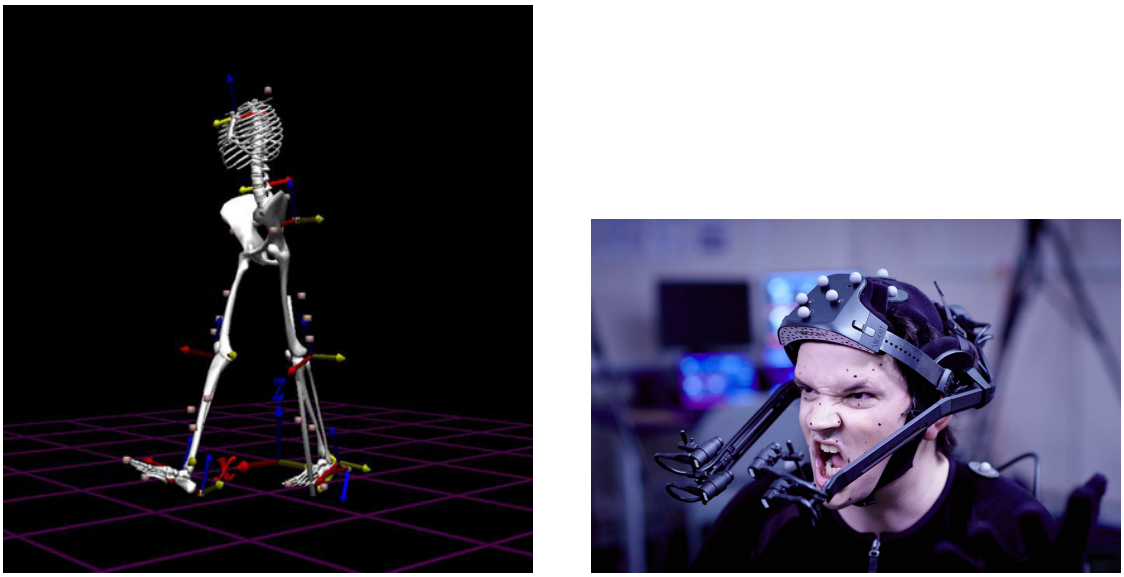


Figura 1.4: Reconstrucción por triangulación

capturadas por las diferentes cámaras. Con estos sistemas se consigue mucha precisión aumentado el número de cámaras que están captando el entorno, ya que nutre de información redundante para mejorar las correspondencias. Precisamente por este motivo se utilizan para el seguimiento en 3D de pequeños objetos. Un ejemplo de estos sistemas, es el sistema VICON que realiza un seguimiento de una serie de marcadores repartidos por el cuerpo de una persona utilizando múltiples cámaras de alta frecuencia y sensibilidad. Los datos capturados son más tarde utilizados, o bien con fines clínicos (análisis biomecánico, figura 1.5(a)), o en la producción de cine de animación (figura 1.5(b)). Actualmente hay sistemas comerciales que tienen empotrada esta técnica para ofrecer de manera transparente información 3D. El ejemplo más común es la cámara *Bumblebee* (figura 1.6).

Una variante de esta técnica es la disparidad. Para poder implementar un sistema basado en disparidad se debe disponer de al menos dos cámaras y, por lo general, dispuestas siguiendo un sistema canónico, es decir, situando las cámaras alineadas apuntando en la misma dirección (figura 1.7). El cálculo de la disparidad se realiza obteniendo la distancia entre las proyecciones de los puntos de una cámara con respecto a la otra, es decir, cuánto se ve desplazado el mismo objeto visto de una cámara con respecto a la otra. El principal problema que conlleva esta técnica es el cálculo de correspondencias. Debemos saber qué píxel de una imagen corresponde exactamente con el de la otra. Si este cálculo no se hace correctamente, obtendremos una disparidad errónea.

Los escáneres TOF basados en tiempo de vuelo determinan la distancia del objeto a la cámara cronometrando el tiempo que tarda un pulso de luz láser en ir del emisor al objeto y volver al receptor. Generalmente, estos sensores sólo miden la distancia directa de un punto por lo que es necesario variar la dirección del haz tras cada medida. Son sistemas de muy alto muestreo, de alta precisión (el error medio es inferior a un milímetro) y gran alcance. Un ejemplo de sistemas basados en esta tecnología son los sensores láser. La principal pega de estos sensores es que no con capaces de funcionar



(a) (b)

Figura 1.5: (a) Vicon para medicina (b) Vicon para animación

Figura 1.6: Cámara Bumblebee

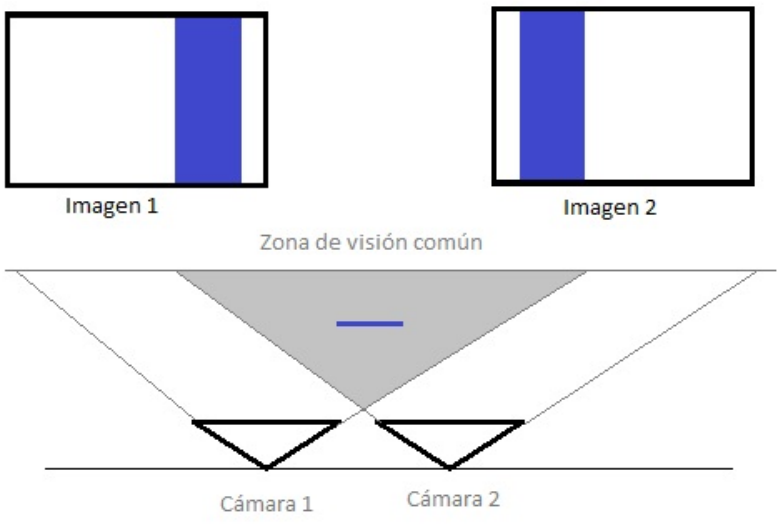


Figura 1.7: Configuración canónica estéreo

(a) (b)

Figura 1.8: (a) D-IMager de Panasonic Electric Works (b) SwissRanger de Mesa Imaging

contra superficies reflejantes como espejos, ni contra superficies transparentes como el cristal. Algunos de los sistemas comerciales basados en esta configuración son el Panasonic D-IMager o Mesa Imaging SwissRanger (figuras 1.8(a) y 1.8(b)) con un precio de 2.000$ y 9.000$ respectivamente.

Los escáneres de luz estructurada basan su funcionamiento en la emisión de un patrón de luz en la escena, captan la deformación del patrón con un receptor y analizan estas deformaciones para calcular la distancia a la que se encuentran los objetos. La principal ventaja de esos sensores es su velocidad. No necesitan escanear punto a punto, sino que escanean varios puntos o un campo entero por cada medición.

Las aplicaciones basadas en detección de silueta y sombreado, basa su funcionamiento en el estudio de las deformidades de las sombras que inciden en la escena. Se calcula el campo de normales asociada a esa escena y, a partir de ella, se calculan las distancias. Uno de los principales problemas de esta técnica consiste en la dificultad para detectar la concavidad y convexidad de los objetos y su relativa precisión.

Aplicando flujo óptico se puede determinar la posición 3D de un punto en la escena, basándose en un campo instantáneo de velocidades. Para determinar estas velocidades es necesario que las imágenes sean en movimiento, lo cual consigue muy buenas aproximaciones siempre y cuando se capturen superficies Lambertianas (tienen el mismo brillo en todas las direcciones), se trate de una fuente de luz ubicada en el infinito y cuando no exista una distorsión fotométrica.

Los sistemas basados en una arquitectura con una cámara suelen estar más orientados a aplicaciones de autolocalización 3D. Un ejemplo es el *Monocular Simultaneous Localization And Mapping* (Mono SLAM) [Davison *et al.*, 2007], asentado sobre filtros extendidos de *kalman*. Sin embargo, estos sistemas también son aplicables sobre mapeado 3D[Einhorn *et al.*, 2010]. Una evolución de este sistema monocular

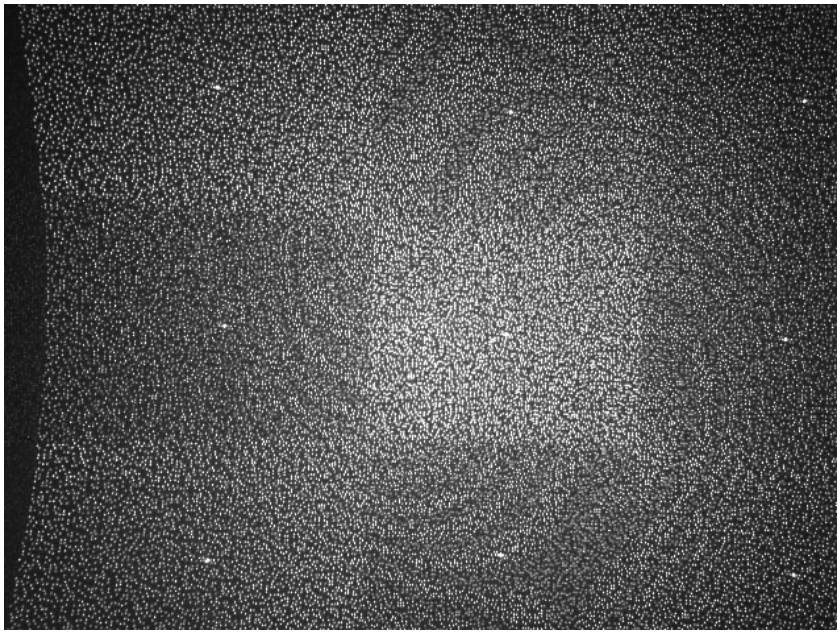


Figura 1.9: Patrón de luz estructurada de Kinect

de localización es el Parallel Tracking and Mapping (PTAM) [Klein y Murray, 2007], que separa el seguimiento del mapeo. Permitiendo su ejecución en hilos independientes aplicando optimizaciones por *ajuste de haces*. Con este nuevo enfoque, se consigue un rendimiento muy alto, permitiendo su ejecución incluso sobre dispositivos móviles en tiempo real. El principal problema de la aplicación de estas técnicas en monitorización de interiores, es la necesidad de estos algoritmos de captar algún tipo de movimiento de la cámara, por lo que no son aplicables directamente sobre cámaras estáticas.

Según esta clasificación, Kinect es englobado como un sensor de *proyección de luz estructurada*. En la figura 1.9, se puede apreciar el patrón utilizado. Gracias a su bajo precio y sus buenas prestaciones, este dispositivo ha tenido mucho éxito. Los componentes de este completo dispositivo son:

- Emisor de infrarrojos.
- Receptor de infrarrojos.
- Cámara de color.
- Micrófono.
- Motor.

1.4. ElderCare

Este trabajo fin de grado se encuadra dentro del sistema *ElderCare*. Este sistema nace dentro del laboratorio robótica de la Universidad Rey Juan Carlos[5](URJC), con el fin de complementar, a través de información tridimensional, a los sistemas convencionales de teleasistencia. Haciendo uso de datos 3D, *ElderCare* es capaz de detectar situaciones de riesgo para las personas, como pueden ser las caídas.

Las técnicas de localización y seguimiento aplicadas en *ElderCare* surgen con el proyecto fin de carrera titulado *Aplicación de seguridad basada en visión* [Pineda, 2006]. En él, se desarrolló un sistema que era capaz de localizar a una única persona vestida con colores llamativos (figura 1.10(a)). Para ello, se utilizaban cámaras de visión convencionales y era necesario conocer a priori el color de la vestimenta del sujeto.

La orientación de esta tecnología del Grupo de Robótica de la URJC hacia la teleasistencia, aparece con la evolución del citado proyecto con el trabajo fin de carrera *Seguimiento 3D visual de múltiples personas utilizando un algoritmo evolutivo multimodal* [Marugán Alonso, 2007]. En este trabajo, se mejoran las técnicas de seguimiento para que sea capaz de localizar a múltiples sujetos y donde el propio sistema es capaz de aprender automáticamente la apariencia cromática de cada uno de ellos. La misma autora realizó dos evoluciones más de *ElderCare* [Marugán Alonso, 2010a; Marugán Alonso, 2010b], donde se introdujeron grandes novedades:

- Introducción de primitivas 3D volumétricas (figura 1.10(b)).
- Mejoras de robustez en el aprendizaje de color utilizado.
- Desarrollo de un interfaz de usuario en Android para visualizar el entorno monitorizado desde el exterior utilizando dispositivos móviles.
- Pruebas piloto en entornos reales (figura 1.11).

La migración de este sistema hacia el uso de sensores RGB-D se realizó durante el desarrollo de un trabajo fin de grado previo [Rivas Montero, 2014] (figura 1.12), donde se consiguió implementar un sistema de teleasistencia completo, basado en el uso nubes de puntos, capaz de detectar una serie de situaciones de riesgo. Para la detección de personas en el citado proyecto, se utilizó la segmentación que viene integrada en el propio software del sensor, con la librería NITE, y un filtro de Kalman para abordar, de manera básica, la problemática del seguimiento. Todo ello, procesando fundamentalmente la información instantánea.

El presente trabajo supone la sexta evolución de *ElderCare*, siendo la segunda versión que utiliza como fuente sensorial los dispositivos RGB-D. Antes de utilizar estos novedosos sensores se utilizaban cámaras de color convencionales. Sin cambiar el enfoque teleasistencial de *ElderCare*, el objetivo principal de este proyecto es desarrollar un sistema que permita, al menos, la misma funcionalidad que la versión anterior,

[5]http://jderobot.org/ElderCare

(a) (b)

Figura 1.10: Versiones anteriores de ElderCare (a) Versión 1.0 (b) Versión 4.6

Figura 1.11: ElderCare en entornos reales

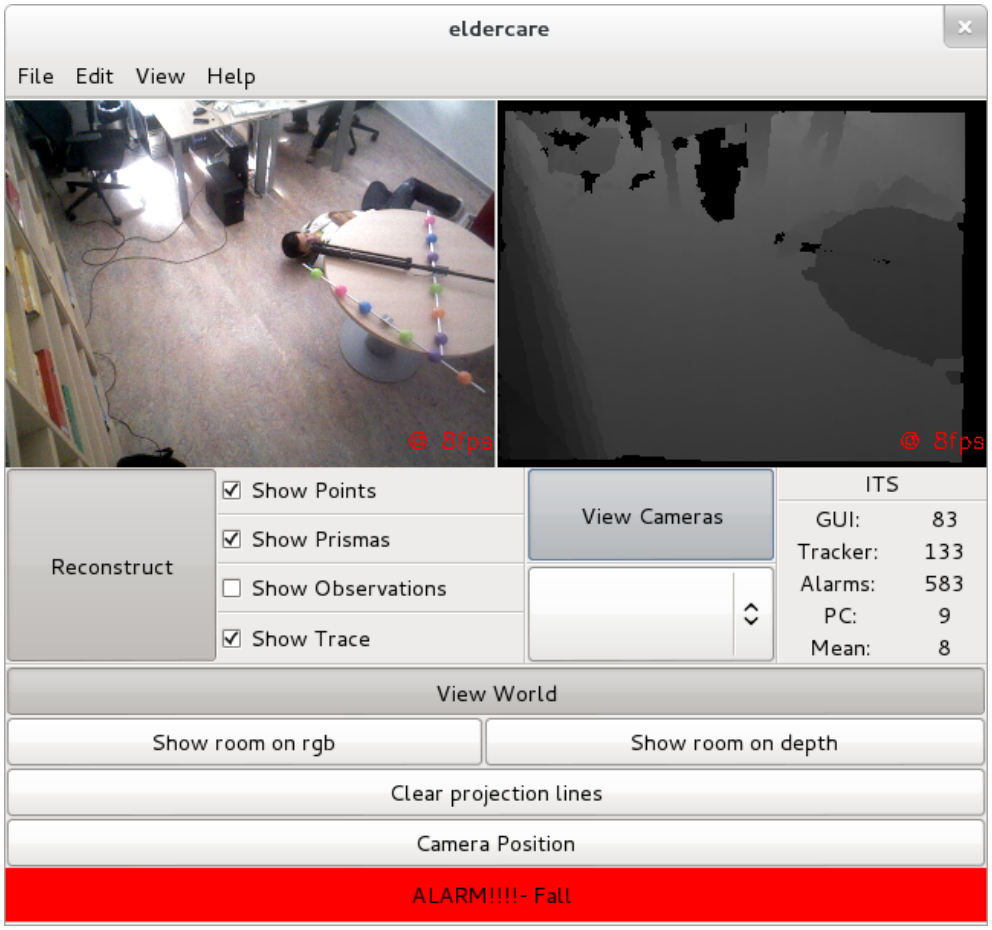


Figura 1.12: ElderCare 5.0.

basada en los mismos sensores, implementando un sistema de detección y seguimiento de personas propios, basado en algoritmos evolutivos. En concreto, con este trabajo se pretende diseñar y desarrollar un sistema robusto de seguimiento tridimensional de personas utilizando sensores RGB-D capaz de funcionar las 24 horas del día.

En el siguiente capítulo se detallan tanto los objetivos de este proyecto como la planificación y metodología empleada. Seguidamente, se describen el entorno de desarrollo software y los dispositivos hardware empleados. En el capítulo 4 se explican las contribuciones realizadas al entorno *JdeRobot* que sirven como infraestructura para este trabajo fin de máster. A continuación, en el capítulo 5 se define la solución conseguida al sistema de seguimiento 3D de personas junto con los resultados obtenidos y finalmente, en el capítulo 6 se exponen las conclusiones y las líneas de trabajo futuras generadas a partir de este trabajo.

Capítulo 2
Objetivos

Una vez presentados la motivación y el contexto en el que se desarrolla el presente trabajo, en este capítulo se detallarán los objetivos concretos que se pretenden alcanzar con este trabajo, así como los requisitos que debe cumplir la solución aportada.

2.1. Descripción del problema

Como se ha detallado en la introducción, los sistemas de teleasistencia actuales no son del todo completos. Necesitan la colaboración de la persona monitorizada para conocer que se ha producido una situación peligrosa, y entonces tomar las medidas oportunas.

El principal objetivo de este proyecto es desarrollar un sistema de seguimiento 3D de personas en interiores, capaz de detectar situaciones de riesgo de forma totalmente autónoma. Este sistema utilizará dispositivos RGB-D de consumo como fuente sensorial, siendo éstos los elegidos debido a que dan solución a algunos de los problemas básicos de la teleasistencia:

- Incorpora una cámara de vídeo para poder conocer desde el exterior qué está pasando en el área monitorizada.
- Ofrece información tridimensional del entorno.
- El sensor de profundidad es capaz de funcionar perfectamente en condiciones de total oscuridad, siendo capaz de cubrir las 24 horas 7 días a la semana.

Este objetivo global,ha sido articulado en varios subobjetivos:

1. Crear un *segmentador de primer plano* que tome como datos fuente mapas instantáneos de profundidad de sensores RGB-D. Los datos segmentados se convierten a nube de puntos utilizando un modelo calibrado, siendo éstos los datos de entrada al algoritmo de detección y seguimiento de personas en 3D.
2. Diseñar y desarrollar un *algoritmo de estimación 3D* que reciba una nube de puntos desde diferentes sensores y sea capaz de detectar la posición tridimensional de *todas las personas* presentes en la escena y seguirlas a lo largo del tiempo.

3. Diseñar y desarrollar un módulo de *alarmas* que, a partir de la secuencia de posiciones en 3D de las personas del entorno, detecte si se ha producido alguna situación de riesgo.

4. Desarrollar las *herramientas* necesarias para grabar datos en disco y poder reproducirlos posteriormente. Esto permitirá validar cambios en el algoritmo sobre exactamente los mismos datos fuente.

5. Validar experimentalmente el sistema desarrollado, para estudiar tanto la fiabilidad como la viabilidad de su uso en entornos reales.

2.2. Requisitos

Además de cumplir los objetivos presentados en el punto anterior, nuestro proyecto deberá satisfacer una serie de requisitos:

1. Se desarrollará bajo la arquitectura *JdeRobot*, implementando todos los componentes usados mediante el lenguaje de programación C++.

2. Deberá poder ejecutarse sobre Ubuntu 12.04 y Debian Wheezy tanto en x64 como en x32.

3. Los algoritmos desarrollados deben permitir una detección y seguimiento fluidos de múltiples personas en tiempo real.

4. El módulo de alarmas debe ser robusto frente a condiciones cambiantes de iluminación y oclusiones parciales.

5. El sistema desarrollado deberá ser fácilmente integrable en cualquier equipo de teleasistencia convencional, con el fin de complementar su funcionalidad.

2.3. Metodología

En el desarrollo de los componentes software de nuestro trabajo, el modelo de ciclo de vida utilizado ha sido el modelo en espiral basado en prototipos, ya que permite desarrollar el proyecto de forma incremental, aumentando la complejidad progresivamente y haciendo posible la generación de prototipos funcionales.

Este tipo de modelo de ciclo de vida nos permite obtener productos parciales que puedan ser evaluados, ya sea total o parcialmente, y facilita la adaptación a los cambios en los requisitos, algo que sucede muy habitualmente en los proyectos de investigación.

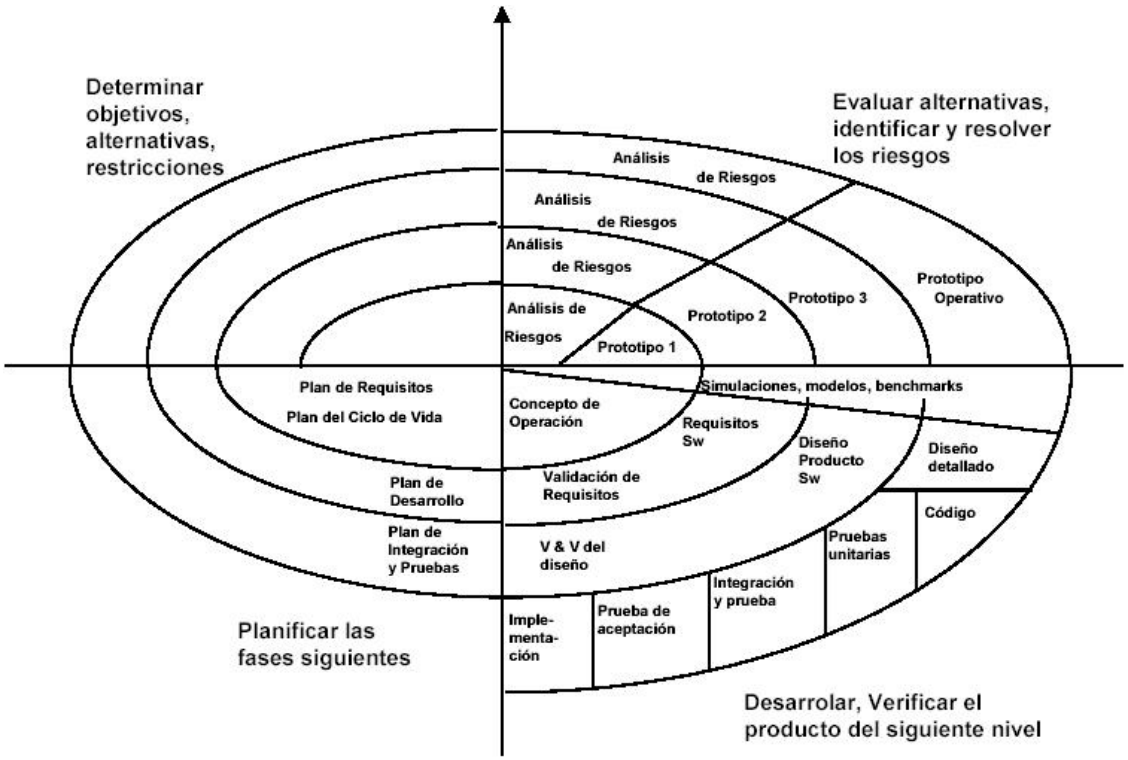
El modelo en espiral se realiza por ciclos, donde cada ciclo representa una fase del proyecto software. Dentro de cada ciclo del modelo en espiral se pueden diferenciar 4 partes principales que pueden verse en la figura 2.1, y donde cada una de las partes tiene un objetivo distinto:

- **Determinar objetivos.**
  Se establecen las necesidades que debe cumplir el producto en cada iteración teniendo en cuenta los objetivos finales, por lo que según avancen las iteraciones aumentará el coste del ciclo y su complejidad.

- **Evaluar alternativas.**
  Determina las diferentes formas de alcanzar los objetivos que se han establecido en la fase anterior, utilizando distintos puntos de vista, como el rendimiento que pueda tener en espacio y tiempo, las formas de gestionar el sistema, etc.. Además se consideran explícitamente los riesgos, intentando reducirlos lo máximo posible.

- **Diseñar, desarrollar y verificar.**
  Diseñaremos el producto siguiendo la mejor alternativa para poder alcanzar los objetivos del ciclo y se desarrollará según las especificaciones fijadas. Una vez diseñado e implementado el producto, se realizan las pruebas necesarias para comprobar su funcionamiento.

- **Planificar.**
  Teniendo en cuenta el funcionamiento conseguido por medio de las pruebas realizadas, se planifica la siguiente iteración revisando posibles errores cometidos a lo largo del ciclo y se comienza un nuevo ciclo de la espiral.



(a)

Figura 2.1: Modelo en espiral

Los ciclos que se han seguido en nuestro proyecto están relacionados con cada una de las etapas que se describirán en la siguiente sección. A lo largo de estas etapas, se han realizado reuniones semanales con el tutor del proyecto para negociar los objetivos que se pretendían alcanzar y para evaluar las alternativas de desarrollo.

Durante toda la realización de este proyecto, se ha utilizado una herramienta web donde se han ido reflejando las avances realizados, el mediaWiki. Esta herramienta ha servido como cuaderno de bitácora y como apoyo fundamental en las reuniones semanales. En la página web [1] están reflejados todos los pasos dados desde el comienzo de este proyecto a principios de 2013 así como diverso material multimedia con todos los resultados obtenidos durante el proceso.

2.4. Planificación

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

- **Asentamiento con el entorno software.**
  JdeRobot es la plataforma base sobre la que se desarrollan la mayoría de los proyectos realizados en el laboratorio de robótica de la URJC, siendo ésta la base sobre la que se realizaron todas las anteriores versiones de *ElderCare*. La primera fase de este proyecto se centrará en una profunda refactorización de la versión anterior de este sistema, que ya utilizaba sensores RGB-D, con el fin de eliminar todas las dependencias de la librería NITE y crear nuevas clases donde desarrollar el algoritmo de detección y seguimiento de personas en 3D.

- **Creación de herramientas de reproducción enlatada.**
  Con el fin de validar las mejoras realizadas durante el desarrollo en las diferentes técnicas implementadas en el proyecto es necesario realizar una herramienta que sea capaz de capturar los datos de los sensores y volcarlos a disco para, posteriormente, reproducirlos utilizando la misma capa que ofrecen los servidores originales. Es decir, que sea capaz de servir los datos grabados como si se tratase realmente del dispositivo físico.

- **Diseño y desarrollo de un segmentador de primer plano.**
  Para simplificar y ayudar en la detección de persona se desarrolla un segmentador de fondo que debe tomar como fuente el mapa de profundidad, convirtiendo los datos segmentados en el primer plano a nube de puntos. Esta conversión se debe realizar con el dispositivo calibrado, anclando los datos tomando como referencia el origen de coordenadas del entorno que se esté monitorizando.

[1] http://jderobot.org/Frivas-tfm

- **Diseñar y desarrollar un sistema de búsqueda global (exploración).**
  Sobre los datos de primer plano es necesario aplicar alguna exploración grosera para determinar qué zonas son de interés para el sistema, siendo éstas susceptibles de pertenecer a una persona (o ser parte de ella).

- **Diseñar y desarrollar un sistema de búsqueda local (explotación).**
  Sobre las zonas de interés marcadas por la exploración a groso modo es necesario realizar una búsqueda local fina para determinar si en esa zona hay una persona. Y en caso de haberla, determinar un representante que explique a esa persona.

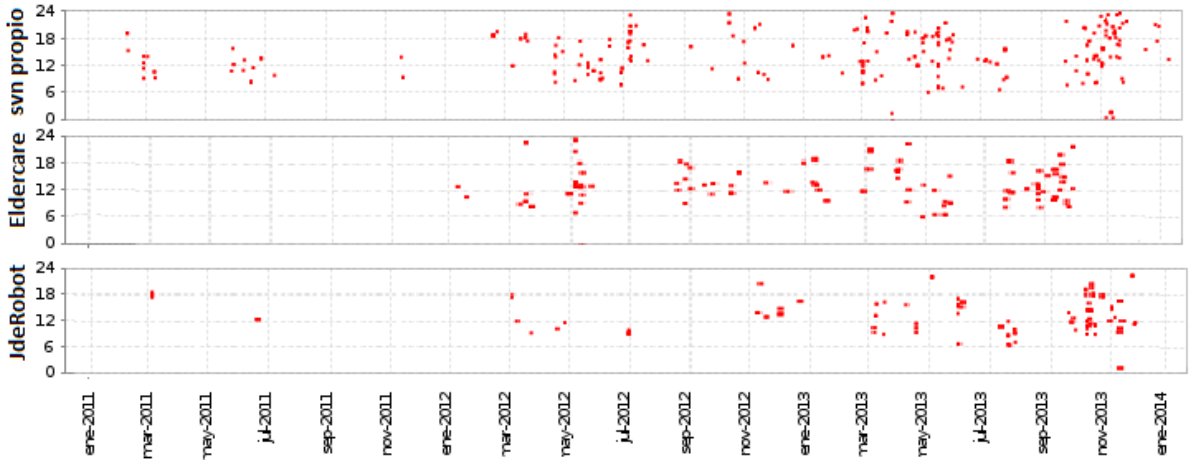
- **Diseñar y desarrollar el algoritmo evolutivo multimodal.**
  El algoritmo implementado en esta fase, consiste en unificar las técnicas de exploración y explotación estudiadas en las fases anteriores mediante el diseño de un algoritmo evolutivo multimodal con su propio método de evolución y una función de evaluación de resultados.

- **Ajustar y mejorar el módulo de alarmas.**
  Volver a desarrollar un módulo de alarmas que cumpla las mismas funciones que el incluido en la versión anterior de *ElderCare* pero ajustándose a las nuevas especificaciones.

- **Validación experimental de desarrollos realizados.**
  Finalmente, se realizaron una serie de pruebas en entornos reales para comprobar tanto el correcto funcionamiento del sistema como la fiabilidad y robustez del mismo. Estas pruebas se realizaron en un domicilio particular y en una habitación de una residencia de ancianos.

2.5. Esfuerzo Temporal del proyecto

Este proyecto empezó a fraguarse a principios de 2011. En la figura 2.2 se ha representado de manera visual el esfuerzo temporal realizando en este proyecto. Igualmente se pueden observar los aportes en los diferentes repositorios utilizados en este proyecto: el propio del trabajo fin de máster, uno específico para *ElderCare* y el repositorio de *JdeRobot*.



(a)

Figura 2.2: Esfuerzo temporal del proyecto

Capítulo 3
Plataforma de desarrollo

En este capítulo, se describirá tanto la plataforma hardware (los sensores utilizados) como las librerías sobre las que nos hemos apoyado para desarrollar este proyecto.

El sistema operativo utilizado en la versión final de este proyecto ha sido la versión oficial de Debian. En concreto, se ha utilizado la última versión estable (Wheezy, 7.5), la cual tiene soporte para todas las herramientas y aplicaciones que necesitamos.

3.1. JdeRobot

JdeRobot [1] es una herramienta de desarrollo para aplicaciones de robótica, domótica y visión artificial que nació en el año 2003 a partir de la tesis doctoral del profesor José María Cañas Plaza [Plaza, 2003]. En la actualidad, incluye soporte a multitud de sensores y actuadores, y todas las herramientas necesarias para la interconexión de componentes. *JdeRobot* ha sido diseñado para ayudar a programar software inteligente, está escrito en C++ y proporciona un entorno de programación básico orientado a componentes distribuidos y multilenguaje capaces de comunicarse entre sí. Este *middleware* ha sido evolucionado y es actualmente mantenido por los miembros del grupo de robótica de la Universidad Rey Juan Carlos (URJC).

JdeRobot está orientado a sistemas distribuidos, sirviendo de enlace a través de conexiones TCP/IP para los distintos componentes. Para realizar estas conexiones, *JdeRobot* se apoya el la librería ICE (Internet Communications Engine) de Zeroc [2]. ICE ofrece la posibilidad de conectar componentes escritos en distintos lenguajes (C++, .NET, Java, Python, Objective-C, Ruby, PHP y ActionScript) y que corren en las arquitecturas y sistemas operativos más comunes. Siguiendo este paradigma, un sistema basado en *JdeRobot* tendría el esquema representado en la figura 3.1. Cada uno de los componentes se conecta a un dispositivo o realiza una tarea concreta, conectándose entre sí utilizando la abstracción de *proxies*. Esto permite que cada componente pueda ser ejecutado en un computador distinto, con una arquitectura distinta e incluso con sistemas operativos distintos. Si dos componentes se ejecutan

[1] http://jderobot.org
[2] http://www.zeroc.com/

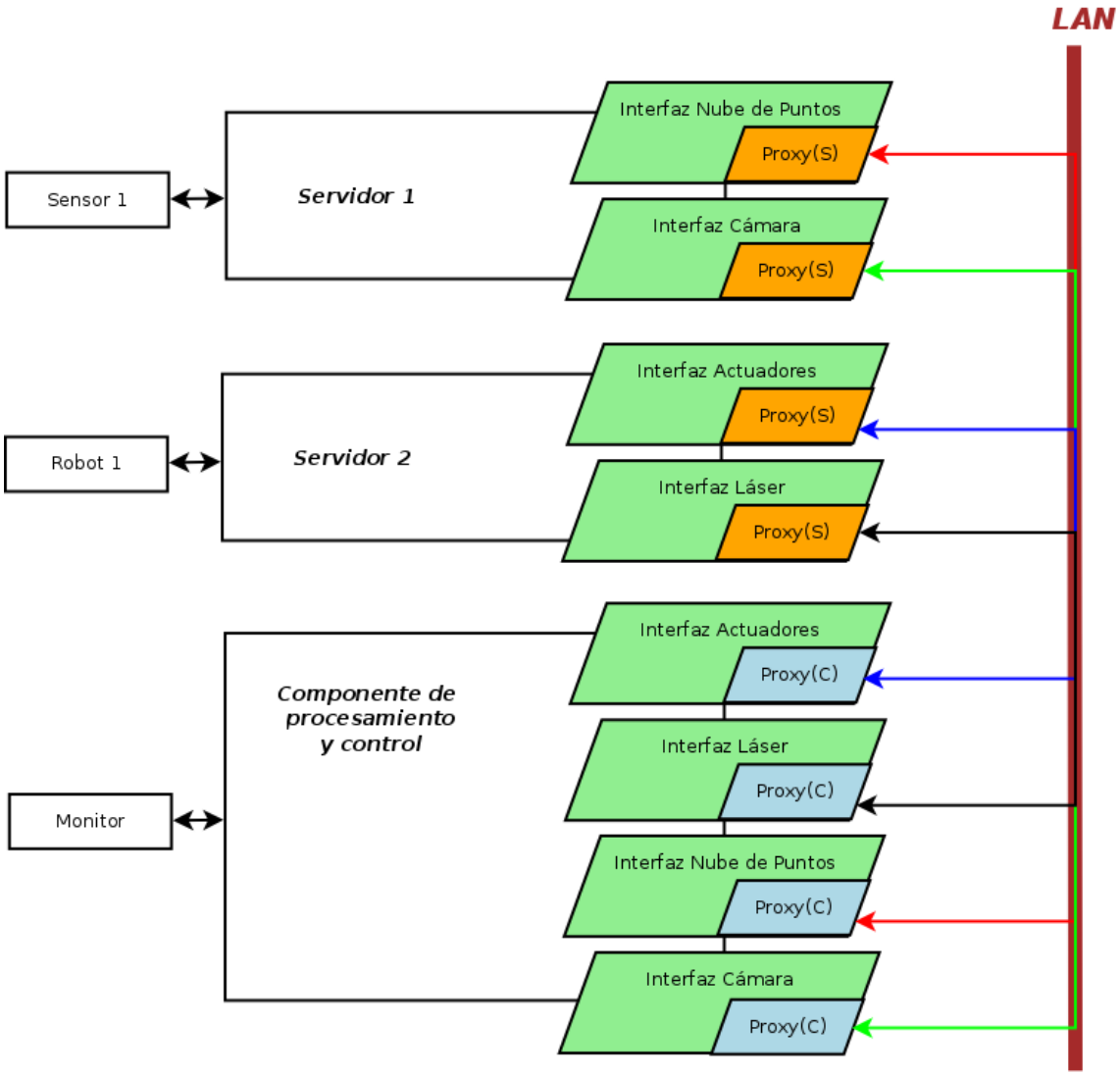


Figura 3.1: Estructura de la interconexión de componentes utilizando JdeRobot. (S) Servidor (C) Cliente

en un mismo ordenador, ICE se encarga de encapsular la conexión basada en *proxies* utilizando memoria compartida para mejorar el rendimiento de los mismos. Cada uno de los *proxies* tiene una función concreta e implementan las funcionalidades de un interfaz determinado. Por ejemplo, un componente servidor que se conecte a un robot puede servir numerosos sensores como cámaras, encoders o actuadores y cada uno tiene un interfaz propio. De esta forma, se pueden realizar conexiones selectivas en la que un cliente en concreto se conecte a aquellos interfaces del servidor que realmente necesita.

JdeRobot permite crear servidores que se conectan directamente a los componentes físicos y ofrecen los datos de los sensores, o bien clientes que reciben datos, los procesan y generan información a partir de ella. ICE, mediante su lenguaje *slice*, permite describir interfaces que son utilizados como comunicación entre componentes. Una vez definido el interfaz, sólo es necesario implementar las funciones necesarias para devolver los datos en el formato adecuado que recibirá el componente al que esté conectado.

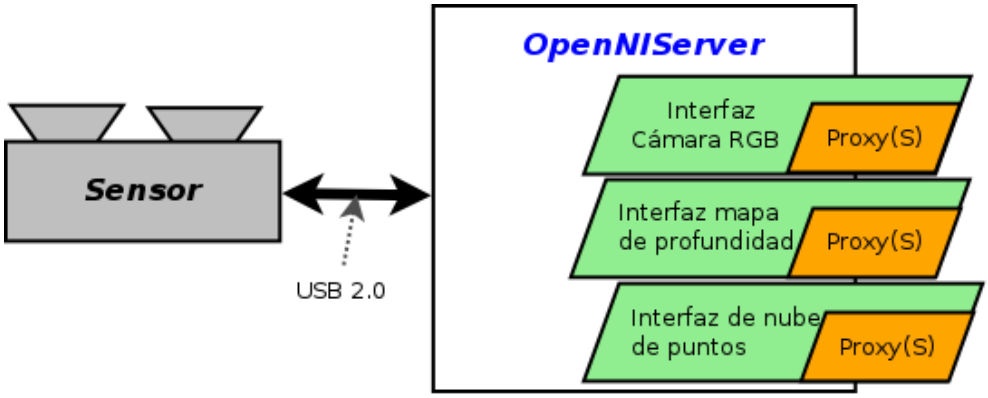


Figura 3.2: Esquema OpenNIServer

Debido al gran nivel de encapsulación que nos ofrece *JdeRobot*, la tarea de interactuar con el alto número de componentes de cualquier sistema robótico es mucho más sencilla. Dispondremos de todos los componentes detrás de interfaces ICE, a los que sólo es necesario pedirle datos para procesarlos y trabajar con ellos.

La versión de *JdeRobot* utilizada para realizar este proyecto es la última versión estable, la 5.2.3.

3.1.1. Componentes utilizados de JdeRobot

En este proyecto, se han utilizado un serie de componentes disponibles en el entorno *JdeRobot*, *OpenNIServer* para la conexión con los sensores, *RgbdViewer* para representar gráficamente la información 3D, ofrecida por el componente *OpenNIServer*. Además, el componente *RgbdCalibrator* para la calibración de los dispositivos, y por último el componente *RemoteConfigurator* para enviar parámetros de configuración de forma remota al sistema desarrollado.

3.1.2. Componente OpenNIServer

Este componente, implementado durante el desarrollo de mi trabajo fin de grado [Rivas Montero, 2014], es un servidor de *JdeRobot* que utiliza OpenNI para dar soporte a diversos sensores RGB-D. Este componente es capaz de proporcionar las imágenes de la cámara de color, el mapa de profundidad y también ofrece la posibilidad de combinar las dos anteriores en forma de nube de puntos (figura 3.2). Incorpora también funcionalidades de la librería NITE para la segmentación de personas, aunque éstas no han sido utilizadas en el presente proyecto. La dependencia de NITE está controlada desde los comandos de compilación, permitiendo compilar el componente sin tener esta librería instalada en el ordenador, desactivando las funcionalidades correspondientes.

3.1.3. Componente RgbdViewer

Este visor, implementado también durante mi trabajo fin de grado, es un cliente de *JdeRobot* capaz de representar gráficamente toda la información ofrecida por el componente *OpenNIServer* (figura 3.3(a) y 3.3(b)). Una funcionalidad útil de este componente es la capacidad de representar la información de un mapa junto la información 3D de la nube de puntos, lo cual permite al usuario comprobar de forma visual y sencilla el correcto funcionamiento del sensor.

3.1.4. Componente RgbdCalibrator

El componente, desarrollado por Roberto Calvo [3] permite calibrar cualquier sensor RGB-D siguiendo un proceso semiautomático basado en un patrón conocido (figura 3.4). Una vez colocado el dispositivo en la posición en la que se quiera calibrar, se sitúa el patrón en un zona captada por el sensor y mediante su interfaz gráfico se seleccionan diversas partes del patrón. Una vez hecho esto, el componente es capaz de devolver las matrices KRT necesarias para anclar el dispositivo al origen de coordenadas de nuestro entorno. Estas matrices integran los parámetros tanto intrínsecos como extrínsecos (posición y orientación) del sensor.

3.1.5. Componente RemoteConfigurator

Poder configurar aplicaciones o sistemas de forma remota es muy útil y en ocasiones indispensable. En el caso concreto del sistema planteado en este proyecto, donde la aplicación de teleasistencia estará instalada localmente en la casa del sujeto monitorizado, es vital poder hacer modificaciones en la configuración del sistema de forma remota, evitando el desplazamiento físico y simplificando todo el proceso.

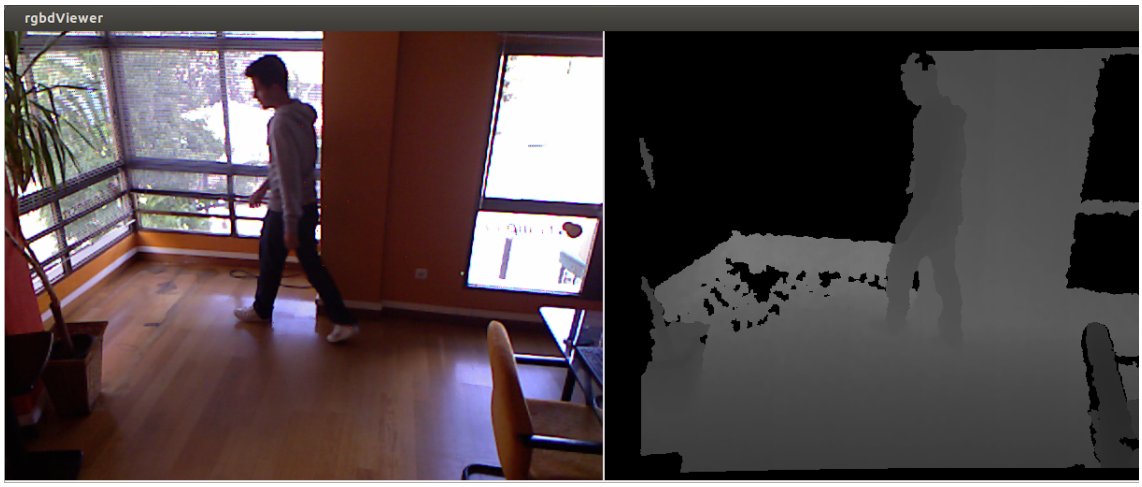
El componente RemoteConfigurator (figura 3.5), es capaz de, partiendo de una plantilla XML, generar un interfaz gráfico con todos los parámetros configurables de otro componente, permitiendo al usuario modificar de forma sencilla cualquiera de ellos. Una vez terminado, envía en caliente el resultado como un fichero XML, con todos los parámetros necesarios, al componente remoto permitiendo que éste modifique en tiempo de ejecución su comportamiento.

3.2. Sensores RGB-D

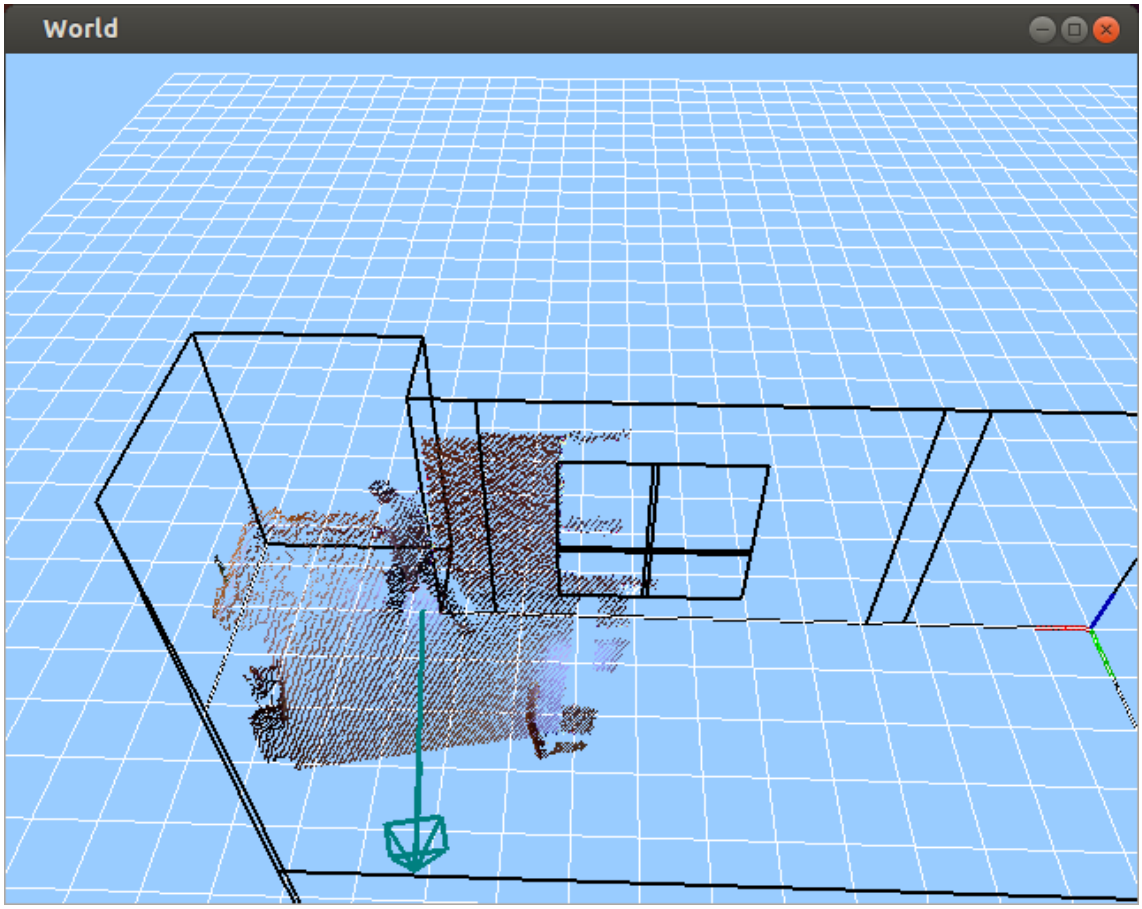
Con Kinect apareció un sensor que ofrece de forma simultánea una imagen en color y el mapa de profundidad correspondiente. Es capaz de funcionar a unos 30 fotogramas por segundo (fps) y su coste relativamente bajo.

En esta sección, describiremos sus características técnicas, las opciones software para su manejo y un estudio experimental que se ha realizado con el fin de caracterizarlo empíricamente.

[3] http://svn.jderobot.org/jderobot/trunk/src/stable/components/rgbdCalibrator/



(a)



(b)

Figura 3.3: (a) Imágenes (b) Nube de puntos

Figura 3.4: Patrón de calibración

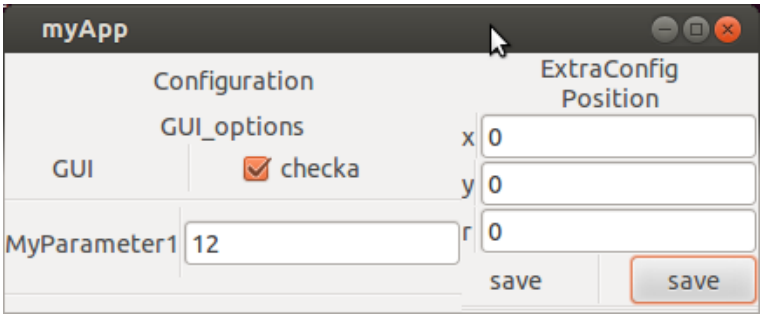


Figura 3.5: Ejemplo del interfaz gráfico del componente RemoteConfigurator

3.2.1. Descripción técnica

El fabricante que está detrás de la tecnología sobre la que se apoya Kinect es PrimeSense[4]. Este fabricante israelí, es el responsable del chip PS1080, el cual se encarga de controlar todo el sistema. En la tabla 3.1 están reflejadas las especificaciones oficiales distribuidas por Microsoft del sensor Kinect. Aunque la tabla refleje que el rango de actuación del sensor está entre 0.8 y 3.5m, realmente es bastante mayor, llegando hasta los 8-10 metros. Microsoft lo acotó a ese rango porque es donde tiene mayor precisión su algoritmo de reconocimiento gestual, y por lo tanto, es donde desempeña correctamente su función de interfaz de interacción con la consola.

Property	**Value**
Angular Filed-of-View	57º horz., 43º vert.
Framerate	approx. 30Hz
Nominal spatial range	640x480 (VGA)
Nominal Spatial resolution (at 2m distance)	3mm
Nominal depth range	0.8m - 3.5m
Nominal depth resolution (at 2m distance)	1 cm
Device connection type	USB (+ external power)

Tabla 3.1: Especificaciones de Kinect

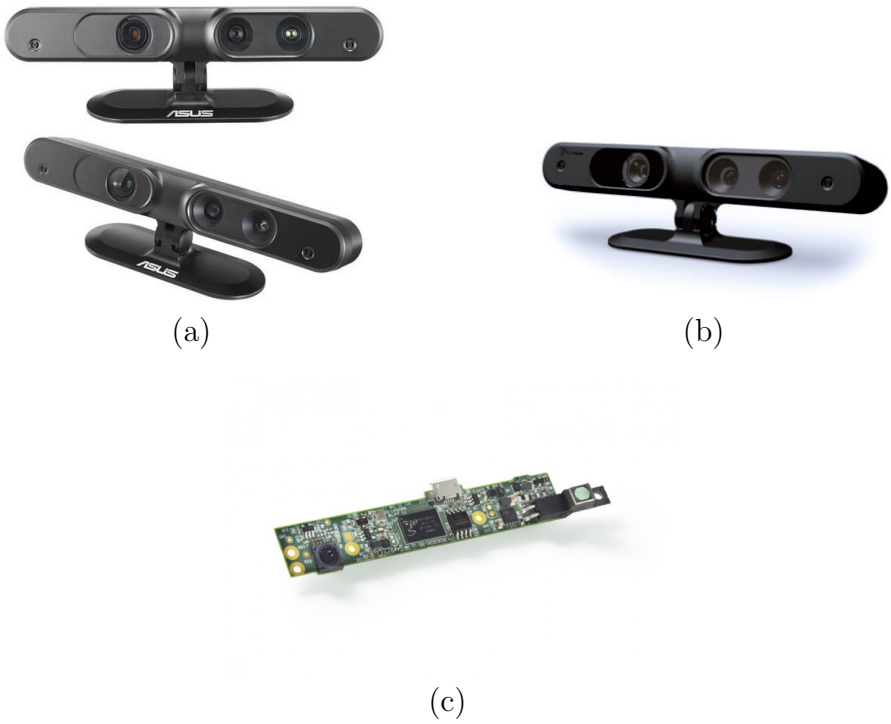
Estos sensores RGB-D empezaron a popularizarse en el ámbito de la visión artificial y por ello, la empresa taiwanesa Asus comenzó a comercializar un sensor llamado Xtion (figura 3.6(b)). Este sensor está basado en el mismo chip de PrimeSense que monta Microsoft en Kinect. Xtion se comercializó como un sensor susceptible de ser incorporado en aplicaciones de visión artificial, facilitando su conexión a través de USB 2.0 alimentado e incorporando todas las herramientas necesarias para acceder a él.

A la hora de aplicar este tipo de sensores a cualquier aplicación de visión artificial, existen una serie de consideraciones que se deben tener en cuenta:

- **Precisión y linealidad**
  Las mediciones de los sensores basados en el chip PS1080 son bastante precisas y a la vez lineales. En la figura 3.7, se puede ver el resultado del análisis realizado para comprobar la precisión y linealidad del sensor. En este experimento se promediaron todos los valores de profundidad devueltos por el dispositivo apuntando a una superficie plana paralela al plano imagen del sensor. Este procedimiento se repitió colocando el sensor a diferentes distancias de la superficie plana, dentro de su rango de actuación.

  Utilizando este tipo de medidas se consiguen valores muy precisos, ya que estamos obteniendo el valor final de profundidad promediando numerosas medidas. Si observamos el comportamiento de la distancia para un punto en concreto, este valor oscila bastante alrededor de la distancia real (figura 3.8).

[4]http://www.primesense.com/



(a) (b)

(c)

Figura 3.6: (a) Xtion Pro Life. (b) Primesense Carmine 1.09. (c) Caprie

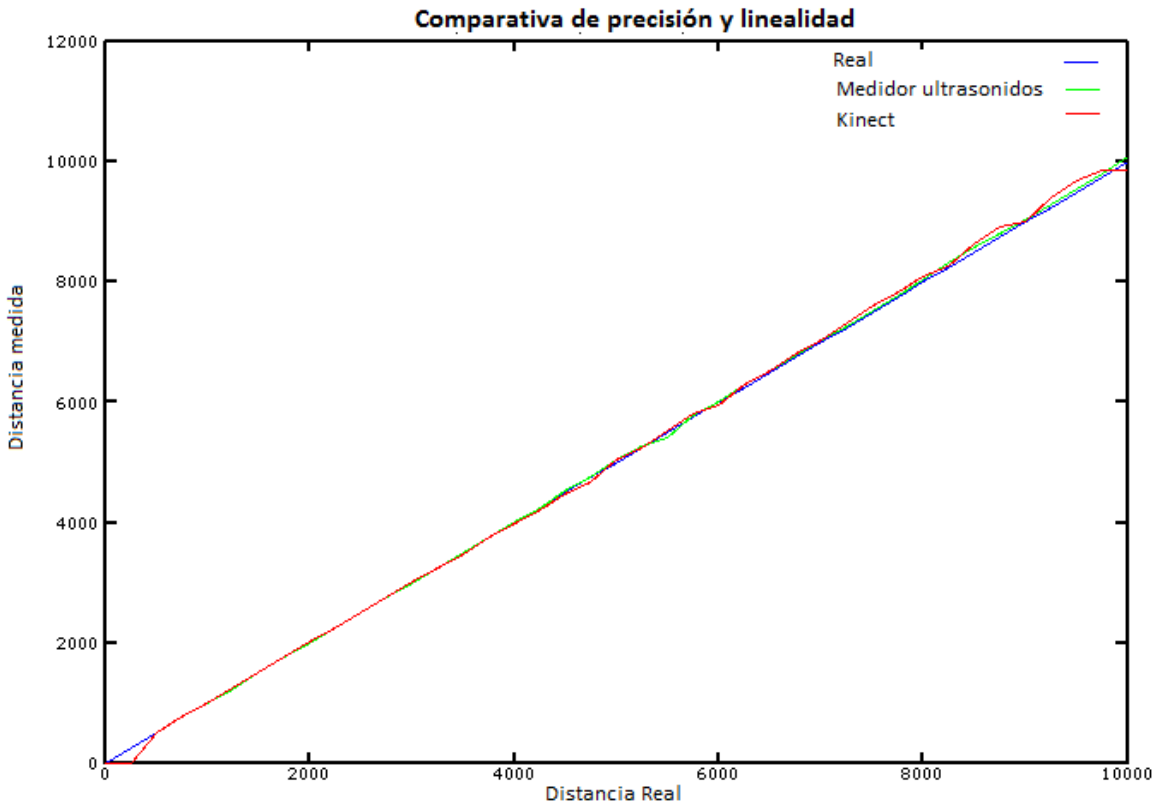


Figura 3.7: Representación de los valores del sensor de profundidad en diferentes distancias

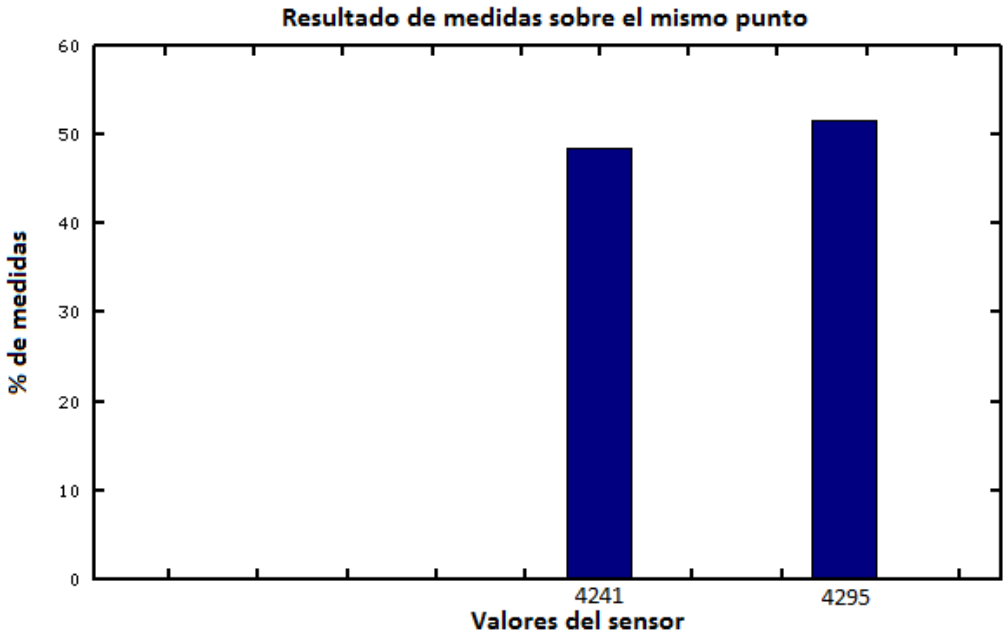


Figura 3.8: Variación de las medidas de profundidad de un píxel

- **Precisión espacial**
  Uno de los grandes problemas de estos sensores, es su precisión espacial. Es muy habitual que, por ejemplo, en los bordes de un objeto, el valor devuelto por el sensor oscile entre la distancia del objeto y la distancia que hay tras el mismo. En la figura 3.9, se puede ver el resultado de un experimento, en el cual se colocó una silla delante de una pared a 1.4 metros del sensor, y donde la distancia a la pared era de 2.6 metros (figura 3.10). El píxel sobre el que se obtiene el valor de distancia es uno de los píxeles en el borde superior de la silla, marcado en la imagen anterior.

- **Sombras**
  Debido a la propia configuración del sensor, donde el emisor está separado unos 8 cm del receptor de infrarrojos, se generan numerosas zonas de sombras alrededor de los objetos. Este fenómeno se puede apreciar gráficamente en la figura 3.11.

- **Tiempo de calentamiento**
  Un factor a tener el cuenta a la hora de utilizar estos sensores es que tardan cierto tiempo en estabilizar sus mediciones. En el caso del sensor Kinect este tiempo ronda los 40 segundos según el estudio realizado en la universidad de Aarhus [Andersen *et al.*, 2012]. Sin embargo, tras haber realizado el mismo estudio con la infraestructura utilizada en el presente proyecto (OpenNI 2.2.0.11 y Asus Xtion Pro Live) los resultados han sido muy diferentes, obteniendo valores estables en el sensor desde prácticamente la primera captura (figura 3.12).

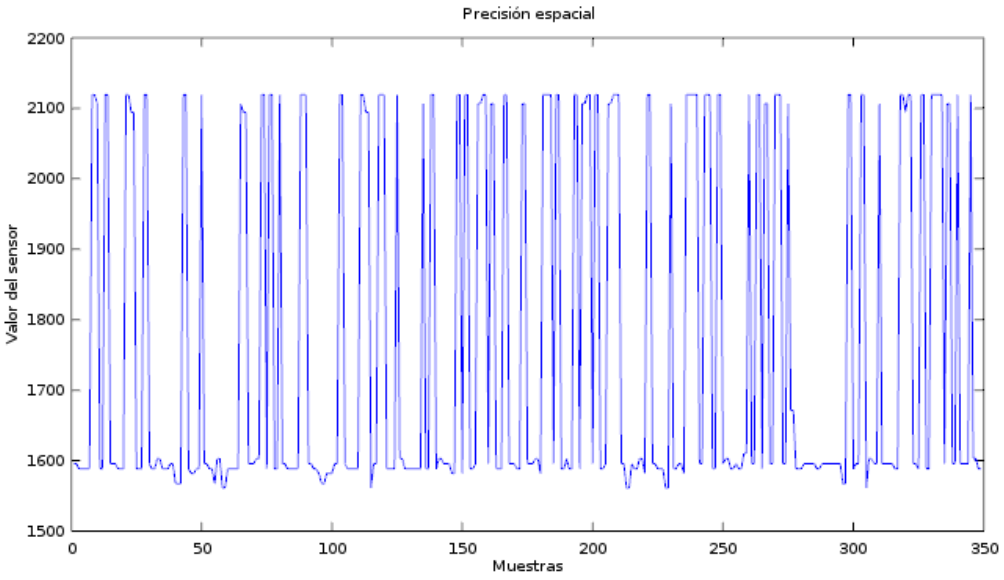


Figura 3.9: Precisión espacial.

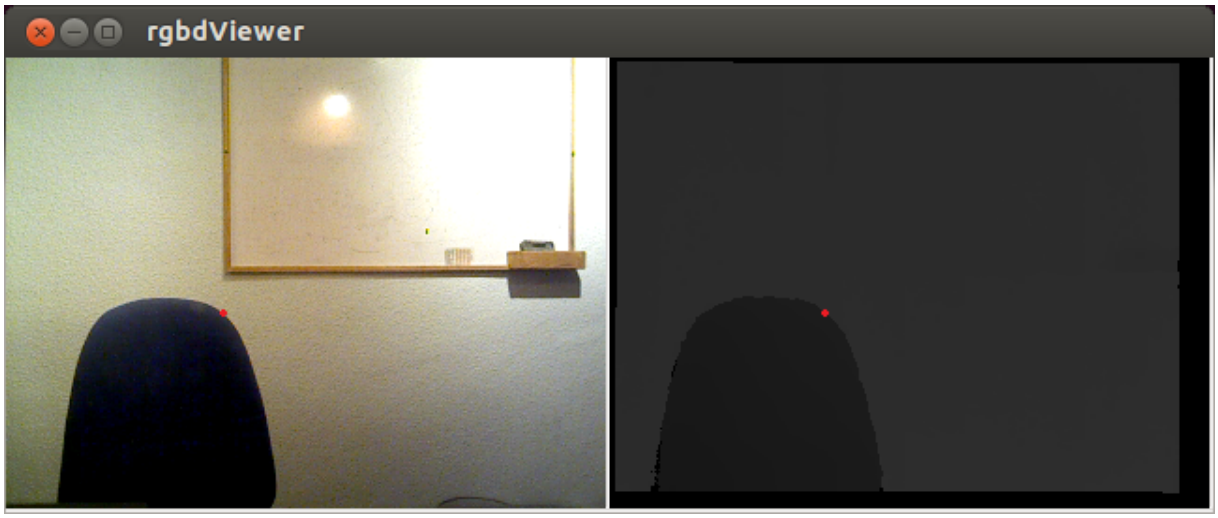


Figura 3.10: Escenario de pruebas de precisión espacial.

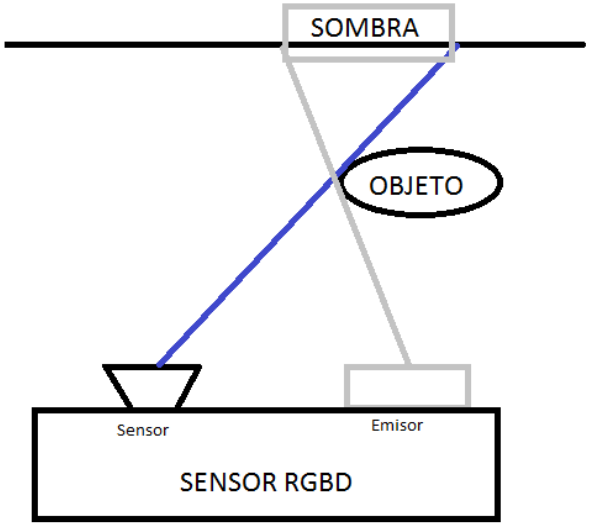


Figura 3.11: Configuración del sensor

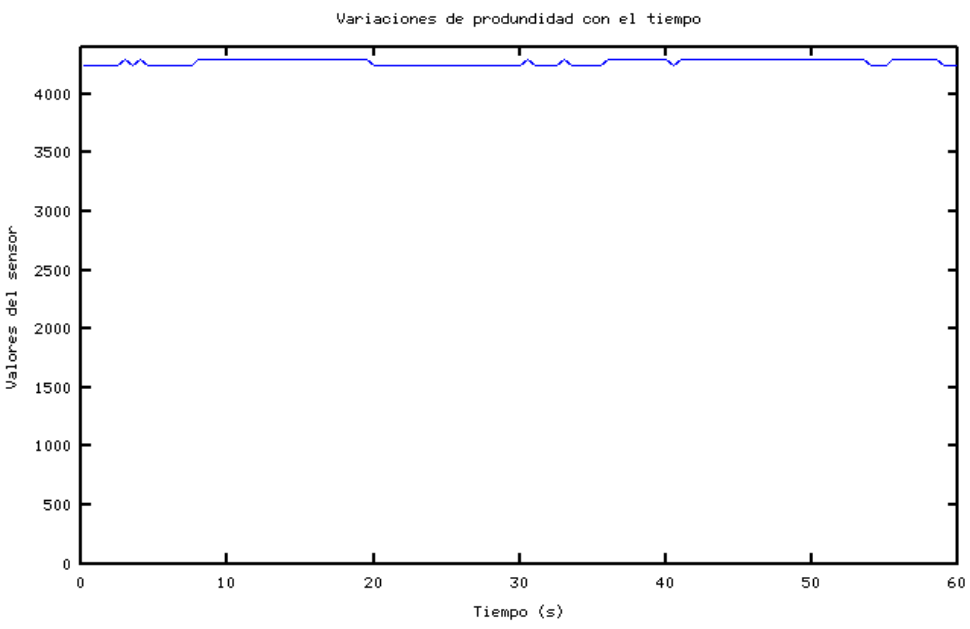


Figura 3.12: Tiempo de estabilización de las mediciones

3.2.2. Acceso software para desarrolladores

El sensor Kinect apareció como un interfaz sin mandos para jugar a la consola Xbox 360 de Microsoft. En el momento de su comercialización, no se aportó ningún tipo de soporte para acceder a los datos de este sensor desde un ordenador. Una vez que se entrevió el potencial de este dispositivo, se empezó a generar un gran interés por crear un Interfaz de Programación de Aplicaciones (API), que permitiese conectar este nuevo sensor al ordenador y poder acceder a los datos del mismo, con el fin de desarrollar nuevas aplicaciones. Es por ello, que la empresa Adafruit ofreció en noviembre de 2010 una recompensa de 3.000 dólares para aquél desarrollador capaz de acceder al sensor y crear un API con las funciones básicas para su uso, todo ello bajo licencias de software libre. La competición no duró más que unos días, ya que el 10 de noviembre Adafruit anunció que el ganador de la recompensa era el español Héctor Martín, responsable de la librería *libfreenect*, posteriormente llamada *OpenKinect*[5].

Freenect y OpenKinect

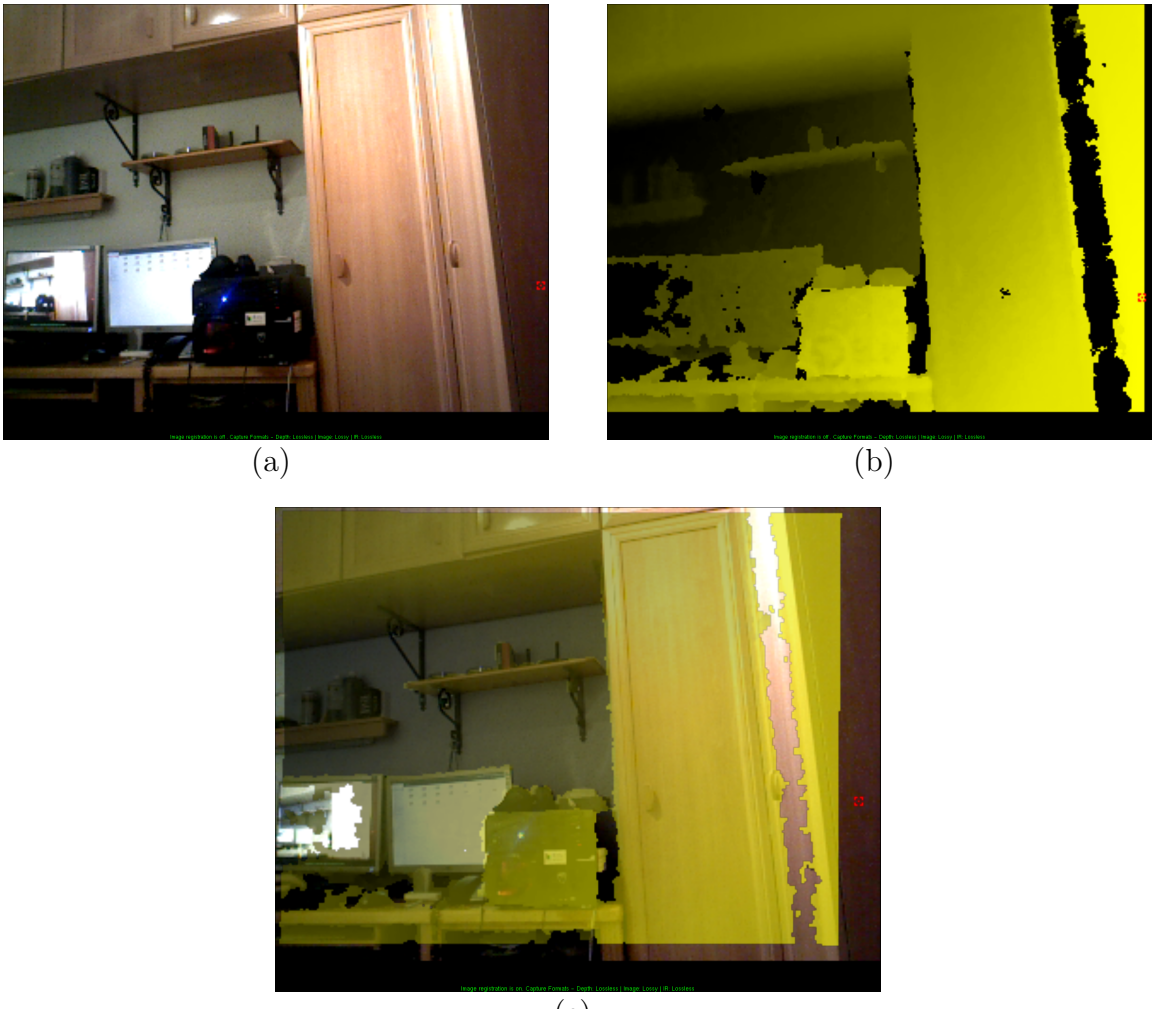
Como primer API de software libre para los sensores Kinect, *libfreenect* tuvo mucho éxito, ya que ofrecía las funciones básicas para poder acceder tanto la imagen de color de su cámara RGB como a los valores del sensor de profundidad. Los principales problemas de esta librería eran su alto consumo de recursos al acceder al sensor, y que no realiza ningún tipo de procesado de los datos, ofreciendo los datos crudos directamente. Esto implica que los valores que se obtienen a través de este API no corresponden directamente con los valores reales de distancia, y que a diferencia del resto de entornos no realiza una linealización de los mismos.

OpenNI y NITE

Un mes después de la aparición de *libfreenect*, PrimeSense publicó su propia librería para acceder a los sensores basados en el chip que ellos mismos fabricaban. Esta librería se liberó con licencia Apache 2.0, bajo el nombre de OpenNI[6]. Esta librería no sólo ofrece acceso a los datos de los diferentes sensores del dispositivo, sino que también incorpora diversas funciones muy útiles:

- **Registro de las imágenes de profundidad y de color**
  Aplicando este registro, cada píxel de la imagen de color corresponde directamente con el mismo píxel del mapa de profundidad (figura 3.13).
- **Sincronización entre sensores**
  OpenNI ofrece un parámetro configurable para que se asegure la sincronización entre los datos obtenidos por los diferentes sensores (RGB y profundidad).
- **Selección entre diversos modos de vídeo**
  Se puede configurar tanto la resolución como los fps de los sensores.

[5] https://github.com/OpenKinect/libfreenect
[6] http://www.openni.org



(a) (b)

(c)

Figura 3.13: (a) Imagen de color (b) Mapa de profundidad (c) Imagen de profundidad registrada y sobrepuesta sobre la imagen de color

De la mano de OpenNI, Primsense también publicó una librería adicional llamada NITE. Esta librería aunque siendo de código cerrado, también ha sido distribuida bajo la licencia Apache 2.0. NITE ofrece mucha de la funcionalidad necesaria para el desarrollo de interfaces humano-computadora:

- **Detección y seguimiento de manos.**
- **Detección y seguimiento de personas.**
- **Cálculo y seguimiento de articulaciones.**
- **Reconocimiento de gestos preestablecidos.**

Este entorno está escrito completamente en C++ y tiene soporte multiplataforma, estando disponible para windows (x86, x64), linux (x86, x64, arm) y MacOSX (intel).

Es tanto el interés generado sobre este tipo de sensores, que Apple sorprendió a finales de 2013 adquiriendo PimeSense por una suma aproximada de 34 millones de dólares.

Microsoft SDK

Más de un año después de la comercialización de Kinect, Microsoft publica el 16 de junio de 2011 el SDK (Software Development Kit) oficial de uso no comercial para su sensor Kinect. Aunque en octubre de ese mismo año se publicó una nueva versión del mismo SDK para poder desarrollar aplicaciones comerciales. Este SDK sólo está disponible para Windows 7 y Windows 8, ofreciendo prácticamente toda la funcionalidad del conjunto OpenNI y NITE. La principal desventaja de este entorno es que, al menos es sus primeras versiones, el rango de actuación del sensor de profundidad estaba forzado de 0.5 metros a 4 metros (rango donde la detección de las articulaciones incluida funciona de forma precisa). Si el desarrollador necesitaba utilizarlo para capturar distancias mayores, era necesario parchear el código fuente del SDK.

Tras haber analizado los 3 entornos, nos hemos decantado por utilizar OpenNI, en su versión más actual (2.2). Las razones de esta elección son su mantenimiento directo por los propios fabricantes del sensor, su difusión en una amplia comunidad de desarrolladores[7][8], tener soporte para linux (x86 y x64) y no consumir una gran cantidad de recursos en su ejecución.

[7] http://community.openni.org/openni
[8] http://groups.google.com/forum/openni-dev

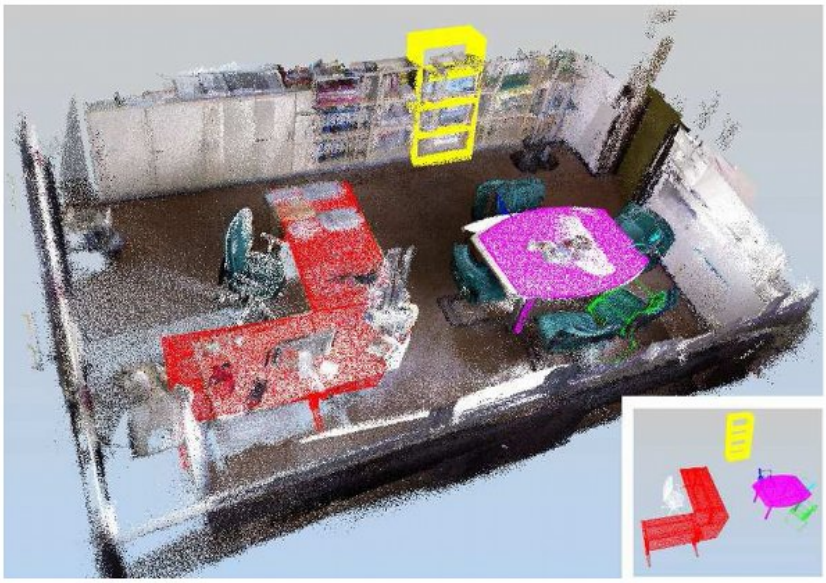


Figura 3.14: Ejemplo de segmentación con Point Cloud Library

3.3. Point Cloud Library

Point Cloud Library (PCL[9]) es una librería escrita en C++ para el tratamiento de imágenes 2D, mapas de profundidad y nubes de puntos. Está publicada bajo una licencia BSD de software libre, lo que permite su uso tanto para aplicaciones comerciales como de investigación. Ha sido desarrollada por un gran consorcio de ingenieros e investigadores de todo el mundo, Willow Garage[10], siendo presentada su primera versión en mayo de 2011.

Esta librería independiente, incluye numerosos algoritmos del estado del arte sobre tratamiento de nube de puntos n-dimensionales y procesamiento geométrico en 3D, incluye además, algoritmos de filtrado, extracción de características, reconstrucción de superficies, registro, ajuste de modelos y segmentación (figura 3.14).

Las aplicaciones principales de esta librería consisten en percepción en robots, filtrado de datos obtenidos con sensores ruidosos, combinación de nubes de puntos en 3D, segmentación de partes relevantes de escenas, extracción de puntos de interés o cálculo de descriptores para crear reconocedores de objetos basados en su geometría. También incluye numerosas funciones para el mallado y reconstrucción de superficies, así como herramientas básicas de visualización.

En este proyecto se han utilizado los tipos de datos disponibles en PCL para la representación de datos como nube de puntos, así como las diversias funciones que incluye para su manejo (combinar varias nubes, eliminar puntos concretos, etc.).

[9] http://pointclouds.org/
[10] http://www.willowgarage.com/

3.4. OpenCV

OpenCV[11] es una librería de visión artificial publicada bajo una licencia BSD de software libre. Esta librería fue originalmente desarrollada por Intel y su primera versión fue publicada en enero de 1999. Está disponible tanto para GNU/Linux, como para Mac OS X y Windows. Al igual que PCL, en la actualidad está siendo mantenida por Willow Garage y su versión mas actual es la 2.4.9, siendo ésta la utilizada durante la realización de este proyecto.

Esta librería, que cuenta con más de 500 funciones, ha sido desarrollada principalmente en C y C++ primando su optimización, aprovechando al máximo la capacidad de los procesadores multinúlceo y la aceleración GPU. Estas funciones están orientadas principalmente al procesamiento básico de imágenes, al reconocimiento de objetos, calibración de cámaras, visión estéreo y visión en robótica.

Otro de los puntos fuertes de esta librería, es su documentación. Tiene un amplia bibliografía, así como un foro propio [12] y una gran comunidad de desarrolladores que la utiliza.

OpenCV se compone de numerosos módulos en los que divide su funcionamiento. De ellos cabe destacar:

- **cxcore**
  Este módulo contiene todos los tipos de datos y estructuras básicas sobre las que se apoya OpenCV.

- **cv**
  Todos los algoritmos principales de visión están contenidos en este módulo.

- **highgui**
  En este módulo se incorporan todas las herramientas para desarrollar interfaces gráficos sencillos directamente con OpenCV, así como todas las funciones que permiten realizar operaciones de entrada/salida para comunicarse directamente con dispositivos de vídeo.

En el presente proyecto no se ha utilizado la parte gráfica de OpenCV, sólo se han utilizado el core como tipo de dato de las imágenes utilizadas en todo el sistema. Los algoritmos de esta biblioteca utilizados son muy básicos siendo utilizados únicamente para realizar cambios en el espacio de colores, en el tamaño de las imágenes y para realizar operaciones con matrices.

[11] http://opencv.org/
[12] http://answers.opencv.org/questions/

3.5. Glade y librerías GTK+

Glade (Glade Interface Designer o Diseñador de Interfaz Glade) es una herramienta de desarrollo visual de interfaces gráficas a través de GTK/GNOME. Este interfaz es totalmente independiente del lenguaje de programación utilizado para el desarrollo del programa. Todo el interfaz es generado en un fichero xml, donde son almacenados los elementos del interfaz. Estos archivos pueden emplearse para construir el interfaz en tiempo de ejecución mediante una biblioteca llamada *libglade*. Un pequeño ejemplo del formato de estos ficheros es el mostrado en el cuadro 3.1 que representa simplemente una ventana con un botón.

```
<interface>
<requires lib="gtk+" version="2.16"/>
<!-- interface-naming-policy project-wide -->

<object class="GtkWindow" id="window1">

<child>

<object class="GtkButton" id="button1">
<property name="label" translatable="yes">button</property>
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="receives_default">True</property>
</object>
</child>
</object>
</interface>
```

Cuadro 3.1: Fichero xml con un interfaz de GTK

Aunque algunas herramientas de Glade sí que permiten la generación directa de código, en la que sólo es necesario rellenar la función de cada uno de los elementos para generar un interfaz interactivo, este sistema está desaconsejado y obsoleto.

GTK+ (The GIMP Toolkit) es un conjunto de bibliotecas multiplataformas que son utilizadas por Glade para la creación de interfaces y está orientado principalmente para los entornos gráficos GNOME, XFCE y ROX. Sin embargo, también se están abriendo paso en el escritorio de Windows, MacOS y otros.

GTK+ está bastante extendido sobre todo en el mundo Linux. Herramientas como Nero-Linux, Firefox o Evolution utilizan esta librería para generar sus interfaces gráficas.

En este proyecto hemos utilizado GTK+ para la creación de todos los interfaces gráficos usados por *JdeRobot* para la interacción con el usuario. Aunque todos estos interfaces se podrían haber escrito directamente en xml, nosotros nos hemos decantado por apoyarnos en Glade-3, un programa con un interfaz gráfico muy intuitivo que simplifica el desarrollo de estos ficheros. GTK+ esta diseñado para programar con C, C++, C#, Java, Ruby, Perl, PHP o Python, por lo que no tendremos ningún problema

Figura 3.15: Captura del juego Rally Master Pro desarrollado con OpenGL.

al utilizar esta librería, en concreto, el API de C++.

3.6. OpenGL

OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que producen gráficos en 2D y 3D. La interfaz estandarizada consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas, a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics [13] en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulaciones de vuelo, estando también muy extendido en el desarrollo de videojuegos (figura 3.15). Muchas de las tarjetas gráficas actuales tienen soporte hardware para esta tecnología, permitiendo agilizar la generación de gráficos 3D directamente desde tarjetas dedicadas.

En este proyecto, se ha utilizado el API de C++ que nos ofrece OpenGL para realizar la representación 3D de los datos obtenidos desde los sensores, así como los resultados generados por nuestros algoritmos de seguimiento.

[13] http://www.sgi.com/products/software/opengl/?/overview.html

3.7. libxml++

libxml++ es el API en C++ de una de las librerías más utilizadas para analizar ficheros xml, *libxml*. Los ficheros xml son muy útiles para almacenar datos de forma legible y que puedan ser modificados por cualquier usuario utilizando un simple editor de texto. En el cuadro 3.2 se puede ver un ejemplo con los datos de un curso de formación.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Primer_Curso>
     <Asignatura_1>
          <Profesor>
               <Nombre>Nombre del profesor</Nombre>
               <Mail> Correo del profesor </Mail>
          </Profesor>
          <Coordinador>
               <Nombre>Nombre del coordinador</Nombre>
               <Mail>Correo del coordinador</Mail>
          </Coordinador>
          <Examenes>
               <Examen_1> Fecha del examen 1 </Examen_1>
               <Examen_2> Fecha del examen 2</Examen_2>
          </Examenes>
     </Asignatura_1>
</Primer_Curso>
```

Cuadro 3.2: Fichero xml de ejemplo

libxml++ implementa los dos tipos de analizadores de xml más comunes:

- **SAX:** este analizador está basado en eventos. Analiza el documento nodo por nodo y si detecta alguna de las etiquetas buscadas ejecuta una acción determinada.
- **DOM:** a diferencia del analizador SAX, DOM guarda el documento xml completo en memoria, generando un árbol que podemos recorrer para buscar los datos que necesite la aplicación.

En nuestro proyecto hemos utilizado esta biblioteca junto con ficheros xml para diferentes fines:

1. Cargar los diferentes parámetros de configuración del sistema.
2. Volcar el estado actual del sistema para poder utilizarlo posteriormente.
3. Transmitir parámetros de configuración de forma remota.

3.8. Progeo

La biblioteca Progeo (Projective Geometry), incluida en el propio *middelware* de *JdeRobot*, proporciona funciones básicas de geometría proyectiva. Esta biblioteca es útil a la hora de relacionar la información visual de una cámara 2D con información espacial en 3D. Esta relación se consigue con dos funciones básicas:

- **Proyectar**
  Esta función (cuadro 3.3) permite obtener el píxel de la imagen en que proyecta un cierto punto en 3D.

```
int project(Eigen::Vector4d in, Eigen::Vector3d &out);
```

Cuadro 3.3: API de la función de proyección

- **Retroproyectar**
  La función de retroproyección (cuadro 3.4) realiza la operación inversa de la proyección. A partir de un píxel 2D concreto de la imagen, permite calcular la recta 3D que intersecta a ese punto en el plano imagen y todos los puntos 3D que proyectan sobre ese pixel, donde se encuentra el punto 3D real.

```
void backproject(Eigen::Vector3d point, Eigen::Vector4d& pro);
```

Cuadro 3.4: API de la función de retroproyección

Así mismo, Progeo también incluye una tercera funcionalidad muy útil a la hora de depurar y representar objetos virtuales en las imágenes 2D.

- **Visualizar línea**
  Esta función auxiliar (cuadro 3.5) permite saber si un determinado segmento en 3D es visible por alguna de las cámaras de nuestro sistema. De la misma manera, es capaz de informarnos si dicho segmento se ve en su totalidad o no.

```
int displayline(Eigen::Vector3d p1, Eigen::Vector3d p2,
 Eigen::Vector3d& a, Eigen::Vector3d& b);
```

Cuadro 3.5: API de la función de visualización de líneas

El modelo proyectivo de Progeo está basado en el modelo PinHole de cámaras [Hartley y Zisserman, 2004] cuyo esquema se puede observar en la figura 3.16. Para que Progeo funcione de forma adecuada es necesario tener los parámetros de calibración de la cámara, tanto los intrínsecos (distancia focal y posición del píxel central) como los extrínsecos (posición de la cámara y orientación definida como foco de atención y *roll*).

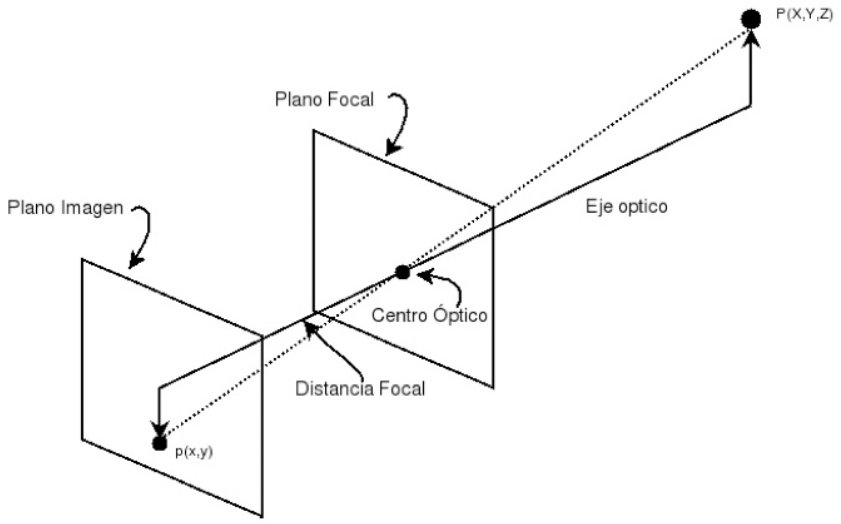


Figura 3.16: Modelo de cámara Pinhole

Capítulo 4
Infraestructura desarrollada para sensores de profundidad

Una vez explicados tanto el contexto del proyecto como los objetivos que se abordan en él, en este capítulo se detallarán los nuevos componentes y funcionalidades que se han desarrollado y añadido a *JdeRobot* como infraestructura útil. Todo ello, con el fin de aumentar las funcionalidades ofrecidas por este entorno para trabajar con sensores RGB-D.

Por un lado, la librería *ParallelIce*, la cual permite realizar peticiones en paralelo a diversos interfaces ICE, otra herramienta para poder grabar y reproducir datos en diferido. Un componente que es capaz de segmentar el primer plano de escenas capturadas con sensores RGB-D, y por último, una segunda librería que incluye diversa funcionalidad muy útil para trabajar con mapas de profundidad.

4.1. Librería ParallelIce

La estructura básica de cualquier componente cliente de *JdeRobot* tiene al menos dos hilos de ejecución, cada uno de ellos con una funcionalidad específica. Un hilo dedicado a la gestión del interfaz gráfico, en caso de que disponga de él y un segundo hilo, encargado de la captura de los datos y del procesamiento y control. Este segundo hilo es el que implementa la propia funcionalidad del componente y envía las órdenes correspondientes.

El hilo principal de captura se encarga de pedir secuencialmente todos los datos a los servidores correspondientes y alojar estos datos en memoria compartida, debidamente protegida frente a acceso concurrente. Esta arquitectura secuencial es idónea, siempre y cuando el acceso a los datos no suponga un tiempo significativo, sobre todo cuando se estén utilizando varias fuentes de datos. Utilizando un ejemplo gráfico, supongamos un escenario donde hay cuatro cámaras y el acceso a cada cámara supone 30 milisegundos. Al acceder a los datos de forma secuencial, se produce un desfase de 30 milisegundos entre cada adquisición, llegando a ser de 90 milisegundos entre la captura de la primera cámara y la cuarta (figura 4.1). Por ello, este tipo de paradigma de petición secuencial no es lo suficientemente eficiente para sistemas que necesitan un mínimo de sincronización entre las capturas de todos los sensores.

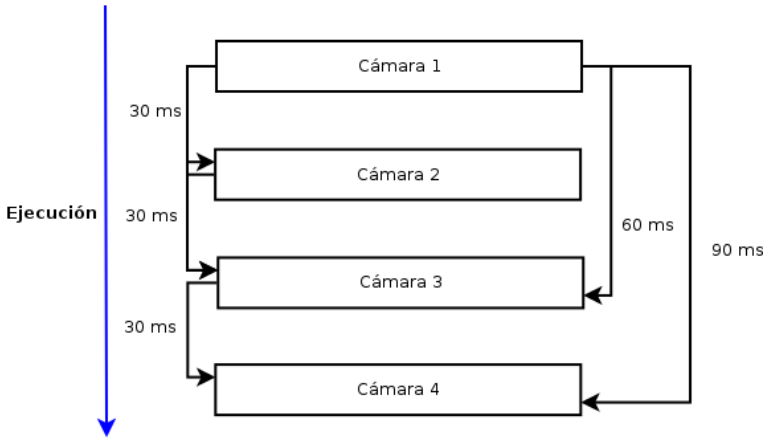


Figura 4.1: Desfases en imágenes en capturas secuenciales

Uno de los problemas que presenta el uso de ICE es la latencia que se suma al problema comentado anteriormente. Cada vez que un componente necesita nuevos datos de un servidor, el cliente envía una petición al servidor y éste le contesta con los nuevos datos. Este tipo de comunicación bajo demanda, se denomina Llamada a Procedimiento Remoto o RPC (Remote Procedure Call), donde es el cliente quien toma siempre la iniciativa de las comunicaciones y procesos. Como se refleja en la figura 4.2, esta comunicación genera cierta latencia desde que el cliente hace la petición de nuevos datos hasta que los recibe. Si estas peticiones se realizan directamente en el propio hilo de procesamiento y control, nuestro componente estará cierto tiempo sin poder hacer nada, bloqueado, esperando a que le lleguen nuevos datos. Esto no supone un gran problema cuando se trabaja con muy pocos servidores o con servidores que ofrecen datos de muy poco tamaño, pero en casos como el del presente proyecto, donde se tienen que realizar 3 peticiones (cámara de color, mapa de profundidad y nube de puntos) a cada uno de los servidores conectados a los dispositivos físicos, supone un gran desperdicio en tiempo de ejecución en bloqueos. Este fenómeno de latencia es especialmente apreciable en interfaces que sirven datos voluminosos como puede ser cámaras o nubes de puntos.

Para dar solución a estos dos problemas, la petición secuencial y la latencia, se ha creado una librería que permite un acceso en paralelo a cada sensor que utiliza varias hebras concurrentes, haciendo peticiones sensoriales periódicas. Esto permite que el cliente pida directamente la última copia local que haya adquirido, alojada en memoria compartida, reduciendo así la desincronización entre los datos y la latencia durante la captura. Esta librería, llamada *ParallelIce*, crea un objeto con un hilo dedicado que se conecta a un servidor y le pide datos a una frecuencia determinada, configurable por el usuario, capturando datos de todos los sensores de forma simultánea. De esta forma, se produce un *polling*, que realiza consultas constantes al servidor, con el fin de tener preparado un dato actualizado, el cual esté alojado en memoria local de nuestro componente (disponible de forma instantánea). De esta manera está disponible un dato de forma directa para cuando cualquier parte de nuestro componente, como el hilo

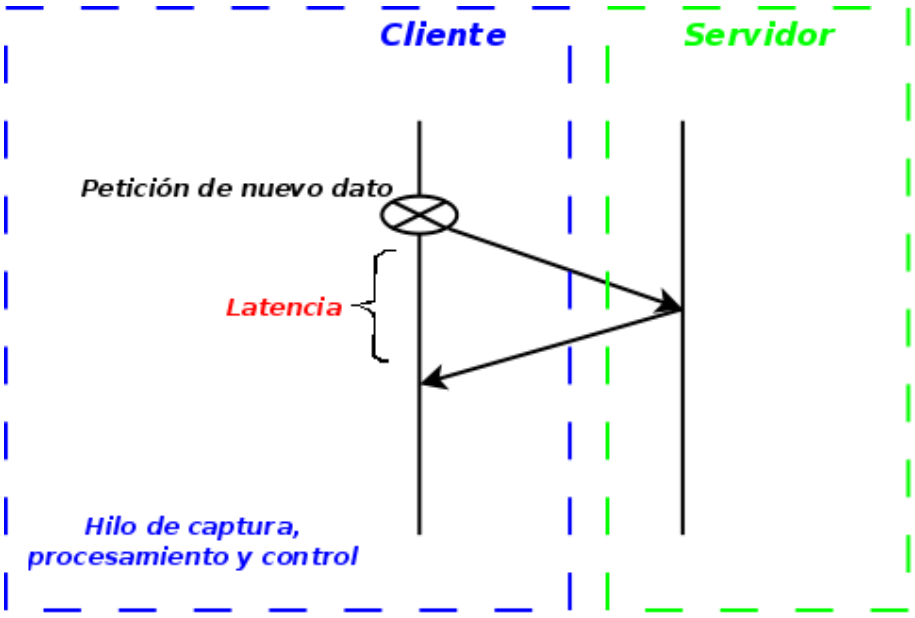


Figura 4.2: Latencia por RPC

de procesamiento y control o el interfaz gráfico, lo necesiten. De esta manera (figura 4.3), se reduce el retardo de acceso total a los nuevos datos, y se consigue una mayor sincronización entre los diferentes sensores, con un desfase pequeño entre los datos del orden del tiempo de captura de un sensor.

Utilizando esta librería se elimina el hilo de captura común, ya que cada cliente tiene el suyo propio, con su propia gestión de memoria compartida, por lo que cualquier hilo de la aplicación le puede pedir nuevos datos con la seguridad de no haber condiciones de carrera, ni que suponga tiempo adicional alguno. Por lo tanto, se pasa de una estructura con dos hebras, una para la captura, el procesamiento y control, y otra para el interfaz gráfico; a otra con 2+n, una para procesamiento y control, y otra para el interfaz gráfico y otras n hebras más, en función del número de servidores al que se conecte nuestro dispositivo (figura 4.4).

Esta librería no sólo soporta interfaces de cámaras, también se ha implementado una versión para el acceso en paralelo a nubes de puntos de sensores 3D y a sensores láser.

4.2. Reproducción enlatada

En muchas plataformas de desarrollo existen un conjunto de herramientas vitales para la validación de sistemas y comparación de resultados obtenidos con diferentes algoritmos. Estas herramientas deben permitir dos funcionalidades básicas. La primera es que debe ser capaz de recopilar datos sensoriales de diferentes dispositivos para volcarlos a disco. Y la segunda funcionalidad consiste en permitir emular los sensores

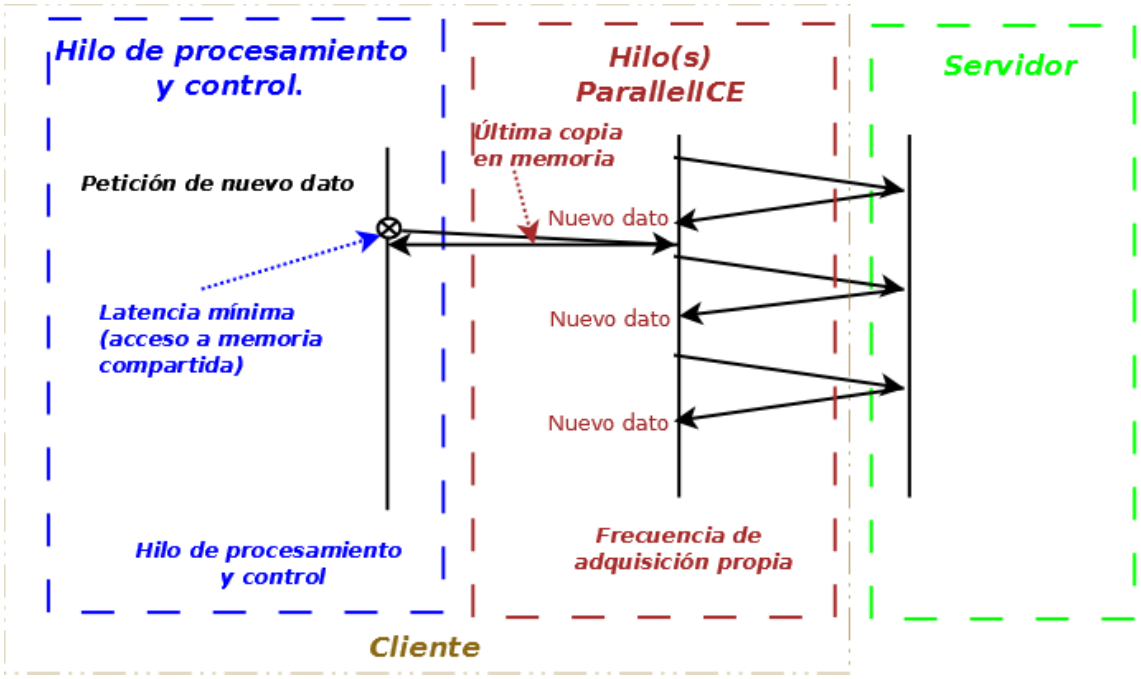


Figura 4.3: Latencia con ParallelIce

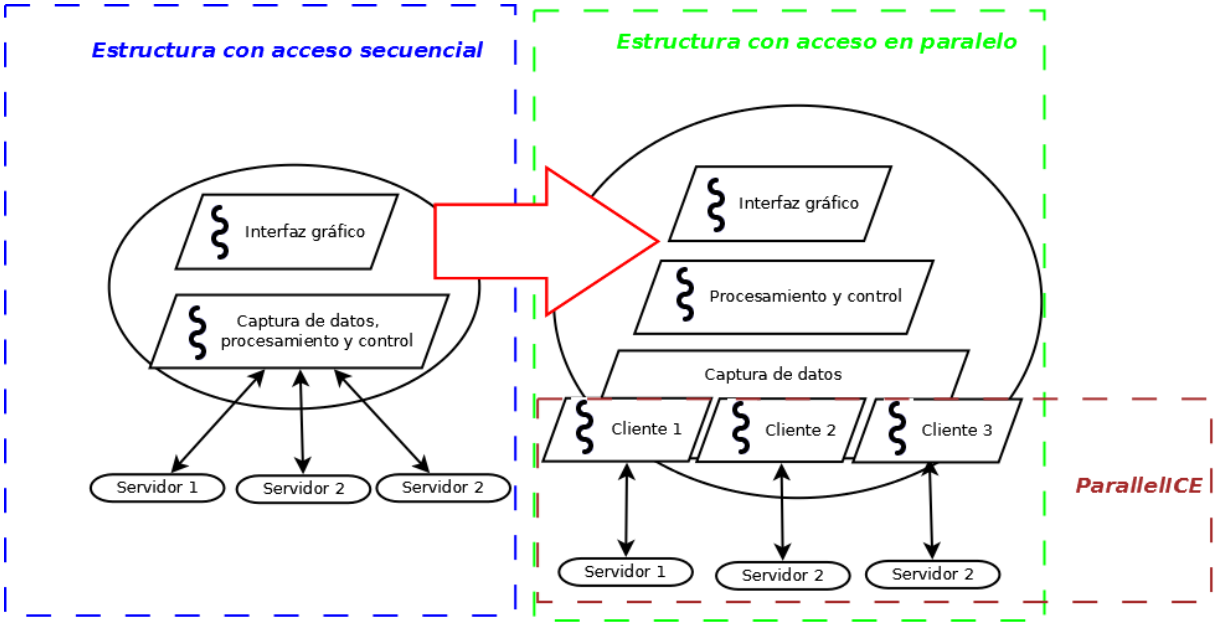


Figura 4.4: Estructura interna del modelo de acceso secuencial y el modelo de acceso en paralelo.

originales, sirviendo esos datos guardados en disco. Esta herramienta ofrece la ventaja de probar con exactamente los mismos datos distintas versiones de los mismos algoritmos, o bien comparar resultados entre diferentes técnicas. Grandes plataformas como ROS tienen sistemas similares al planteado en esta sección, integrado dentro de ROS bags[1], el cual, entre otras cosas, se encarga de grabar datos para poder reproducirlos en diferido. Con el fin de dar soporte a este tipo de herramientas de reproducción enlatada en *JdeRobot* se han desarrollado dos aplicaciones: *Recorder* y *Replayer*

Uno de los principales requisitos para este tipo de aplicaciones es que los datos estén bien sincronizados y que no existan cambios entre una reproducción y otra. Al igual que sucede con el problema al que se ha dado solución con *ParallelIce*, no se puede hacer una captura secuencial con la marca de tiempo de cada iteración, ya que como sucedía en la figura 4.1, cada imagen estaría desfasada con respecto al resto y además, no coincidiría con la marca de tiempo a la que está asociada. Para ello, se ha implementado una metodología de acceso en paralelo a cada uno de los interfaces y a la vez, un sistema de productores/consumidores para volcar los datos obtenidos a disco.

4.2.1. Componente Recorder

El componente *Recorder* aborda la problemática de la sincronización con el mismo paradigma de *ParallelIce*, el acceso paralelo a cada interfaz con módulos que se ejecutan de manera concurrente. De esta forma, cada interfaz ICE tiene asociado un módulo, y éste se encarga de obtener nuevos datos, hacer un volcado a disco y guardar una marca de tiempo asociada él. Esta marca de tiempo se toma con respecto al inicio de la grabación. Con este diseño se consigue un punto común permitiendo que cada módulo pueda controlar independientemente sus marcas de tiempo (figura 4.7). Cada uno de estos módulos concurrente, por tanto, tiene que lidiar con la latencia de la captura de datos y la latencia del acceso a disco. Con el fin de desacoplar estos tiempos de acceso se ha dividido esta funcionalidad, tal y como refleja la figura 4.5. La frecuencia máxima a la que puede adquirir datos un módulo de *Recorder*, siguiendo esta estructura, depende directamente de la latencia que tiene el sistema en capturar datos del servidor. Con el fin de mejorar esta frecuencia máxima, se han diseñado estos módulos siguiendo una arquitectura productor/consumidor. Con este nuevo diseño, la captura de datos será realizada por los productores y el volcado a disco por los consumidores. Como podemos ver en la figura 4.6, donde se ha representado un ejemplo con dos productores y dos consumidores, siendo tiempo de latencia de acceso al servidor el mismo, se obtienen el doble de datos en la misma franja de tiempo, siempre y cuando los productores estén debidamente sincronizados. Esto se debe a que, con este paradigma de acceso concurrente, el tiempo de adquisición no es directamente el de una petición secuencial, sino el tiempo entre las capturas de los diferentes productores.

Al hacer un volcado de los datos directamente a disco, se genera un fichero para cada una de las capturas realizadas de forma que, para recuperar los datos, sólo es necesario

[1] http://wiki.ros.org/Bags

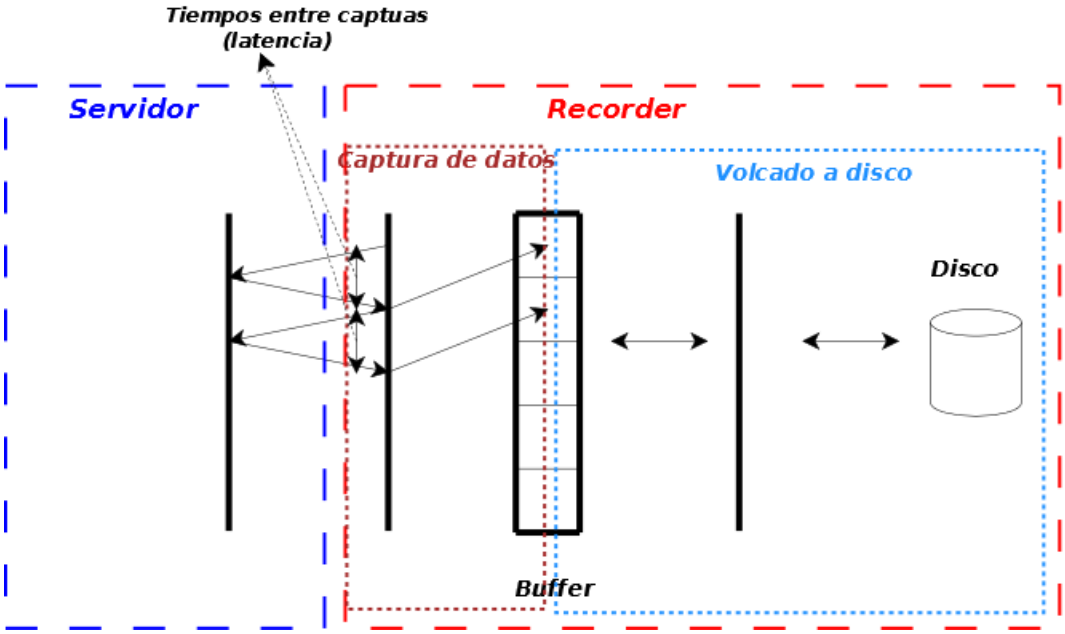


Figura 4.5: Estructura interna de un módulo del componente Recorder

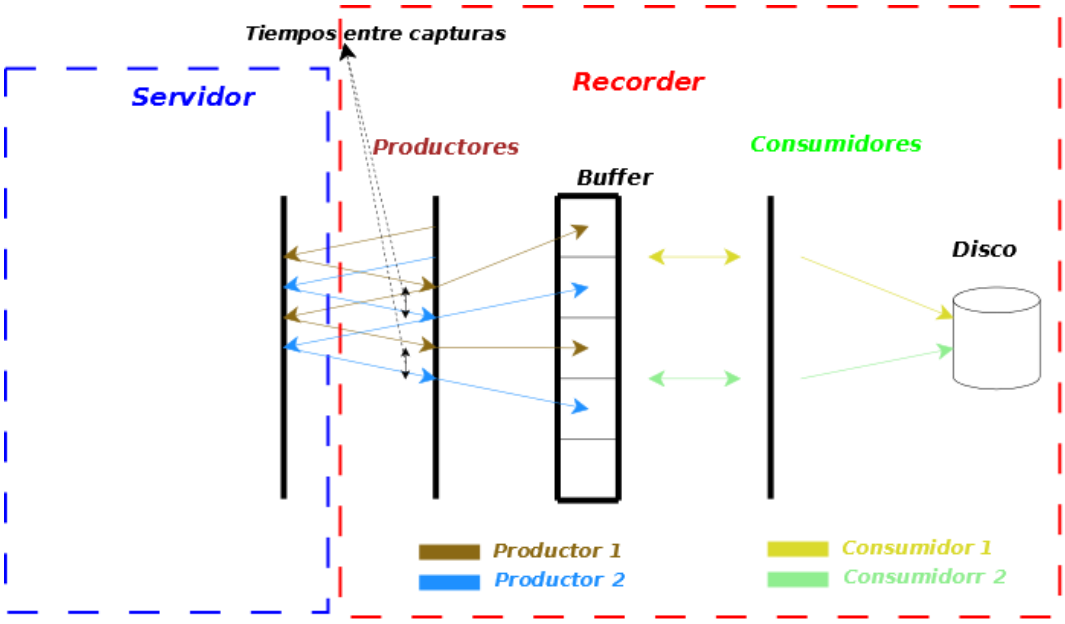


Figura 4.6: Estructura interna de un módulo del componente Recorder con productores/consumidores

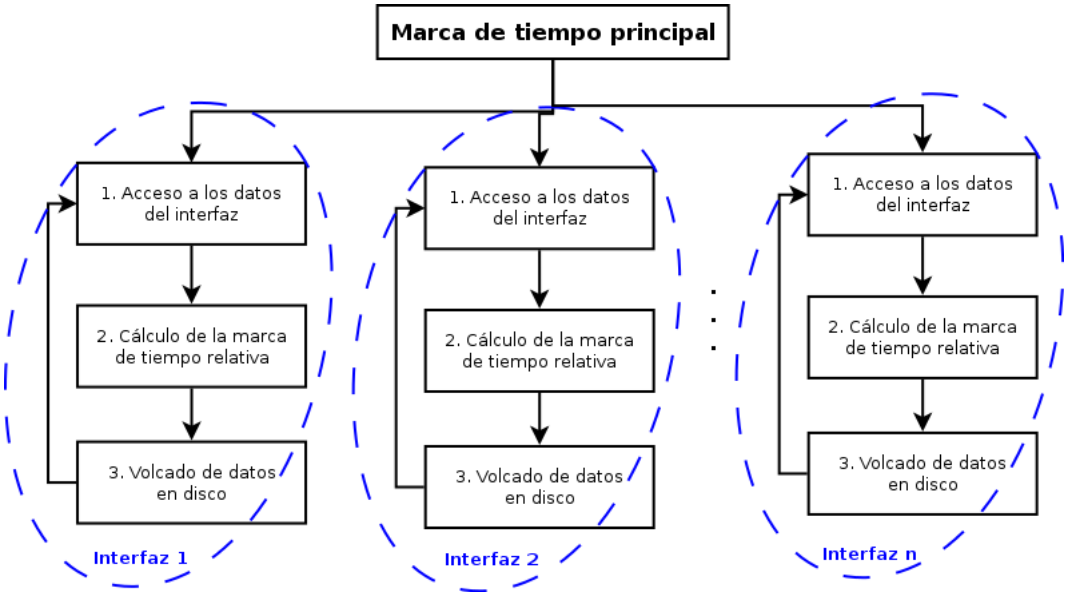


Figura 4.7: Ejecución del componente Recorder

hacer un volcado inverso, es decir, volcando los datos de disco a memoria. Igualmente, se genera un fichero de control por cada interfaz al que está conectado *Recorder*, en el que se guardan las marcas de tiempo relativas en las que se ha producido un volcado de información a disco, es decir, cuándo se ha recibido el dato y se ha guardado.

En el cuadro 4.1, se puede apreciar un ejemplo del fichero de control. Cada una de las líneas de este fichero lleva asociado un archivo, con la misma numeración que la marca de tiempo concreta (con el fin minimizar inconsistencias), donde están los datos reales. En la figura 4.8, se puede ver un ejemplo de los ficheros que se generarían al lanzar *Recorder* sobre dos interfaces de cámara, dos de nubes de puntos y otros dos de láser.

```
51
120
189
250
324
390
451
518
587
651
720
790
852
```

Cuadro 4.1: Fichero fuente de Recorder

La única configuración que necesita este componente para ejecutarse de forma correcta es, por un lado, un fichero de configuración ICE donde se definan todos los interfaces que se van a registrar (cuadro 4.2) y por otro lado, el propio *Recorder*, que se encarga de abrir todas las conexiones necesarias y generar el árbol de directorios necesario para realizar los volcados a disco de forma adecuada. *Recorder* no sólo permite

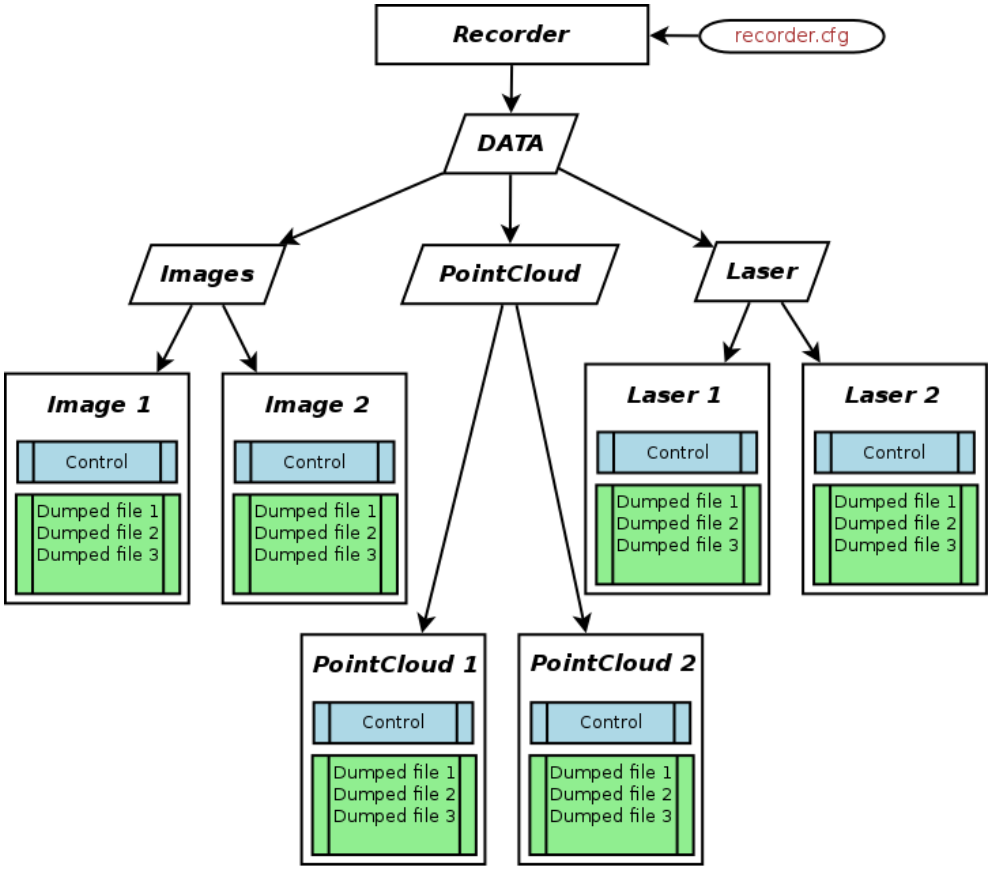


Figura 4.8: Estructura de ficheros generada por el componente Recorder

guardar datos sensoriales, también datos de control enviados a los actuadores. De esta forma, se pueden contrastar los resultados generados por algoritmos de control en diferentes ejecuciones o comprobar las mejoras aplicadas en dichos algoritmos.

```
Recorder.Control=data.jde
Recorder.nCameras=4
Recorder.nDethSensors=2
Recorder.nLasers=1
Recorder.PointCloud1.Proxy=pointcloud1:tcp -h 127.0.0.1 -p 9998
Recorder.PointCloud2.Proxy=pointcloud1:tcp -h 127.0.0.1 -p 9998
Recorder.Laser1.Proxy=Laser:tcp -h localhost -p 9996
Recorder.Camera1.Proxy=cameraA:tcp -h localhost -p 9999
Recorder.Camera2.Proxy=cameraB:tcp -h localhost -p 9999
Recorder.Camera3.Proxy=cameraA:tcp -h localhost -p 9998
Recorder.Camera4.Proxy=cameraB:tcp -h localhost -p 9998
```

Cuadro 4.2: Fichero de configuración de Recorder

4.2.2. Componente Replayer

Este componente, que realiza las veces de reproductor de modo temporizado de los datos enlatados, recibe el nombre de *Replayer*. Es capaz de reproducir cualquier *log* guardado con *Recorder*, utilizando un hilo de ejecución para cada interfaz donde el propio hilo se encarga de cargar los datos de disco a memoria y controlar su sincronización. De esta forma, entrega los datos replicando la temporización con la que fueron grabados. La sincronización es relativa, al igual que se hace en la grabación de los datos, donde la aplicación tiene una marca de tiempo global y cada hilo sirve los datos que correspondan al instante de tiempo correspondiente (figura 4.9). *Replayer* ofrece la posibilidad de activar o desactivar los diferentes interfaces grabados con *Recorder*. Con esto se permiten grabar logs de todos los sensores disponibles y reproducir sólo aquel subconjunto que sea realmente necesario, optimizando así el uso de la capacidad de cómputo del ordenador.

4.2.3. Componente ReplayController

Una herramienta complementaria al componente *Replayer*, que permite interactuar con la reproducción de datos enlatados es el componente *ReplayController*. Mediante un interfaz ICE dedicado se pueden controlar una serie de parámetros comunes a los sistemas de reproducción convencionales (figura 4.11):

- Pausar y reanudar. Permite parar en un instante de tiempo y servir los datos de ese instante hasta que se reanude la reproducción. Esta funcionalidad es muy útil para comprobar la estabilidad y evolución de algoritmos, ya que son datos completamente invariantes.

- Avance lento. Utilizando esta funcionalidad podemos avanzar en plazos de tiempo determinados desde un estado pausado, por ejemplo de 20 milisegundos en 20 milisegundos.

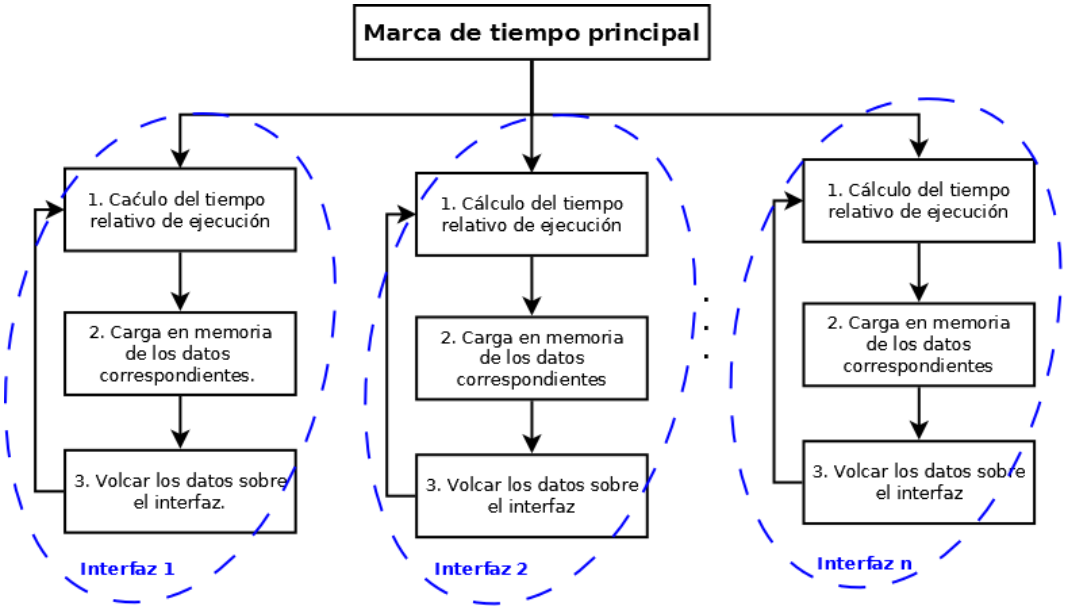


Figura 4.9: Ejecución del componente Replayer

- Línea temporal. Esto permite avanzar directamente hasta un punto determinado del log guardado, servir los datos de ese instante o reproducir los datos desde ese instante.
- Repetición. Con esta opción se puede fijar que la reproducción del log se realice en bucle, cuando un log llega al final de los datos se vuelve a repetir desde el principio.

Estas opciones se pueden controlar utilizando su interfaz gráfico integrado basado en Gtk (figura 4.12), siendo éste completamente opcional.

Este componente utiliza interfaces estándares de ICE con las llamadas básicas que implementan las funciones de control y de reproducción. Este componente se puede utilizar para cualquier otro componente que implemente funciones de reproducción, no solamente para el componente *Replayer*.

4.3. Componente BackgroundSubstractor

En numerosos escenarios, dentro de la visión artificial, se utilizan herramientas de aprendizaje de fondo para enfocar el procesamiento en la imagen de primer plano. Con el fin de introducir esta funcionalidad dentro del entorno *JdeRobot* para sensores tipo Kinect, se ha desarrollado el componente *BackgroundSubstractor*. Este componente se conecta a un sensor RGB-D (a través de *OpenNIServer*) y aplica técnicas de aprendizaje de fondo sobre el mapa de profundidad para obtener la escena del primer plano. Una vez hecho esto, es capaz de servir los datos de primer plano, bien como imágenes de color y profundidad segmentadas, o bien la nube de puntos correspondiente (figura 4.13).

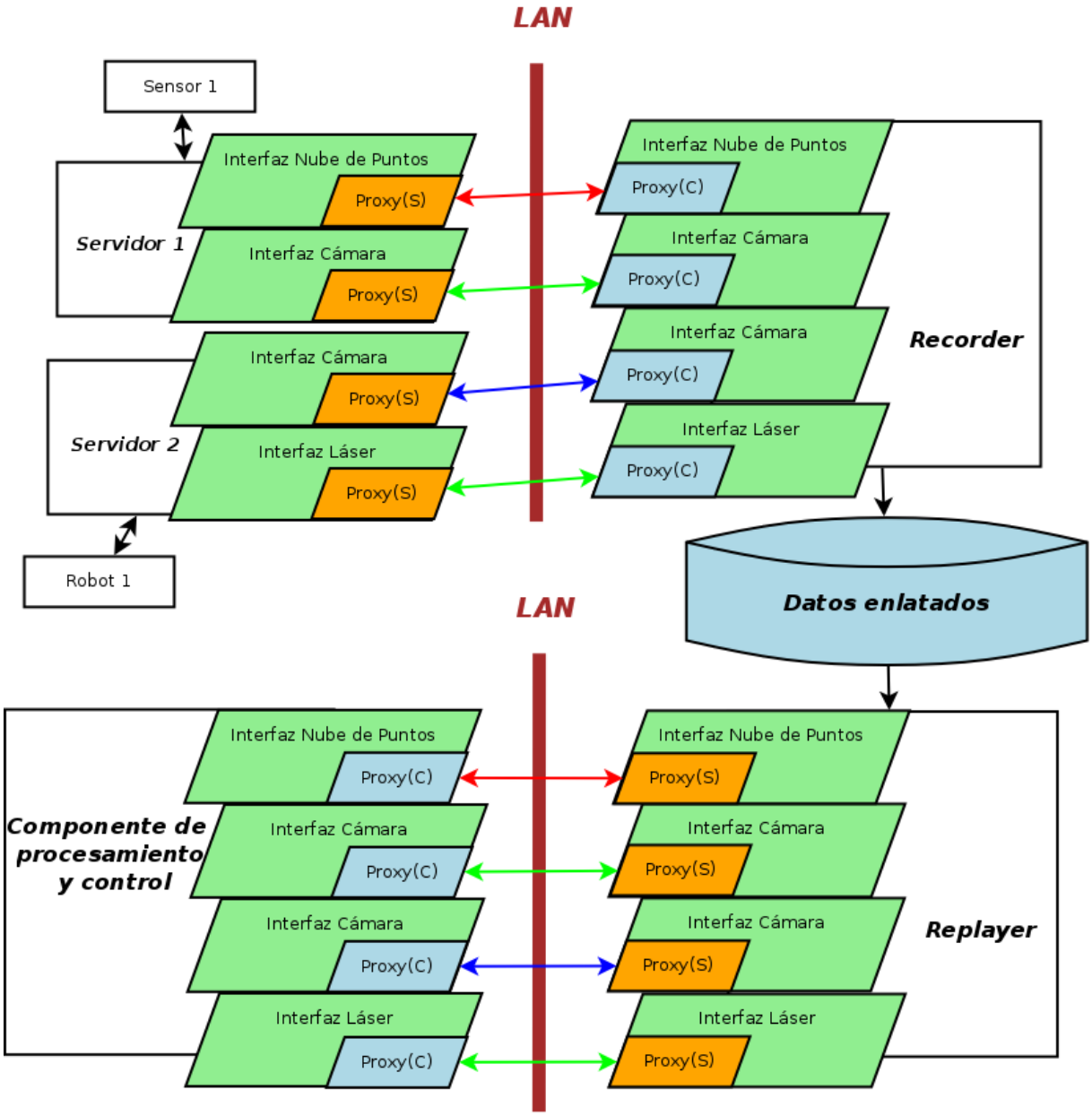


Figura 4.10: Esquema de conexión Recorder-Replayer. (S) Servidor (C) Cliente

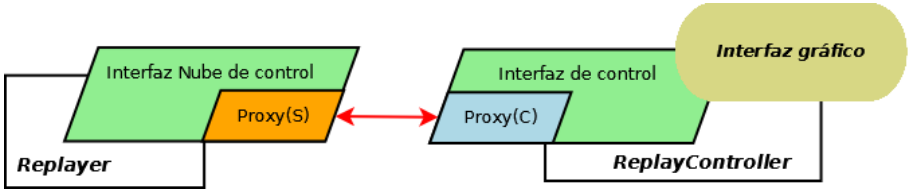


Figura 4.11: Conexión del componente Replayer con el componente ReplayController. (S) Servidor (C) Cliente

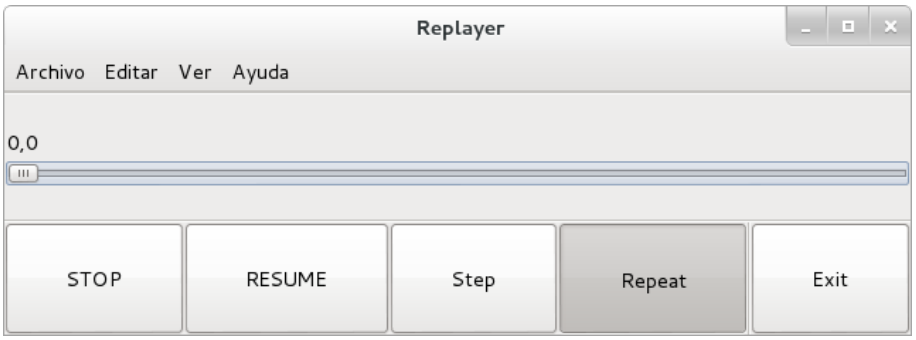


Figura 4.12: Interfaz gráfico del componente replayController

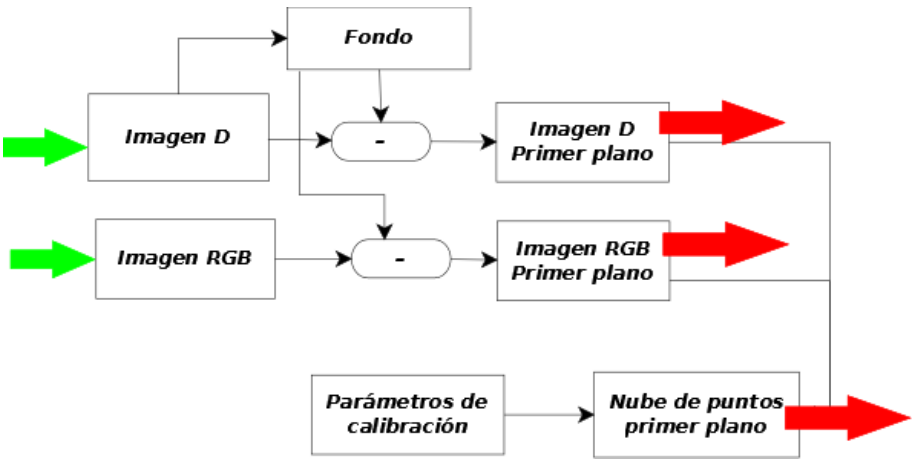


Figura 4.13: Esquema del proceso de segmentación de primer plano

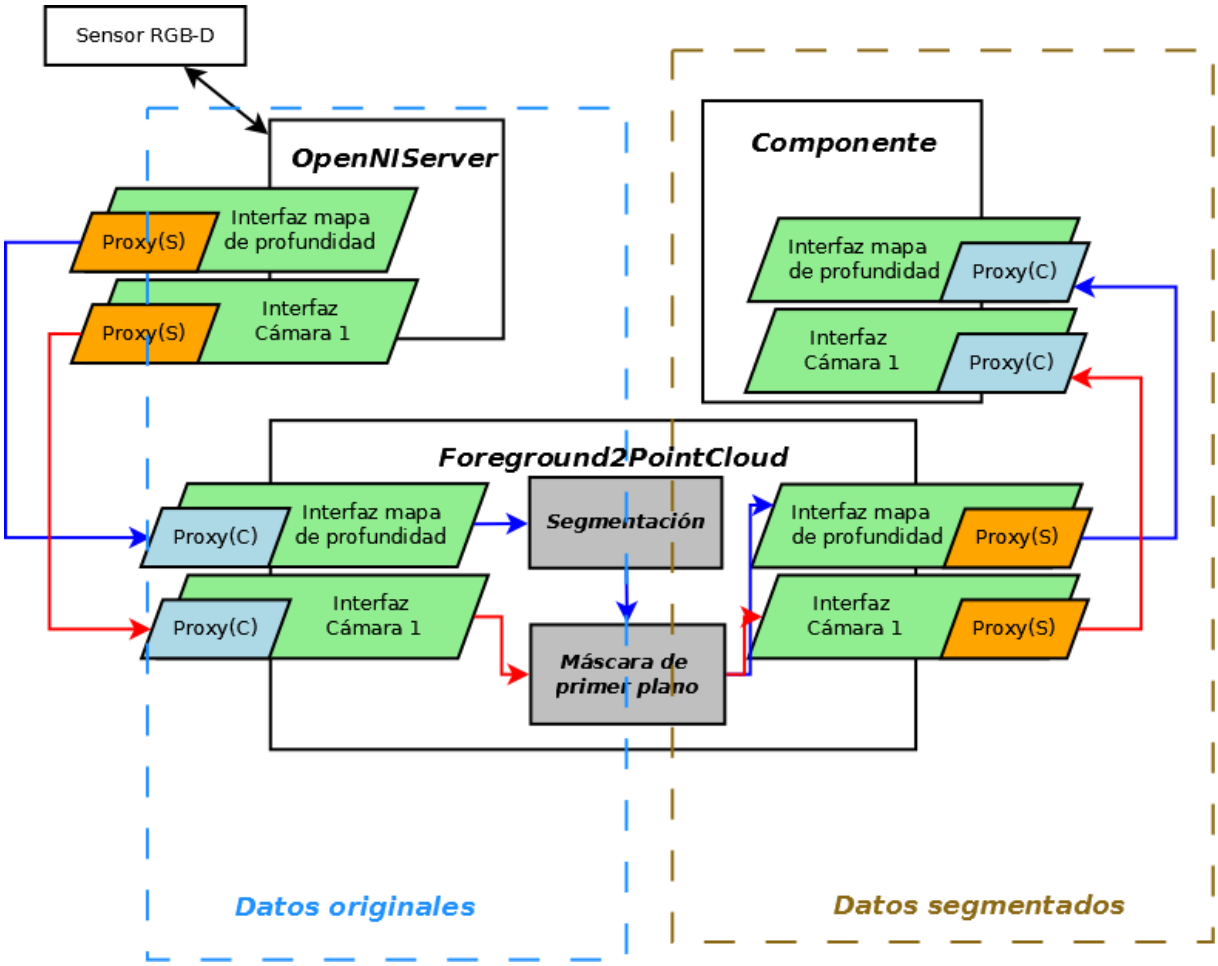


Figura 4.14: Conexión de componentes utilizando el segmentador basado en mapas de profundidad. (C)Cliente (S)Servidor

Para cumplir con esta funcionalidad es necesario enfrentarse a varios aspectos intrínsecos al sensor utilizado para este proyecto. Los sensores RGB-D convencionales tiene un rango útil desde los 0.5 hasta los 10 metros aproximadamente. El ruido producido por el dispositivo es mayor cuanto más alejado está el objeto captado, por ello, es necesario aplicar algún procesamiento para discretizar los datos del mapa de profundidad, con el fin de ocultar el ruido. Sin aplicar este proceso en distancias largas, el propio ruido haría saltar de un escalón a otro.

Con el fin de englobar otras posibles aplicaciones de este componente, es posible realizar ciertas configuraciones adicionales como habilitar o deshabilitar el servicio de nube de puntos, el cual en algunas aplicaciones no es necesario, siendo el procesamiento directamente sobre imágenes segmentadas. Por ello, este componente es capaz de servir de forma independiente las imágenes segmentadas obtenidas de los diferentes interfaces a los que esté conectado (4.14).

4.3.1. Discretización no lineal de la profundidad

Con el fin de aplicar técnicas convencionales de aprendizaje de fondo en imágenes y reducir la complejidad de implementar técnicas nuevas, es conveniente aplicar algún tipo de preprocesado sobre el mapa de profundidad, con el fin de compensar el ruido generado por el sensor. Este fenómeno, inherente en los mapas de profundidad registrados con los sensores RGB-D de consumo, es mayor cuanto mayor sea la distancia de los objetos captados. Por ello, el umbral utilizado para determinar qué está en primer plano o no, depende de la distancia del objeto al sensor.

Para solucionar este problema se ha implementado, por una parte, un cambio en la linealidad de los datos ofrecidos por el sensor, y por otra parte, se ha aplicado una discretización de los datos procesados. De esta forma, un objeto será tomado como primer plano con pequeños cambios cuando está cerca del sensor, pero si está alejado estos cambios deben ser mayores para discernirlos de simple ruido. La conversión se ha realizado aplicando la siguiente fórmula:

$$d' = d^{\frac{1}{4}} \cdot 1000;$$

En la figura 4.15, se puede observar la función lineal inicial y el resultado tras aplicar la conversión anterior. En el ejemplo de esta figura se ha utilizado una discretización de 30 campos, como se puede comprobar en la función inicial, el rango es constante de los 0 a los 10 metros; sin embargo, en la función modificada, el rango es mucho menor cuando el objeto está cerca que cuando éste esta muy alejado. En la aplicación final se ha utilizado una discretización de 256 campos, ya que se han utilizando imágenes en escala de grises utilizando como tipo de datos *char* sin signo.

4.3.2. Aprendizaje del fondo y cálculo de primer plano

Una primera aproximación para reducir la información a procesar, consiste en aplicar técnicas de segmentación de primer plano. Para ello, es necesario crear un modelo del entorno sin ninguna persona presente, pasando a formar parte de la imagen de primer plano aquellas zonas u objetos que no se ajusten a dicho modelo. Al estar trabajando con mapas de profundidad, se puede tomar como modelo una imagen estática del entorno vacío (figura 4.16(a)) y aplicar técnicas sencillas de segmentación de primer plano basado en diferencia de imágenes con un umbral. Esta técnica la podemos aplicar con un umbral común a toda la imagen debido al preprocesado que se le aplica, detallado en el apartado anterior. Este modelo simplificado implica un ahorro de consumo de procesamiento, aunque también se podrían aplicar técnicas de aprendizaje estadístico, como mezcla de gaussianas o aprendizaje exponencial. Adicionalmente, se puede inicializar o resetar la captura base del segmentador desde una imagen guardada en disco, de esta manera, es posible controlar el estado del segmentador de forma remota, e incluso lanzar el componente sin interfaz gráfico.

Una vez calculada la imagen de primer plano se le aplica un filtrado para descartar zonas segmentadas demasiado pequeñas. Este proceso se realiza calculando el área dentro de un contorno de componentes conectados. Para ello, el primer paso es

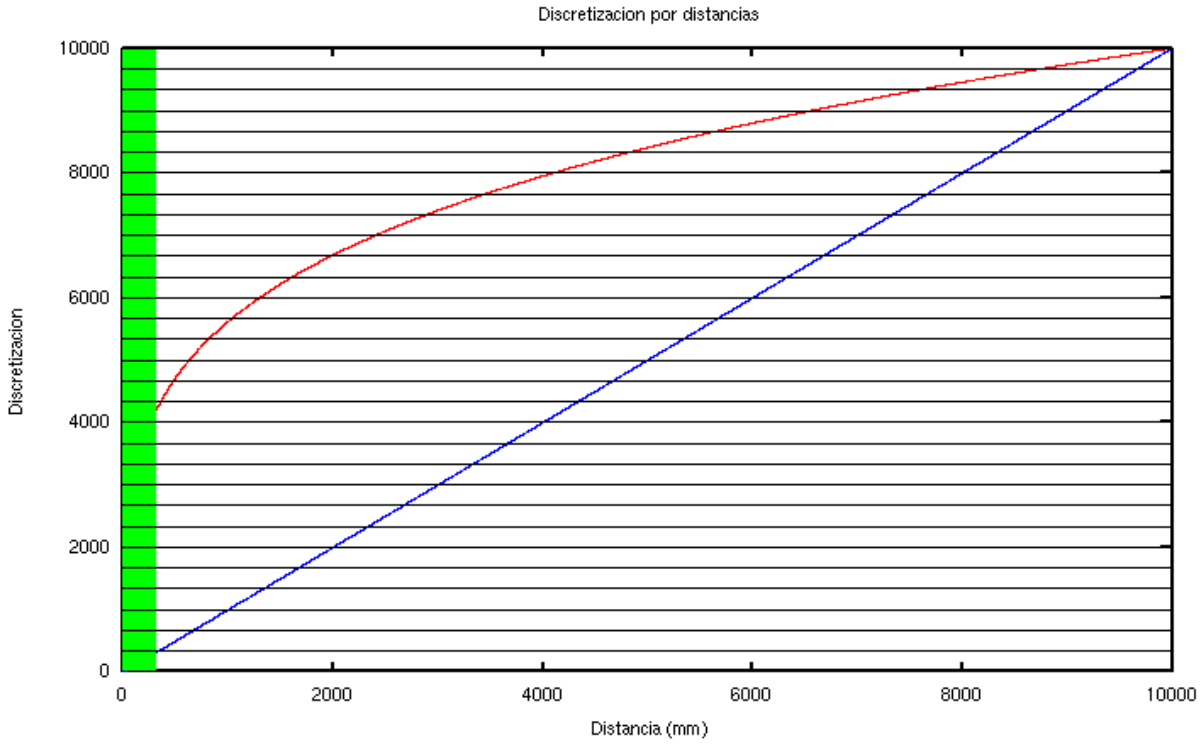
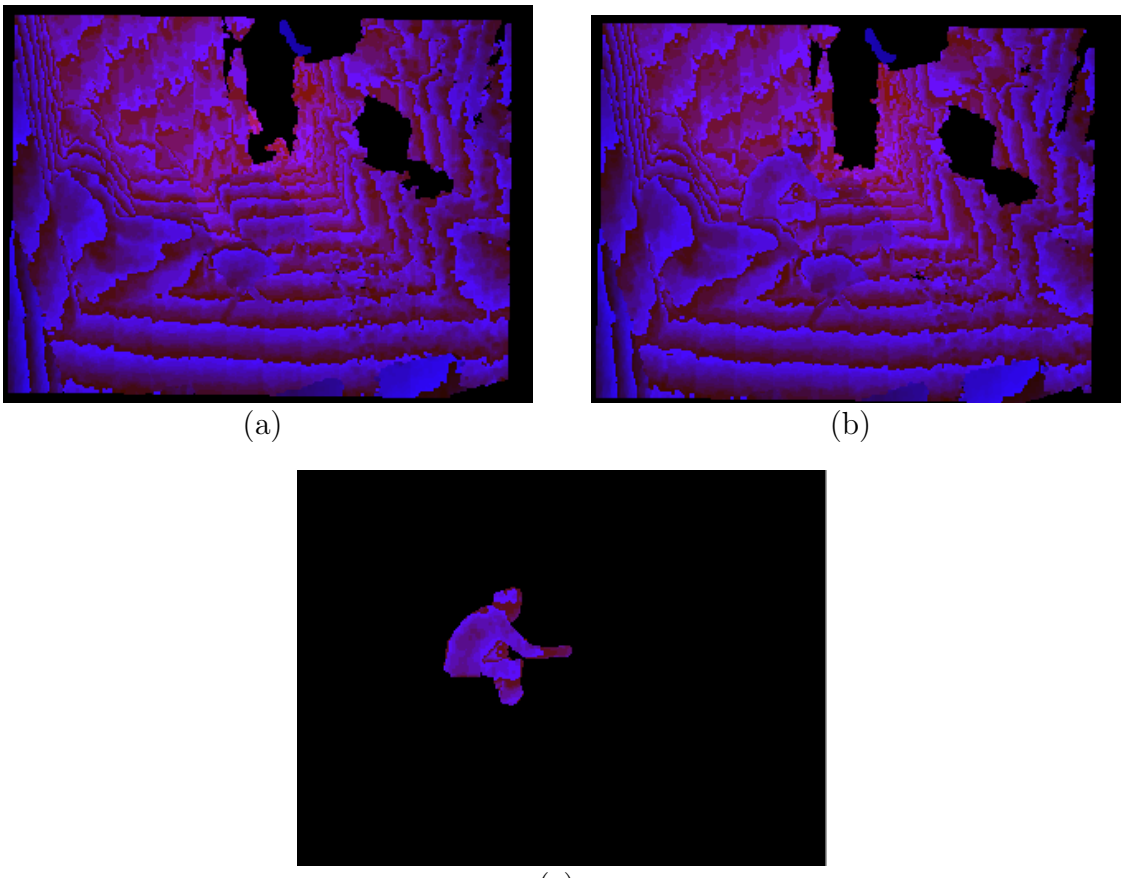


Figura 4.15: Conversión y discretización de la distancia

binarizar la imagen de primer plano y aplicar operaciones morfológicas con el fin de simplificar la imagen eliminando zonas irrelevantes. La operación morfológica que se ajusta a los requerimientos presentados es la operación de *cierre*, dilatación más erosión ($(X \oplus B) \oplus B$), que además, es capaz de conectar objetos próximos entre sí, rellenar huecos y suavizar contornos. Una vez pasado este primer filtrado, se calcula el contorno de todas las las zonas segmentadas, aplicando un algoritmo propuesto por [Suzuki y Be, 1985], el cual esta incluido dentro de la biblioteca OpenCV (función *cv::findContours*). Se computa el área determinado por cada uno de los contornos, usando la fórmula de Green [Kaplan, 1991]. Esto permite descartar aquellas que sean menor que un cierto tamaño (120-400). En general, este tamaño se fija en función del ruido captado por los sensores en cada entorno (dependiendo de la incidencia del sol en la escena), siendo bajo en entornos con poco ruido y, en caso contrario, se fijará el valor mínimo sobre el cual no se obtienen segmentaciones de primer plano captando el escenario inmóvil. Para el cálculo de áreas también se ha utilizado OpenCV, que incorpora la citada técnica en la función *cv::contourArea*. Por lo tanto, el primer plano estará formado por aquellas zonas del mapa de profundidad que hayan cumplido todas las condiciones anteriores (figura 4.16(c)).

4.3.3. Conversión a nube de puntos

Como se ha explicado en la introducción, los sensores RGB-D generalmente ofrecen la información en dos imágenes; una para representar la imagen de color convencional y una segunda, donde se representa la distancia que corresponde a cada uno de los



(a) (b)

(c)

Figura 4.16: (a) Escenario vacío (b) Datos sin segmentar (c) Datos segmentados

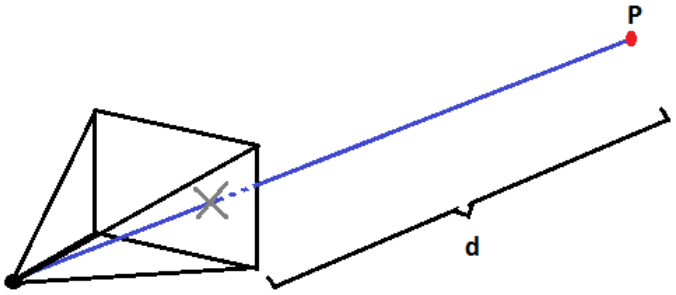


Figura 4.17: Cálculo del punto 3D con la información de distancia

píxeles de la imagen de color (imagen D).

Para representar esta información conjunta en 3D, se utilizó la biblioteca de geometría proyectiva de *JdeRobot*, *Progeo*, y su modelo inHole de proyección (apartado 3.8). La ventaja de trabajar directamente con medidas de distancia es la posibilidad de acotar la recta de retroproyección de los píxeles a un solo punto tridimensional. El punto 3D asociado a cada píxel es aquel que se encuentre en su rayo de retroproyección a la distancia (d) asociada a dicho píxel (punto P, en la figura 4.17). *Progeo* entrega un punto 3D que unido con el foco de la cámara forma el rayo de retroproyección en el cual están todos los puntos 3D que proyectan en ese píxel.

La asociación de un píxel de color con su correspondiente valor de distancia no es un proceso trivial. Cada cámara, la de color y la de infrarrojos, está localizada en una ubicación diferente por lo que no ven exactamente lo mismo. Para que esta asociación sea directa, el proceso más rápido y sencillo que se puede llevar a cabo es registrar las dos imágenes. De esta forma, para obtener el valor de la distancia de un píxel de la imagen de color, sólo sería necesario obtener el valor del píxel homólogo, es decir, el píxel en esa misma posición en la imagen de distancia. Además de estar registradas ambas imágenes, deben estar bien sincronizadas para asegurarnos de que ambas capturas están tomadas en el mismo instante de tiempo y que una corresponde directamente con la otra. Afortunadamente el API de OpenNI nos ofrece ambas configuraciones mediante las funciones reflejadas en el código 4.3. Esta funcionalidad está incluida en el componente *OpenNIServer*, encargándose de hacer corresponder las dos imágenes (RGB y D) que entrega.

```
//RGB and DEPTH registration
setImageRegistrationMode( openni::IMAGE_REGISTRATION_DEPTH_TO_COLOR )

//RGB and DEPTH syncronization
m_device.setDepthColorSyncEnabled(true);
```

Cuadro 4.3: Funciones ofrecidas por OpenNI para el registro y sincronización de imágenes

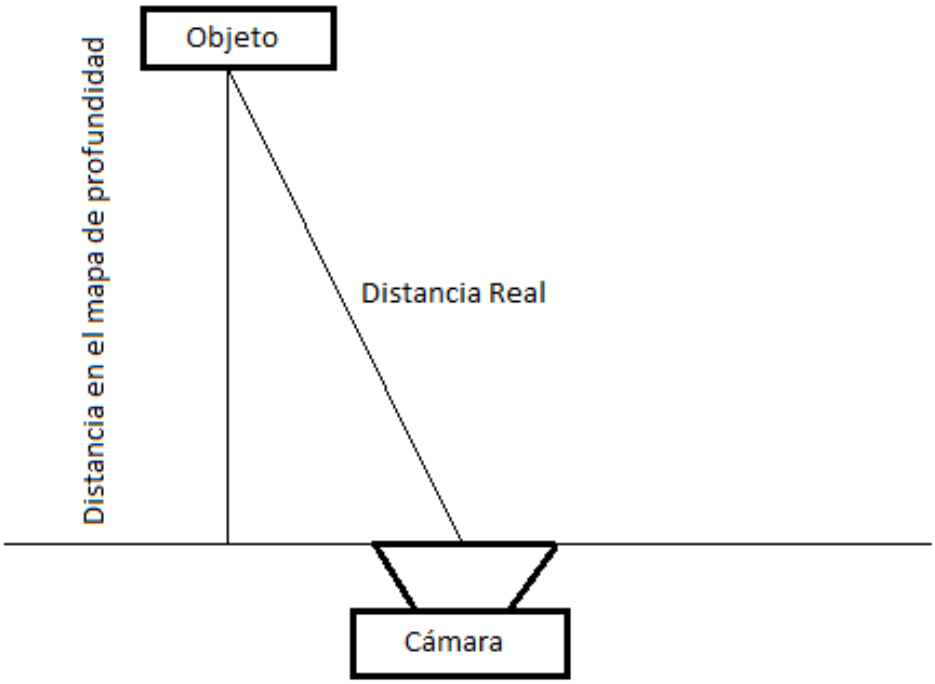


Figura 4.18: Configuración del mapa de profundidad

Basándonos en este modelo de reconstrucción, supongamos que queremos calcular la posición 3D asociada a un píxel cuya recta de retroproyección viene dada por el punto **BP2** (punto obtenido mediante la función de *backproject* de *Progeo*) y la posición de la cámara, es decir, la recta **r** en la figura 4.19. Según lo explicado hasta ahora, es de suponer que el punto buscado sea el punto **P1**, ya que es el punto que está en la recta **r** a una distancia **d** del centro óptico representado por el punto **CAM**. El problema es que el proceso de reconstrucción utilizado, se está basando en un modelo proyectivo desde el centro óptico, y sin embargo, la distancia que realmente nos da el sensor es la distancia perpendicular al plano imagen (figura 4.18). En otras palabras, la distancia codificada en la imagen de profundidad corresponde a la distancia entre la cámara y el plano paralelo al plano imagen donde se encuentra el punto real. Siguiendo con la figura 4.19 el punto real sería el punto **P2**, ya que es el punto que está en el plano paralelo al plano imagen a una distancia **d** y en la recta de retroproyección del píxel.

Para calcular el punto real **P2**, debemos encontrar el punto de intersección entre la recta **r** y el plano **Π**. Definimos el plano **Π** como aquel plano con vector normal $\overrightarrow{k}$ que pasa por el punto **K**. El $\overrightarrow{k}$ se puede obtener como el vector unitario que une el centro óptico de la cámara con el *focus of attention (**foa**)*. Este vector es un vector perpendicular al plano de la imagen. El **foa** es un punto que obtenemos desde el proceso de calibración, por lo que es un punto conocido y marca, junto con el **roll**, la orientación de la cámara en 3D.

Calculamos el vector unitario de $\overrightarrow{k}$ entre **CAM** y **foa**:

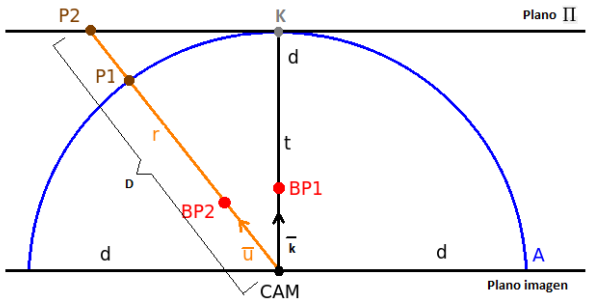


Figura 4.19: Explicación de la corrección de distancia

$$\widehat{k} = \frac{\overrightarrow{CAMFOA}}{|CAMFOA|}$$

Una vez obtenido el vector $\widehat{\mathbf{k}}$, podemos obtener el punto **K** como el punto cuya dirección es $\widehat{\mathbf{k}}$ y está a una distancia **d** de la posición de la cámara, aplicando la ecuación paramétrica de la recta:

$$K_x = CAM_x + d \cdot k_x$$
$$K_y = CAM_y + d \cdot k_y$$
$$K_z = CAM_z + d \cdot k_z$$

Con esto, ya tendríamos definido el plano en el que se encuentra el punto **P2**. El punto **P2** es aquel punto sobre la recta **r** a una distancia **D** (la distancia corregida) del punto **CAM**. Para ello, utilizamos también la ecuación paramétrica de la recta siendo $\overrightarrow{u}$ el vector director de la recta pudiendo calcular el vector unitario como:

$$\widehat{u} = \frac{\overrightarrow{CAMBP2}}{|CAMBP2|}$$

Por lo tanto:

$$P2_x = CAM_x + D \cdot u_x \tag{4.1}$$
$$P2_y = CAM_y + D \cdot u_y \tag{4.2}$$
$$P2_z = CAM_z + D \cdot u_z \tag{4.3}$$

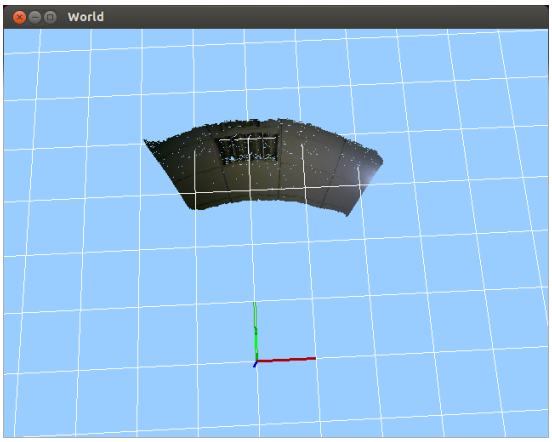
Como sabemos que el punto **P2** se encuentra en el plano **Π**, podemos aplicar la ecuación del plano: $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$. Utilizando como puntos $P(x_0, y_0, z_0) = K$ y **P'**(x,y,z)=**P2** y el vector normal al plano $\overrightarrow{N}$(A,B,C)=$\overrightarrow{k}$:

$$k_x(CAM_x + t \cdot u_x - K_x) + k_y(CAM_y + t \cdot u_y - K_y) + k_z(CAM_z + t \cdot u_z - K_z) = 0$$

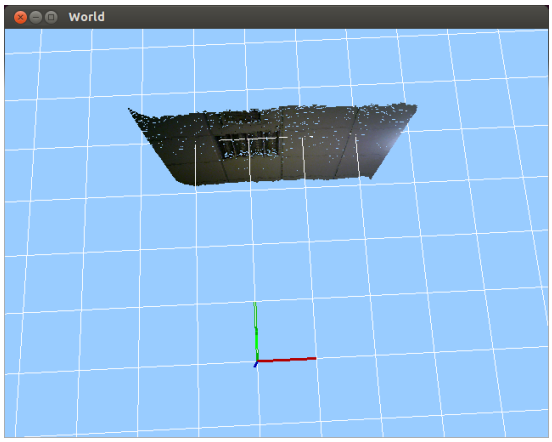
De esta fórmula, podemos despejar la **t**:

$$t = \frac{-k_x \cdot CAM_x + k_x \cdot K_x - k_y \cdot CAM_y + k_y \cdot K_y - k_z \cdot CAM_z + k_z \cdot K_z}{k_x \cdot u_x + k_y \cdot u_y + k_z \cdot u_z}$$

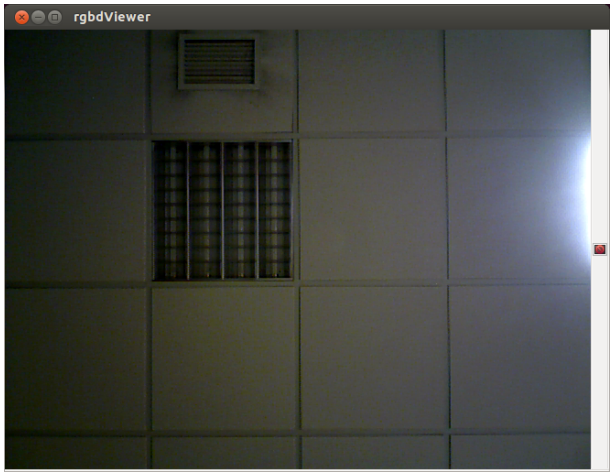
Aplicando el resultado de **t** sobre la función 4.1, obtenemos el punto real buscado **P2**. En la figura 4.20, vemos en un ejemplo real, la diferencia entre la reconstrucción con la distancia que nos da directamente el sensor y la reconstrucción con la distancia corregida. En este ejemplo, la cámara enfoca directamente al techo del laboratorio, que es plano, (figura 4.20(c)) y como podemos ver en la figura 4.20(a), la reconstrucción del plano es esférica alrededor de la cámara, mientras que una vez corregida la distancia (figura 4.20(b)), la reconstrucción 3D representa fielmente la realidad.



(a)



(b)



(c)

Figura 4.20: (a) Reconstrucción con la distancia directa (b) Reconstrucción con la distancia corregida (c) Imagen desde la cámara de color

Al utilizar *progeo* con su modelo PinHole como base de proyecciones geométricas, es necesario tener calibrada la cámara antes de realizar cualquier operación. Al registrar las imágenes tomando como imagen principal la imagen de color, bastaría con calibrar

esta cámara y asumir que dicha calibración es válida para cualquiera de las dos.

4.4. Librería DepthLib

Con el fin de aunar toda la funcionalidad desarrollada para el procesamiento de mapas de profundidad, durante la elaboración de este proyecto se ha creado una librería denominada *DepthLib*. Algunas de las funcionalidades más destacables de esta librería son un muestreado dependiente de la distancia y un filtrado de estabilización para los datos obtenidos con sensores RGB-D.

Por lo tanto, el proceso de reconstrucción en nube de puntos de primer plano, de forma simplificada, es el expresado en la figura 4.21. Donde se aplica, en una fase previa, el cálculo del primer plano del mapa de profundidad y de la imagen en color, y posteriormente, sobre estos datos ya segmentados, se calcula la nube de puntos asociada.

4.4.1. Muestreado dependiente de la distancia

Es habitual en visión artificial realizar un submuestreo de los datos ofrecidos por el sensor para optimizar la aplicación del algoritmo desarrollado. En este proceso se debe mantener una representación lo suficientemente fiel de los datos originales, reduciendo considerablemente el tamaño de los datos a procesar.

Una técnica común es aplicar técnicas de redimensionamiento de la imagen original, consiguiendo una imagen más pequeña sobre la que poder aplicar algoritmos de forma más eficiente. La librería *OpenCV* integra esta funcionalidad aplicando diferentes alternativas de interpolación (vecino más cercano, interpolación bilineal...). Este tipo de funciones no son válidas para utilizar con mapas de profundidad, ya que, por el propio proceso de interpolación, se pueden crear distancias distintas a las originales. Este fenómeno se hace patente en los bordes de los objetos donde el nuevo valor tras el redimensionamiento generaría un valor entre la distancia real del objeto y la distancia de la zona en la parte posterior del mismo (figura 4.22).

Los objetos lejanos ocupan poco espacio en las imágenes y si aplicamos una redimensión puede llegar al punto en el que diversos objetos se queden sin representación en la imagen final. Debido a las propias características del sensor utilizado, la resolución espacial disminuye con el crecimiento de la distancia. Para paliar este fenómeno intrínseco, es necesario realizar un muestreo, cuya frecuencia espacial de muestreo sea proporcional a la distancia del objeto al sensor. Si no se controla este fenómeno, para un objeto del mismo tamaño, obtendríamos muchos más puntos si el objeto está cercano al sensor, ya que ocupa más espacio en el mapa de profundidad, mientras que si está alejado tendríamos pocos puntos. Por ello, es necesario realizar una ecualización de densidades espaciales para el muestreo.

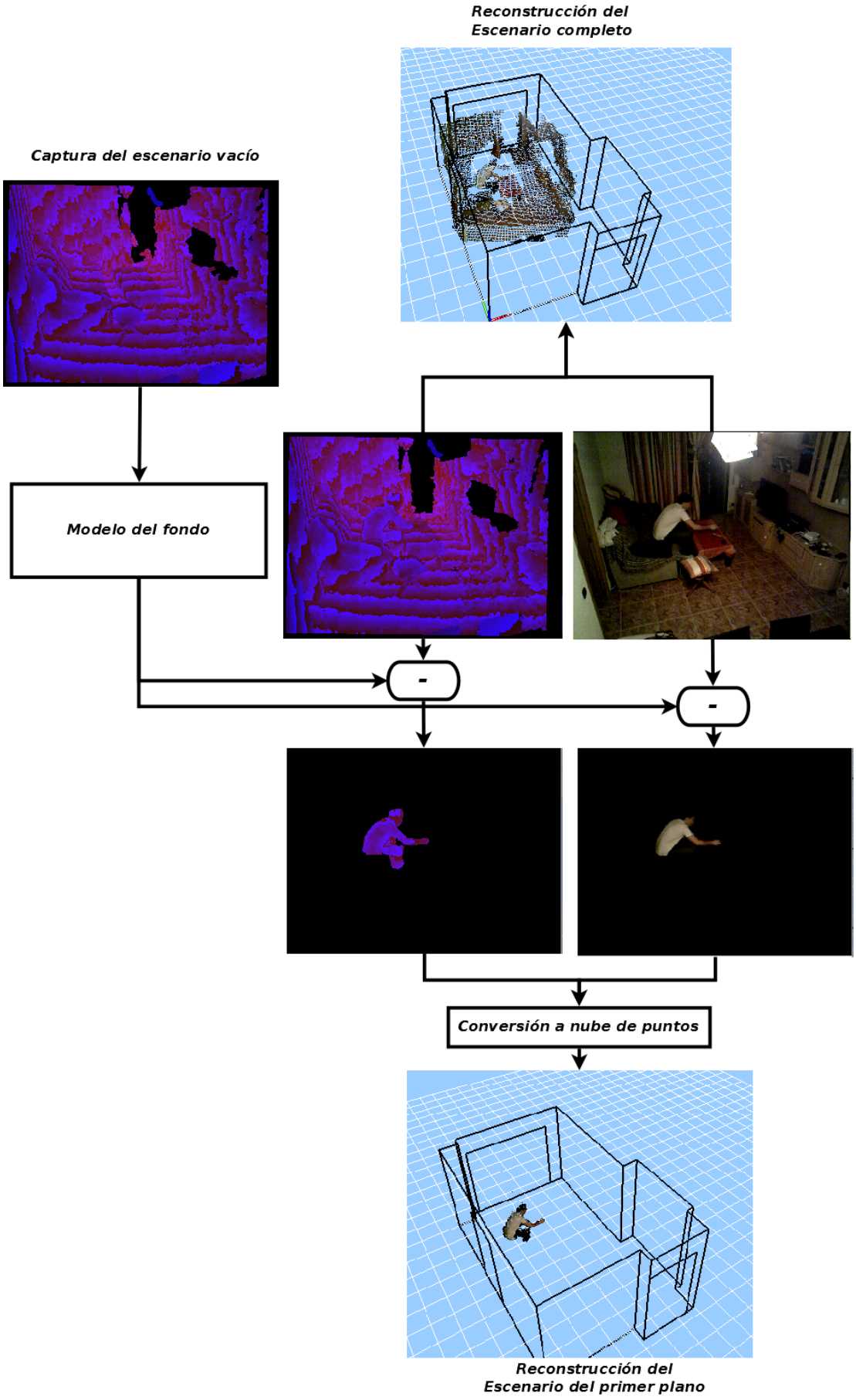


Figura 4.21: Proceso de recontrucción del primer plano de una escena en nube de puntos

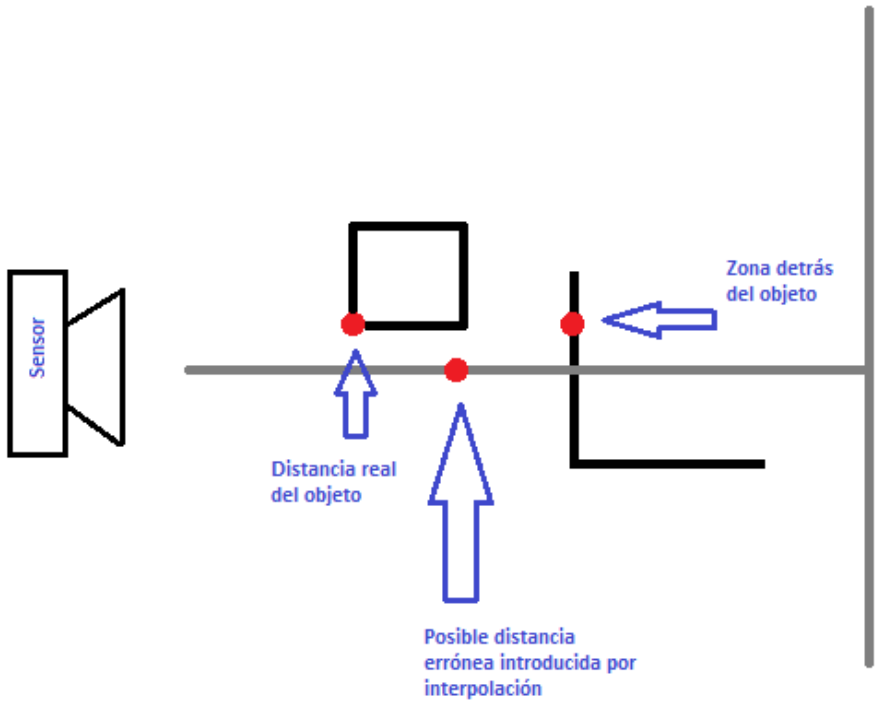


Figura 4.22: Distancia errónea introducida por el proceso de interpolación.

Con el fin de dar solución a la problemática planteada, se ha desarrollado una técnica de muestreo basada en capas. Estas capas se forman paralelas al plano imagen y aglutinan todos los puntos que estén a cierta distancia. En la figura 4.23, se puede apreciar un ejemplo gráfico de lo explicado anteriormente, donde cada color corresponde a una capa, dividiendo todo el espacio monitorizado en 8 capas. El siguiente paso es calcular una máscara para cada capa, es decir, una imagen binaria en la que estarán activos los puntos del mapa de profundidad que cumplan con la distancia de la capa a la que corresponda. Estas máscaras tendrán la posición de todos los puntos que están asociados a cierta capa. Una vez obtenidas las máscaras, se muestrea cada una de ellas con una frecuencia diferente, siendo mayor la frecuencia de las máscaras que corresponden a capas más alejadas y menor a las más cercanas.

En la figura 4.24(a), se puede observar una captura del Laboratorio de Robótica de la Universidad Rey Juan Carlos, donde el sensor llega a captar la zona exterior del umbral de la puerta, estando esta zona a 10 metros aproximadamente. Si este escenario se muestrea utilizando un método convencional, por ejemplo, muestreando uno de cada tres píxeles en horizontal y uno de cada tres en vertical se obtendría una máscara de muestreo como la representada en la figura 4.25(a). Como se puede apreciar en esta máscara, el muestreo es completamente homogéneo en la imagen y no en el espacio, es decir, la imagen como tal es muestreada pero sin tener en cuenta el espacio, lo cual implica que se muestree con mayor frecuencia la zona cercana al sensor y con menor frecuencia las zonas alejadas. Sin embargo, si aplicamos el concepto de muestreado dependiente de la distancia explicado anteriormente, la máscara de muestreo obtenida

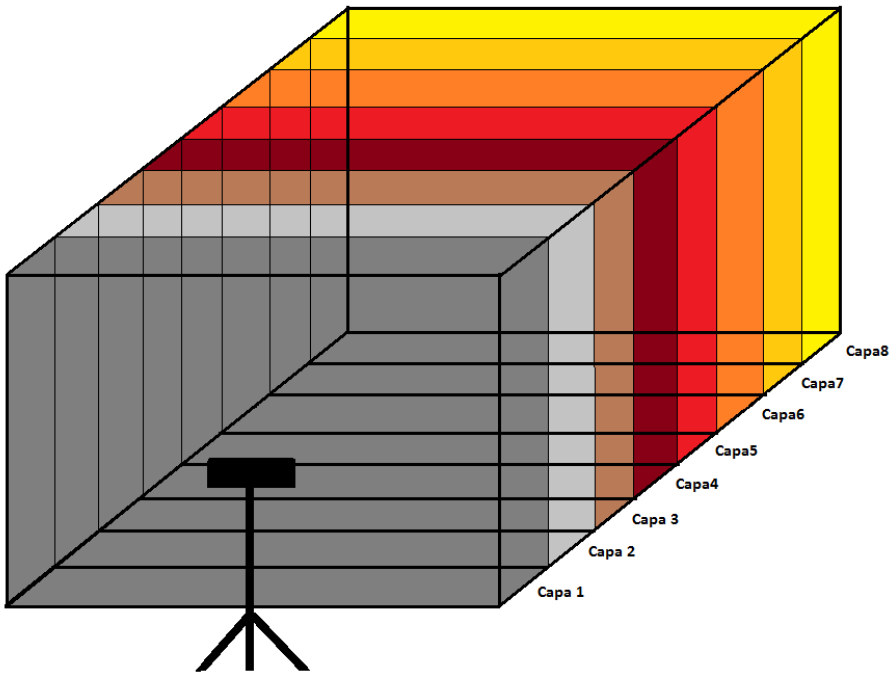


Figura 4.23: Escenario monitorizado divido en capas para el muestreo dependiente de la distancia.

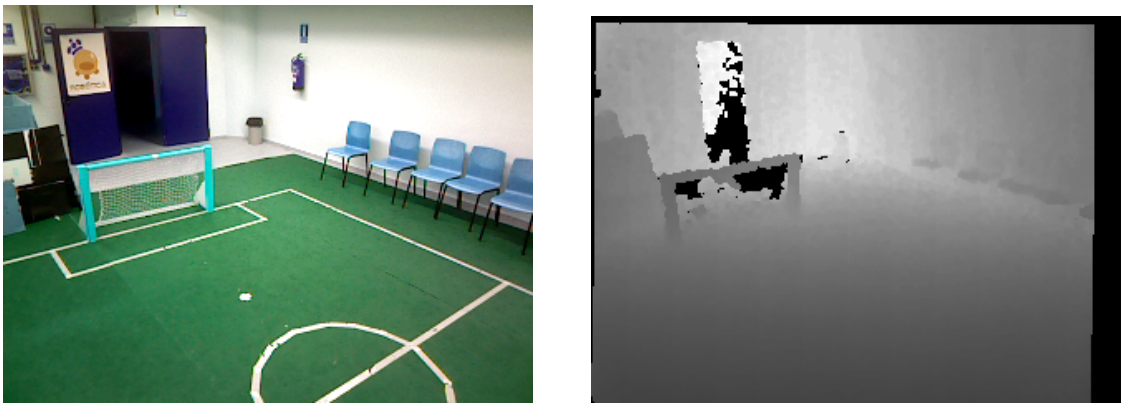
sí que depende de la distancia medida, tal y como se puede apreciar en la figura 4.25(b). Esta máscara se ha calculado a partir de la combinación de las máscaras parciales (figura 4.26(a)), muestreado cada una de ellas con una frecuencia diferentes, siendo mayor las máscaras asociadas a zonas alejadas del sensor y menor para aquellas cercanas.

En la figura 4.27, se puede ver una comparación de la distribución de puntos, en función de la distancia al sensor, obtenidos con las dos técnicas de muestreado. Como se puede apreciar, al utilizar el muestreado dependiente de la distancia se consigue una densidad estable mientras que con el muestreado estático la densidad de puntos se reduce demasiado al aumentar la distancia.

4.4.2. Filtrado temporal de los datos

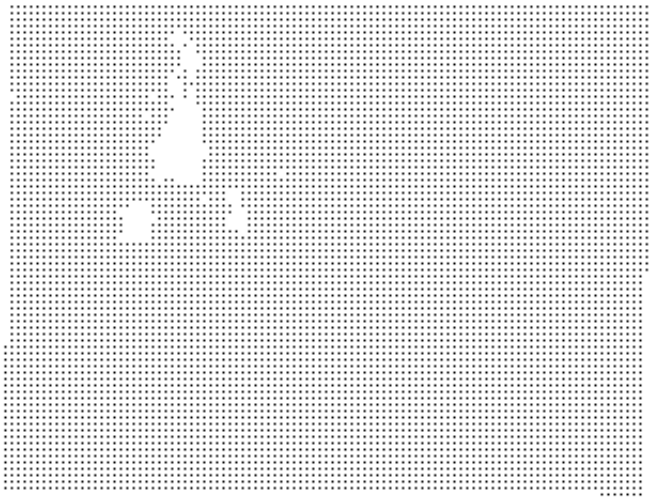
Tal y como se ha descrito con anterioridad, el sensor utilizado es un dispositivo que proporciona datos con bastante ruido y que oscilan de una captura a la siguiente. Al estar utilizando estos datos para el aprendizaje de fondo, es necesario que tengan cierta estabilidad para no tener detecciones de primer plano espúreos, que sean erróneas debido a las oscilaciones de los datos del sensor.

Una técnica habitual para conseguir estabilidad en imágenes ruidosas es utilizar un filtrado de media, es decir, tomar tomo imagen a tratar la media de las últimas n fotogramas. Esto funciona muy bien cuando se trabaja con entornos estáticos, donde no hay objetos en movimiento en la escena. El problema de aplicar esta técnica cuando se



(a) (b)

Figura 4.24: (a) Entorno captado con la cámara de color (b) Mapa de profundidad del entorno



(a)

(b)

Figura 4.25: (a) Muestreado uniforme (b) Muestreado dependiente de la distancia

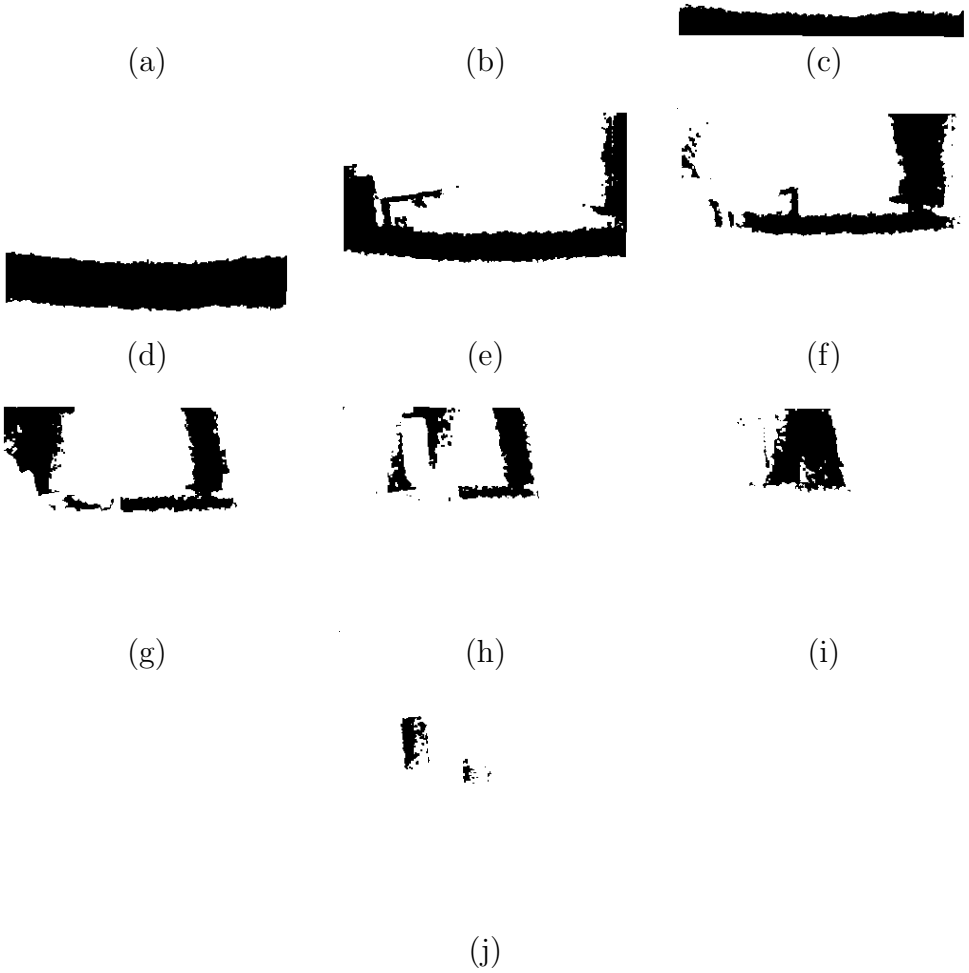


Figura 4.26: (a) Máscara de 0-1 metro (b) Máscara de 1-2 metros (c) Máscara de 2-3 metros (d) Máscara de 3-4 metros (e) Máscara de 4-5 metros (f) Máscara de 5-6 metros (g) Máscara de 6-7 metros (h) Máscara de 7-8 metros (i) Máscara de 8-9 metros (j) Máscara de 9-10 metros

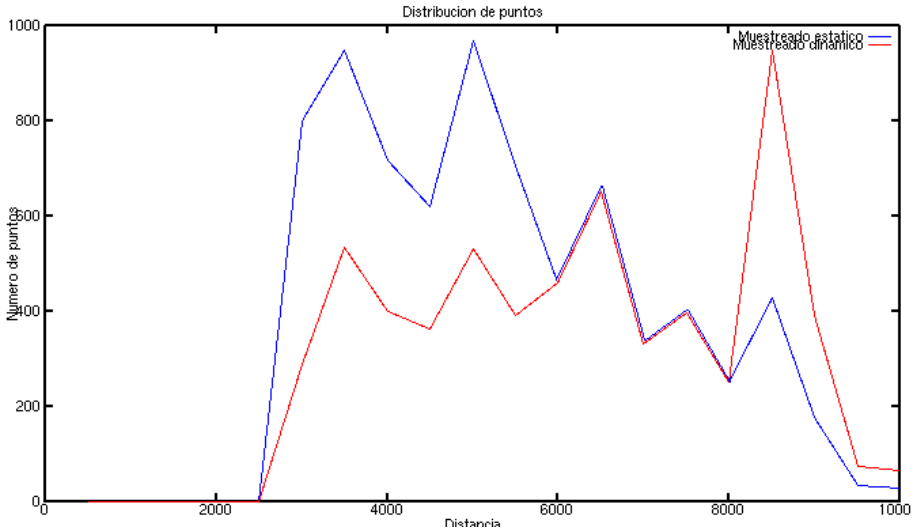
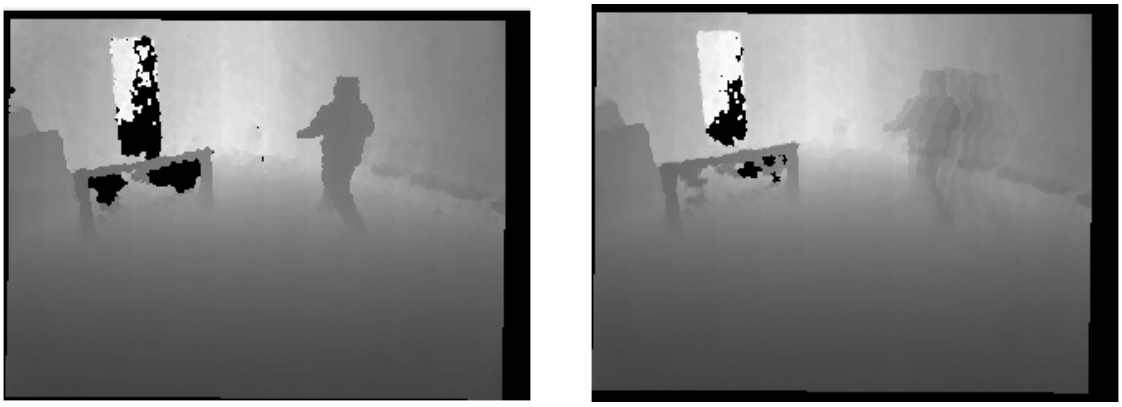


Figura 4.27: Densidad de muestreo.

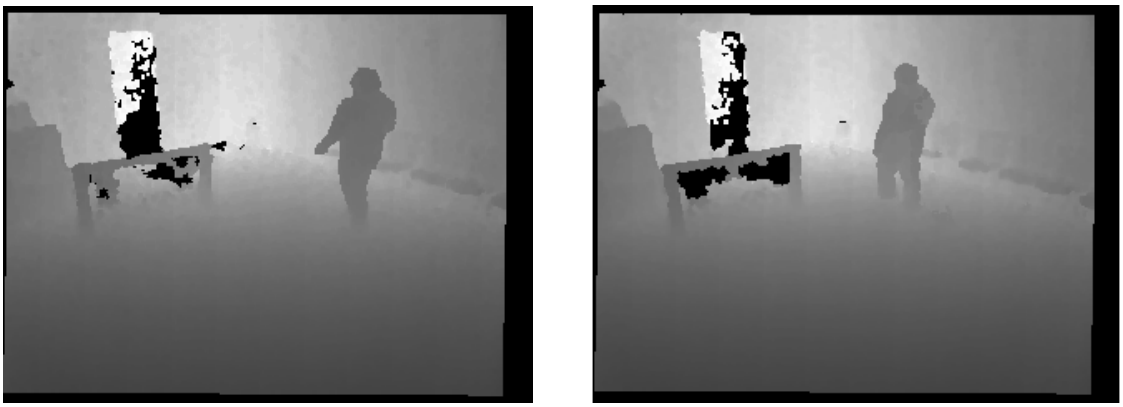
trabaja con objetos dinámicos es que aparece un efecto *fantasma* en torno al trayecto del objeto, y también que el objeto en movimiento nunca se capta con detalle ya que siempre va a estar promediado con valores del fondo. Este efecto se puede apreciar en la figura 4.28, donde se muestra una imagen sin ningún tipo de filtro y una imagen con un filtrado promedio con un tamaño de *buffer* de 10 fotogramas.



(a) (b)

Figura 4.28: (a) Imagen sin filtrar (b) Imagen con filtro promedio

Como hemos introducido en esta sección, es necesario un preprocesado de las imágenes para mejorar los resultados del aprendizaje de fondo, por lo que sólo es crítico estabilizar el fondo, es decir, las zonas del entorno que no se mueven. Como hemos comentado, el filtro de media funciona muy bien para este fin. Por todo ello, la idea del filtrado implementado es aplicar, por un lado, un filtro de media para las zonas estáticas, y por otro lado, para las zonas con movimiento se utilizar los datos de la imagen actual sin ningún tipo de procesado. Aplicando este filtrado selectivo, reducimos las estelas que se generarían usando símplemente un filtrado por media.



(a) (b)

Figura 4.29: (a) Imagen sin filtrar (b) Imagen con filtrado combinado

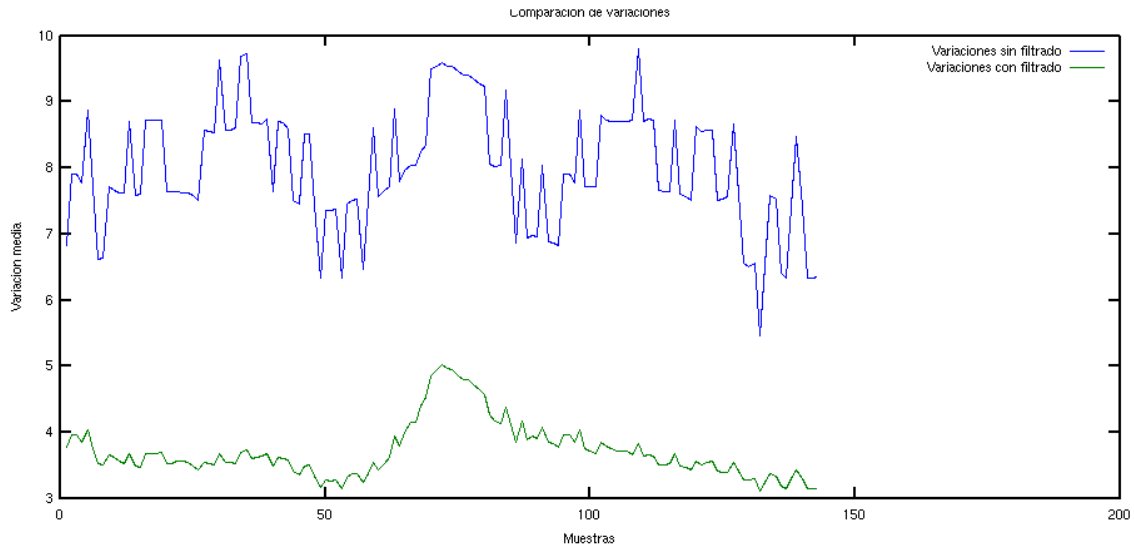


Figura 4.30: Comparación de las variaciones entre imágenes secuenciales con y sin filtrado.

Esta funcionalidad se ha desglosado en los siguientes pasos:

1. Se calcula la imagen promedio de los últimos n fotogramas.

2. Siendo la imagen *source* la imagen actual y *imgBufer* la imagen promedio, se genera la imagen actual filtrada como:

$$imgFiltered(x,y) = \begin{cases} imgBuffer(x,y) & \text{si } |imgBuffer(x,y)| < umbral \\ source(x,y) & \text{si } |imgBuffer(x,y)| > umbral \end{cases} \tag{4.4}$$

Si nos fijamos en la imagen 4.29(a), donde no se aplica ningún filtrado, y la imagen de la figura 4.29(b), en la que sí se aplica el filtrado detallado anteriormente, vemos que no hay grandes cambios a simple vista. Sin embargo, en la figura 4.30 se puede

ver una comparación de la variación, utilizando la fórmulas 4.5 y 4.6, de una captura con la anterior. En esta comparación, podemos observar que las variaciones con el filtrado aplicado son mucho menores, consiguiendo datos más estables, cumpliendo con las especificaciones del filtrado planteado.

$$x_{azul} = \frac{1}{Height * Width} \sum_{XY} |Img(t) - Img(t-1)| \tag{4.5}$$

$$x_{verde} = \frac{1}{Height * Width} \sum_{XY} |ImgFiltered(t) - ImgFiltered(t-1)| \tag{4.6}$$

Capítulo 5
Seguimiento 3D

Después de una primera parte del documento en la que se han detallado las herramientas creadas para dar soporte a este proyecto así como las diversas librerías utilizadas, se detallará en este capítulo el diseño elegido para el sistema de detección y seguimiento de personas 3D y la implementación desarrollada, así como las diferentes situaciones de riesgo que es capaz de detectar.

El sistema presentado se basa en el procesamiento de información 3D tratada como una nube de puntos. Estos datos son obtenidos desde el componente *BackgroundSubstractor*, el cual sirve la información 3D únicamente del primer plano, procesando la información captada desde sensores a los que se conecta mediante el componente *OpenNIServer* (figura 5.1). La idea fundamental del algoritmo presentado es tratar de detectar zonas densas del entorno, que serán tomadas como personas, y seguirlas a lo largo del tiempo, todo ello utilizando como premisa volumétrica prismas 3D de tamaño variable. El comportamiento encapsulado dentro de un componente de *JdeRobot* denominado *ElderCare*.

5.1. Diseño

Como sistema de teleasistencia pensado para monitorizar hogares de personas con algún tipo de dependencia o estancias en residencias de ancianos, es necesario cubrir con sensores diversas zonas de forma simultánea colocando, para ello, los sensores que sean necesarios. En la figura 5.2, se puede apreciar la distribución típica de sensores necesarios para implantar el sistema de teleasistencia presentado en este proyecto en una casa particular. La situación expuesta sería una situación ideal en la que está monitorizado prácticamente todo el espacio útil del domicilio. El baño quedaría exento de monitorización directa, debido a temas de privacidad y posible incomodidad del propio sujeto. Tal como se detallará posteriormente, se ha ideado un sistema de detección de situaciones de riego en el baño de forma indirecta.

La estructura del sistema implementado en este proyecto sigue un modelo distribuido, utilizando la red para conectar cada uno de los componentes implicados. Esto ofrece la posibilidad de tener múltiples nodos satélites situados en diferentes estancias de una casa, transmitiendo la información de cada sensor a un nodo principal, utilizando una red local. Este nodo principal será el encargado de procesar la

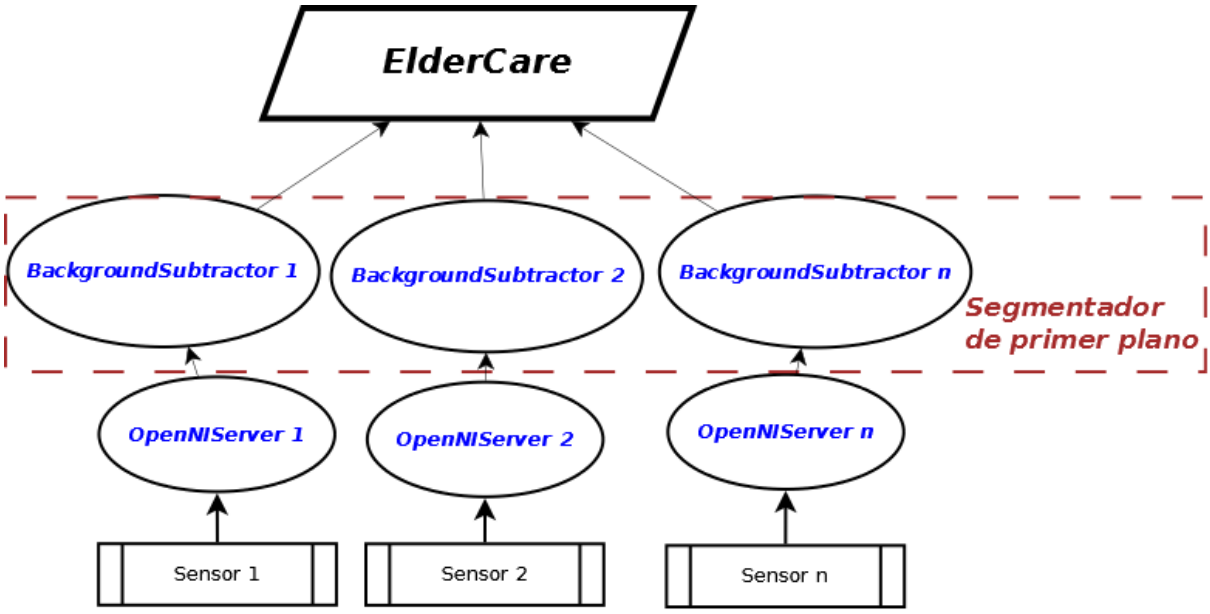


Figura 5.1: Esquema básico del sistema desarrollado

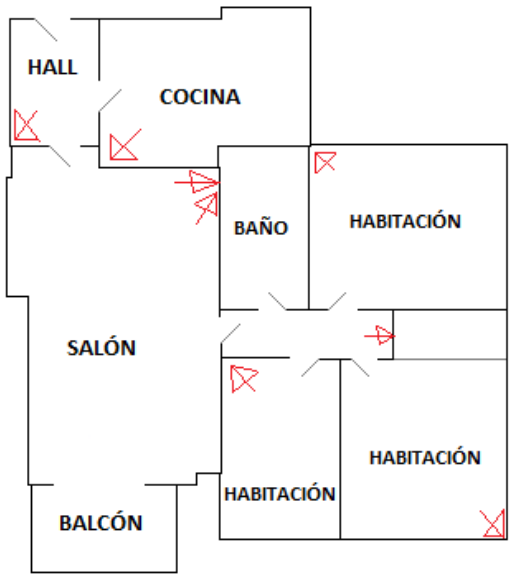


Figura 5.2: Distribución de sensores en una casa

información y detectar las diferentes situaciones de riesgo. Este sistema tendrá una serie de servidores *OpenNIServer* conectados a los dispositivos físicos. Estos componentes ofrecen los datos a sus *BackgroundSubstractor* correspondientes, los cuales se encargan de segmentar los datos de interés y ofrecer los mismos al nodo central, donde se ejecuta el componente *ElderCare*. También tendremos disponibles dos componentes para el acceso remoto al sistema, que se pueden ejecutar en cualquier ordenador, permitiendo visualizar el estado del sistema (*RemoteViewer*) o bien modificar la configuración del mismo (*RemoteConfigurator*). Estos dos componentes serán detallados en este mismo capítulo.

Dentro del componente *ElderCare*, la funcionalidad se ha dividido en módulos (figura 5.3). Un módulo se encarga de pedir los datos a los diferentes componentes *BackgroundSubstractor*, utilizando la librería *ParallIce* y alojarlos en memoria compartida (módulo de adquisición). De esta forma, se obtienen los datos ofrecidos por los sensores, en forma de nube de puntos, pero una vez aplicada la técnica de sustracción de fondo, detallada en la sección 4.3. Otros tres módulos principales son el módulo que realiza todas las tareas de detección y seguimiento de personas en 3D, cuya tarea es agrupar los puntos de primer plano en prismas y seguirlos en el tiempo, el módulo que genera y gestiona las diferentes situaciones de riesgo y alarmas y por último el módulo del interfaz gráfico o GUI, encargado de representar gráficamente todos los datos generados por el algoritmo. Estas circunstancias peligrosas se pueden detectar mediante varios indicadores como son la posición de la persona, altura, ausencia de paso, etc. Por otra parte, *ElderCare* incluye dos componentes dedicados exclusivamente a la comunicación con componentes externos de *JdeRobot*: el componente de configuración remoto y el componente de visualización remota. Cada uno de estos módulos trabaja de forma independiente, con un hilo de ejecución dedicado, sobre memoria compartida debidamente gestionada para no generar condiciones de carrera. Esto permite que, apoyándose en la arquitectura multinúcleo, la aplicación desarrollada pueda ser ejecutada de forma muy ligera llegando al tiempo real. En el módulo de detección y seguimiento de personas 3D, se ha implementado un algoritmo evolutivo multimodal. Este algoritmo constituye el corazón del sistema y será explicado en detalle en los siguientes apartados.

5.2. Módulo de detección y seguimiento multipersona

Este módulo se encarga de detectar nuevas personas que puedan entrar en la escena monitorizada y realizar un seguimiento de ellas hasta que desaparezcan. Como detectar personas sobre toda la escena generada por los sensores RGB-D es una tarea sumamente complicada, se ha decidido aplicar una técnica para eliminar el fondo de la escena y aplicar nuestro algoritmo únicamente sobre los datos de primer plano, es decir, los objetos en movimiento. Una vez realizada esta sustracción, es necesario explorar aquellas zonas densas de espacio y tratar de explicar estas zonas con prismas de volumen variable. Si un prisma cumple una serie de condiciones en una zona concreta, se asumirá que éste está agrupando puntos de una persona y se le empezará a aplicar

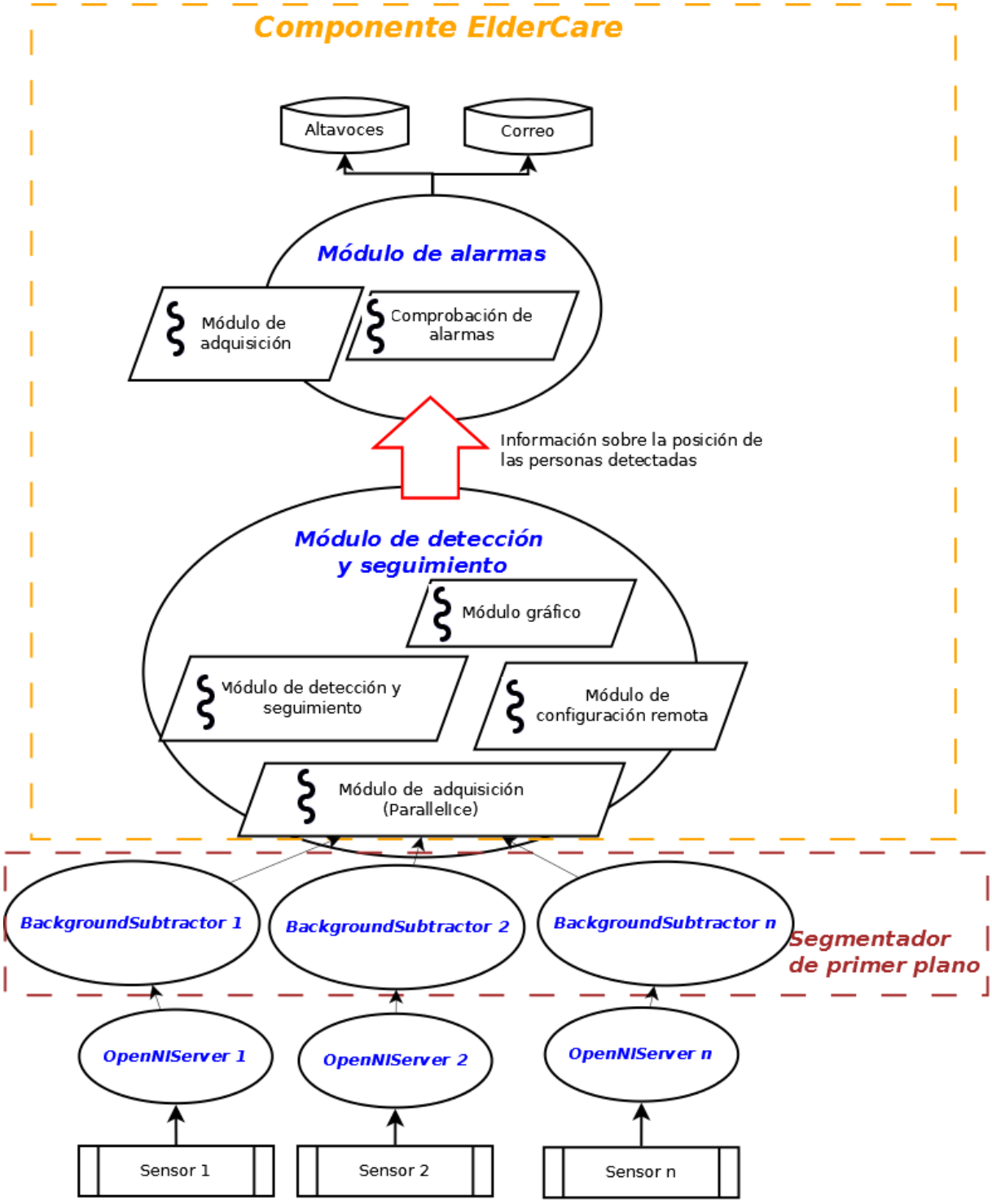


Figura 5.3: Diseño interno de los módulos que componen ElderCare

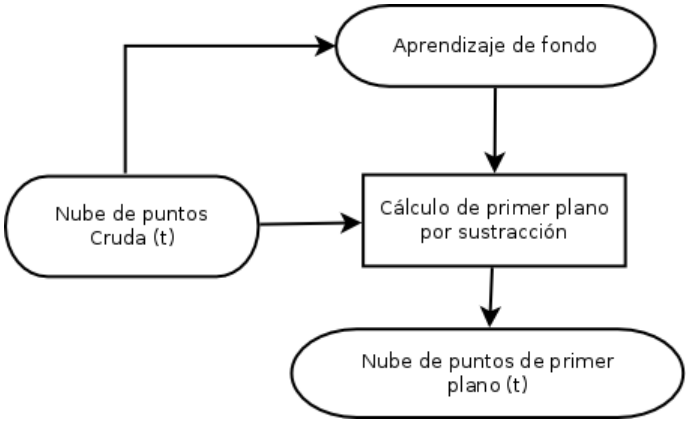


Figura 5.4: Esquema de la sustracción de fondo

la parte de seguimiento del algoritmo.

Para simplificar el problema de la detección de personas, en un paso previo a aplicar cualquier tipo de análisis, se aplicará una sustracción de fondo. De esta manera, nuestro sistema no tiene que procesar todos los datos ofrecidos por el sensor, solamente aquellos que forman parte del primer plano. El modelo de fondo a utilizar para el correcto funcionamiento del sistema, es una instantánea del entorno vacío. De esta forma, cualquier objeto que se mueva en el entorno será tomado como primer plano y, por consiguiente, asumimos que estos movimientos serán realizados por las personas que se estén moviendo en él.

Para ello, se ha utilizado el componente *BackgroundSubstractor*, explicado con detalle en la sección 4.3. La salida de este componente es la nube de puntos correspondiente a las zonas segmentadas como primer plano (figura 5.4). Sobre esta nube segmentada, se aplicará el algoritmo de detección y seguimiento implementado.

5.2.1. Algoritmo evolutivo multimodal

Los algoritmos evolutivos son métodos de optimización y búsqueda basados en los conceptos de la evolución biológica. Estos algoritmos, basados en la evolución de una serie de individuos o soluciones candidatas mediante el uso de operadores genéticos, son así capaces de competir entre sí y prevalecer en el tiempo dando solución a un problema. En este trabajo se utiliza un enfoque multimodal, en el que existen varias poblaciones, tratando de dar solución al seguimiento de cada una de las personas que hayan sido detectadas en el entorno monitorizado.

Este tipo de algoritmos basan su comportamiento en evolucionar de una u otra forma los individuos de cada población, tomando como referencia la calidad de éstos. Es vital fijar algún tipo de evaluación que puntúe de forma correcta los individuos que se crean y evolucionan. Esta evaluación, debe medir la calidad de las soluciones en función de la compatiblidad con las observaciones sensoriales.

Los individuos que componen una población son evaluados en cada iteración con el

fin de conocer la calidad de la solución que está aportado, es decir, cuánto se acerca a la solución óptima del problema. Este algoritmo ya se ha aplicado con muy buenos resultados para la autolocalización de robots [Perdices García, 2010], localización de segmentos en 3D [García Martínez, 2007] e incluso para la problemática del presente trabajo, el seguimiento de personas en 3D [Marugán Alonso, 2010a], aunque con un enfoque muy diferente ya que se utilizaban cámaras convencionales con información de color y no directamente sobre información 3D.

Se ha creado un algoritmo evolutivo multimodal para este módulo que es capaz de detectar y seguir, a lo largo del tiempo, a un número indefinido de personas en 3D a lo largo del tiempo. Como primitiva, tanto de la detección como de seguimiento, se ha utilizado un prisma 3D de volumen variable. Este algoritmo consta de los siguientes agentes:

- **Individuos:** corresponden con la representación seleccionada de una persona en 3D, es decir, la posición del prisma (x,y,z) y el tamaño del mismo (dx,dy,dz) (figura 5.5). Además, guardan también información acerca de su calidad y del número de puntos 3D de la nube de puntos que contiene.

- **Razas:** una raza es un conjunto de individuos que trata de dar solución al seguimiento de una persona en el mundo. Dentro de este algoritmo, pueden coexistir varias razas, cada una de ellas dando solución al seguimiento de una persona en concreto.

- **Exploradores:** individuos completamente independientes que tratan de explorar el espacio sensorizado para encontrar nuevas posiciones susceptibles de convertirse en raza.

- **Explotadores:** son aquellos individuos que forman parte de una raza existente. Examinan en detalle una posición en concreto y las zonas cercanas en busca de la mejor solución posible.

El proceso completo que sigue cada iteración del algoritmo diseñado se puede ver en la figura 5.6. Los puntos más importantes de este algoritmo serán detallados a continuación.

5.2.2. Generación de exploradores

Generalmente, en este tipo de algoritmos la generación de exploradores se realiza de forma completamente aleatoria, o bien, repartiéndolos de forma uniforme en el espacio de las posibles soluciones, siempre que el enfoque sea a zonas acotadas. Sin embargo, en el sistema planteado en este proyecto, no se lanzarán exploradores en todas las zonas posibles, sino que se utilizarán puntos seleccionados aleatoriamente de la nube de puntos para generar los prismas alrededor de ellos (generación abductiva), lo que

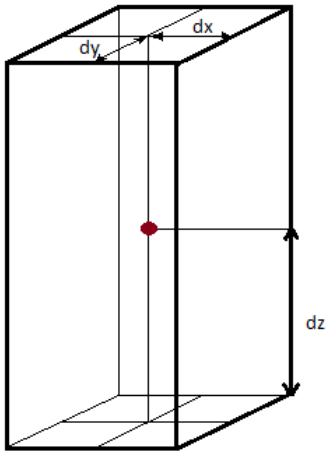


Figura 5.5: Modelado de un prisma en tres dimensiones

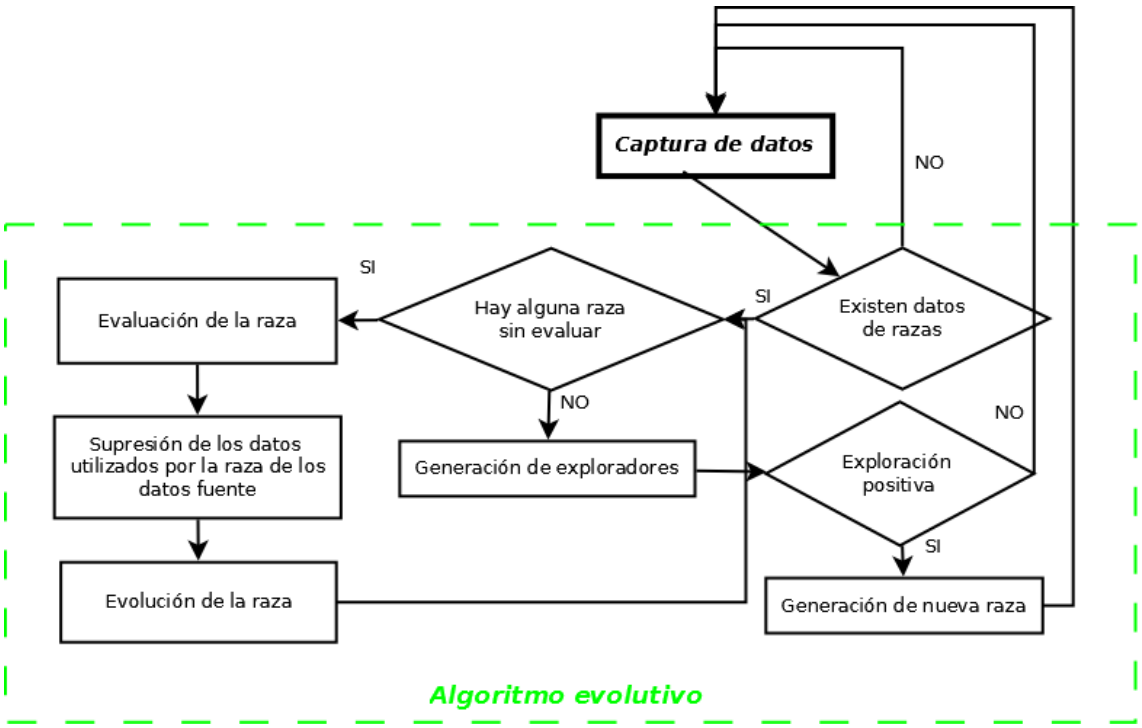


Figura 5.6: Flujo del algoritmo de seguimiento 3D

supone una generación más lista que el azar puro. De esta forma, nos aseguramos que el explorador contenga al menos un punto y se descartan zonas del espacio en las que no se ha recogido ningún dato. Siguiendo este paradigma, pueden haber iteraciones en las que no se compute ningún explorador, bien porque no se haya captado ningún dato, o bien porque todos los puntos hayan sido ya explicados con otras razas.

Como hemos comentado en el punto anterior, un individuo describe una zona del espacio como un prisma de volumen variable. En el caso de los exploradores, dado que no tenemos ningún dato a priori sobre sus dimensiones, se ha optado por utilizar prismas de tamaño fijo 300x300x1000 (mm).

Para que un explorador sea susceptible de ser base para crear una nueva raza debe cumplir una serie de condiciones:

1. Englobar al menos n puntos.
2. Ser mayor que cierto tamaño mínimo (d_x, d_y, d_z). Ya que todos los exploradores tienen un tamaño fijo para la búsqueda en el espacio, es necesario ajustar el tamaño del prisma a la zona que realmente está englobando, calculando su envolvente y siendo ésta sobre la que se evalúa el tamaño mínimo.
3. Estar a más de x cm de cualquier raza ya creada, para evitar duplicidades innecesarias.

5.2.3. Creación de Razas

Una vez que un individuo explotador ha sido evaluado y su resultado ha sido satisfactorio, se genera una nueva raza tomándole a él como base. Si el resultado no supera un umbral de caldiad, se descarta. La primera población de cada raza se genera aplicando el operador genético de ruido térmico sobre los datos del explorador exitoso. Este operador sitúa un nuevo individuo en una posición cercana al individuo original lo que permite hacer una búsqueda local, distribuyendo partículas como ruido gaussiano.

Cada uno de los individuos pertenecientes a la población de una raza se denominan explotadores y suponen el conjunto de soluciones individuales con las que se trata de dar solución exacta a la búsqueda planteada de una persona concreta, ya localizada de manera aproximada. Estos explotadores van evolucionando a lo largo de la ejecución del algoritmo para conseguir una solución persistente.

5.2.4. Evolución de razas

En un momento genérico, el algoritmo tiene n razas activas, al llegar nuevos datos se evalúan los resultados obtenidos con cada uno de los explotadores de la población de la raza, asignándole a cada uno de ellos un valor de calidad. En función de los resultados obtenidos y aplicando operadores genéticos, se evolucionan estos individuos para generar una nueva población que dé solución al problema en la siguiente

iteración, ajustando mejor el tamaño y posición del prisma adaptándose a la evidencias sensoriales. En este proyecto se han utilizado dos operadores genéticos en concreto:

- **Elitismo:** consiste en seleccionar los n explotadores que mejor calidad han obtenido con la nube de puntos actual y mantenerlos en la población de la siguiente generación.
- **Mutación térmica:** consiste en aplicar ruido térmico aleatorio sobre los individuos elitistas modificando tanto su posición como el tamaño del prisma que describe. Con esto se consigue generar nuevos individuos alrededor de los mejores individuos, variando levemente tanto su posición como su tamaño.

El reparto de partículas generadas es: por elitismo el 20 %, mientras que las creadas por mutación térmica el 80 %. Con este reparto se consigue mantener siempre soluciones de alta calidad y hacer evolucionar la población por mutación térmica sobre las mejores soluciones.

En este tipo de aplicaciones basadas en visión, es muy común incluir algún descriptor relacionado con el aspecto visual de las razas, incluyendo así una nueva variable que compruebe la similitud visual entre los datos de la raza y los nuevos datos. Al no añadir información de color en ninguna etapa de nuestro algoritmo de detección y seguimiento, se consigue que el sistema funcione exactamente igual con luz que completamente a oscuras. Sin embargo, esto supone una gran desventaja a la hora de hacer el emparejamiento temporal, ya que no podemos utilizar ningún tipo de información de color como discriminación entre una persona y otra.

A la hora de realizar el seguimiento sobre personas que están lo suficientemente alejadas, el algoritmo funciona de forma correcta ya que no tiene conflictos en el emparejamiento. Sin embargo, si dos personas a las que está siguiendo el algoritmo están relativamente cerca, los explotadores de una raza pueden tomar puntos de la nube de otra persona y producirse rupturas en el seguimiento y generar emparejamientos erróneos. Una forma de paliar este fenómeno es reducir la búsqueda local que realizan los explotadores creados por mutación térmica y no explorar zonas amplias donde puedan estar datos de otras personas. Esta restricción se puede incluir ya que *OpenNIServer* es capaz de servir datos con alta frecuencia sin producir grandes saltos entre diferentes capturas de una misma persona. Una segunda restricción para ayudar a solucionar este problema es limitar el volumen máximo del prisma de los individuos explotadores. Si alguno de los prismas evoluciona con un volumen que supere el umbral determinado se vuelve a repetir la operación de mutación.

Una vez evolucionada la raza, se eliminan de los datos fuentes los puntos que se han utilizado para evolucionar, es decir, los puntos que se encuentran dentro de los individuos elegidos mediante la aplicación de elitismo. Con esto, conseguimos que no influyan en los resultados de la evaluación y evolución de otras razas. Por último, las razas que no se han conseguido asociar con los datos de las observaciones se marcará como no asociadas (figura 5.7) y se eliminará si en un corto periodo de tiempo

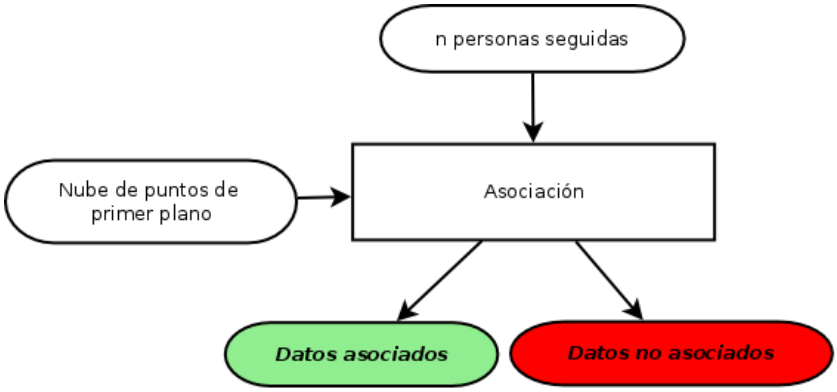


Figura 5.7: Asociación de datos con las razas presentes en el sistema

no consigue asociaciones.

Cada raza implementa el seguimiento de una persona en concreto. El mejor individuo de la población de cada raza será el representante de dicha raza y se utilizará como salida del algoritmo de seguimiento. Para determinar cuál es el mejor individuo, es necesario conocer la calidad, mediante la aplicación de la función salud, de la solución aportada por todos los explotadores seleccionando directamente el individuo que mayor calidad haya obtenido.

5.2.5. Filtro de Kalman

Con el fin aumentar la fiabilidad del algoritmo de seguimiento, se introdujo un modelo dinámico para cada una de las razas, de forma que se pueda estimar dónde estará la raza en la siguiente iteración en función de las posiciones observadas con anterioridad. La idea es combinar el algoritmo evolutivo con el modelo de seguimiento, de modo que la estimación realizada por nuestro algoritmo sirva como observación del filtro de Kalman, que incorpora su propio modelo de movimiento.

Una de las técnicas de modelado dinámico más utilizadas en sistemas ruidosos es el filtrado de Kalman [Kalman *et al.*, 1960]. La técnica de Kalman permite la estimación de una variable de la cual se disponen una serie de observaciones temporales. Esta estimación se realiza aplicando un algoritmo iterativo óptimo de procesamiento de datos. El procesamiento de Kalman consta de dos partes, una primera parte de estimación del estado a partir de datos anteriores y una segunda parte de corrección donde se reestima el estado actual a partir de la estimación anterior usando las observaciones.

Una funcionalidad extra que permite el filtro de Kalman es la persistencia del estimador 3D, manteniendo las razas cierto tiempo en el algoritmo aún sin haber encajado con ninguna observación, utilizando su modelo de movimiento. Esto permite que el sistema se recupere frente a oclusiones temporales totales sin descartar a esa

raza. Evita volverla a crear, consiguiendo así realizar el seguimiento de forma correcta aún sin haber sido captada por los sensores durante un intervalo de tiempo.

Kalman estima el estado x $\in \Re^n$ de un proceso lineal discreto en el tiempo a partir de un conjunto de medidas z $\in \Re^m$, gobernado por las ecuaciones de la dinámica del estado a modelar (ecuación 5.1) y la ecuación de relación entre la el estado X y la observación Z (ecuación 5.2).

$$X_{k+1} = A_k X_k + B u_k + W_k \tag{5.1}$$

$$Z_k = H X_k + V_k \tag{5.2}$$

Donde $\mathbf{w_k}$ y $\mathbf{v_k}$ representan el ruido del proceso y de la medida respectivamente asumiendo que son ruidos blancos de media cero e independientes.

En este trabajo, el estado **x** corresponde con la posición y el volumen del prisma ($\mathbf{x_x}, \mathbf{x_y}, \mathbf{x_z}, \mathbf{d_x}, \mathbf{d_y}, \mathbf{d_z}$) y la configuración dinámica del estado expresada como un vector de velocidades de la persona en los seis planos, tres para la velocidad de movimiento y tres para la velocidad de variación del tamaño del prisma ($\mathbf{v_x}, \mathbf{v_y}, \mathbf{v_z}, \mathbf{v_{dx}}, \mathbf{v_{dy}}, \mathbf{v_{dz}}$). Este proceso será alimentado con las observaciones proporcionadas por el algoritmo evolutivo que serán los datos resultantes del explotador con mayor calidad, siendo éstos la posición y el tamaño del prisma estimado ($\mathbf{z_x}, \mathbf{z_y}, \mathbf{z_z}, \mathbf{z_{dx}}, \mathbf{z_{dy}}, \mathbf{z_{dz}}$).

$$\begin{cases} X_{x_{k+1}} = X_{x_k} + V_x \cdot dt \\ X_{y_{k+1}} = X_{y_k} + V_y \cdot dt \\ X_{z_{k+1}} = X_{z_k} + V_z \cdot dt \\ X_{dx_{k+1}} = X_{dx_k} + V_{dx} \cdot dt \\ X_{dy_{k+1}} = X_{dy_k} + V_{dy} \cdot dt \\ X_{dz_{k+1}} = X_{dz_k} + V_{dz} \cdot dt \\ V_{x_{k+1}} = V_{x_k} \\ V_{y_{k+1}} = V_{y_k} \\ V_{z_{k+1}} = V_{z_k} \\ V_{dx_{k+1}} = V_{dx_k} \\ V_{dy_{k+1}} = V_{dy_k} \\ V_{dz_{k+1}} = V_{dz_k} \end{cases} \qquad A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & dt & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & dt & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

La matriz **B** permite incluir en el proceso una entrada de control externa. En el trabajo actual no se introduce información adicional, por lo que **B** será igual a 0, lo que nos permite descartar la variable **u**. La matriz **H**, en cambio, relaciona las observaciones o mediciones con el estado del sistema:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Los elementos que intervienen en el modelado del filtro de Kalman son:

- $\widehat{\mathbf{X}}_\mathbf{k}$: Vector estado actual en el instante K.
- $\widehat{\mathbf{X}}_\mathbf{k}^-$: Vector estado desde el estado en k-1 echando para delante el modelo de ecuaciones del estado antes de actualizar con las medidas sensoriales.
- $\mathbf{P}_\mathbf{k}$: Matriz que contiene la covarianza del error de la estimación actual del estado (al igual que con los vectores estado a priori y a posteriori).
- $\mathbf{Z}_\mathbf{k}$: Vector con las medidas sensoriales del sistema en el instante actual.
- $\mathbf{R}_\mathbf{k}$: Matriz que denota el error de la medida.
- $\mathbf{Q}_\mathbf{k}$: Matriz que denota el error del proceso.
- $\mathbf{K}_\mathbf{k}$: Ganancia de Kalman, establece la influencia del error entre una estimación y la medida.

La siguiente ecuación computa el estimador del estado a posteriori como una combinación lineal del estado a priori y una corrección. Donde el segundo término es llamado innovación de la medida, siendo la diferencia entre la estimación de Kalman y la observación real del sistema, multiplicado por la ganancia:

$$\widehat{X}_k = \widehat{X}_x^- + K_k(Z_k - H\widehat{X}_k^-)$$

Este residuo está ponderado por la *Ganancia de Kalman*, que será computada de acuerdo a la incertidumbre que se tenga en el sistema de observaciones. Este parámetro determina la proporción del error que será utilizado para corregir el vector de estados.

$$K_k = P_k^- H^T (H P_k^- H^T + R_k)^{-1}$$

Aunque el filtro de Kalman es un proceso de optimización automático es necesario inicializar una serie de parámetros que definirán en gran medida el comportamiento del filtro:

- $\widehat{\mathbf{X}_0}$ y $\mathbf{P_0}$: son el estado inicial el sistema y la covarianza del error en ese estado. Estos parámetros no son demasiado críticos ya que iterativamente se irán actualizando y fijando en los valores correctos.
- **R:** es el error cometido por el sistema a la hora de realizar las observaciones, se puede fijar directamente comprobando las medidas del sistema con los datos reales. En función de este parámetro se define cuánto se confía en las mediciones realizadas por el sistema.

- **Q:** es más complicado de fijar ya que es el error generado por el propio sistema de Kalman y no tenemos acceso a sus estados.

Generalmente, estos parámetros se fijan realizando pruebas y ajustando los parámetros para que se comporte de la manera adecuada.

El funcionamiento del filtro de Kalman, así como las ecuaciones utilizadas en cada estado, queda reflejado en la figura 5.8.

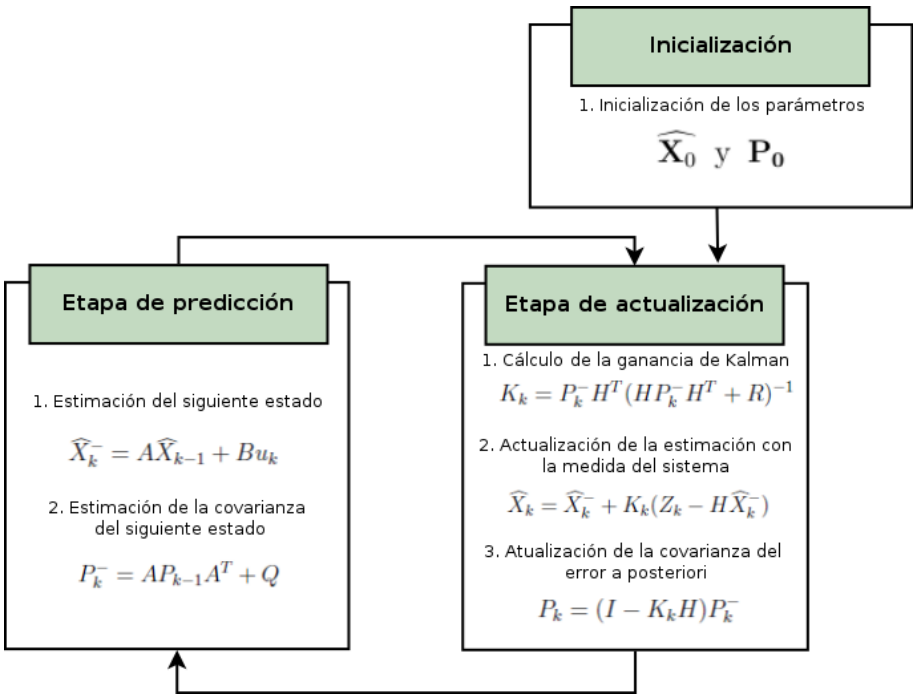


Figura 5.8: Diagrama del filtrado de Kalman.

En este trabajo, se ha utilizado el filtro de Kalman para estimar la zona en la que estará la persona en el estado siguiente tomando como entrada de los datos generados con las razas actuales del algoritmo evolutivo. Con esto conseguimos acotar la zona donde realizar la mutación de los individuos de la raza seleccionados con elitismo mejorando la asociación.

En la sección 5.2.4, se han descrito el reparto de individuos, el porcentaje de la población elitista elegida, siendo del 20 % de la siguiente población directamente con individuos elitistas y un 80 % que evolucionan sobre éstos aplicando el operador genético de la mutación. Con el fin de introducir el modelo de Kalman también para evolucionar las razas se ha realizado una subdivisión de estos porcentajes de la siguiente manera:

- 10 % generados directamente con los individuos elitistas.
- 10 % generados con los individuos elitistas pero una vez aplicada la predicción de Kalman (basadas en estimaciones).

- 40 % generados con mutación térmica sobre los individuos elitistas.
- 40 % generados con mutación sobre térmica los individuos predichos por el filtro de Kalman ya aplicado.

5.2.6. Función salud

Como se ha comentado a lo largo de esta sección, todo el algoritmo evolutivo depende de la calidad de los individuos de cada raza, descartando en cada iteración aquellos con baja calidad y evolucionando la población de explotadores con aquellos prometedores que han conseguido buenos resultados.

Consideramos que un individuo tiene alta calidad si engloba bastantes puntos de la nube o nubes adyacentes pero manteniendo cierta densidad. Esto supone que los explotadores elitistas serán aquellos centrados en zonas densas del entorno las cuales, al estar trabajando sobre segmentadores de primer plano, asumimos que son objetos en movimiento, es decir, personas.

Con el fin de evaluar la calidad de los explotadores de las poblaciones de cada raza, se ha diseñado una función salud que tiene en cuenta dos criterios de modo ponderado:

- **Densidad:** es el número de puntos por unidad de volumen.
- **Completitud:** es el ratio entre el número de puntos que contiene un prisma en concreto con respecto al explotador de su raza que más puntos contiene. Éstas es una forma relativa de medir cuán bien explica un prisma la zona adyacente.

En el caso de que una raza no tenga ninguna otra cerca, consideraremos individuos de alta calidad aquellos que tienen buena densidad pero se dará más peso a la completitud. Con ello se consiguen prismas que engloban muchos de los puntos de la nube que se intenta explicar.

1. **Función salud para razas aisladas.**
   Esta función salud se aplica cuando los individuos a evaluar pertenecen a una raza que no tiene ninguna otra raza cerca de ella. Computándose de la siguiente forma:

   $$salud = densidad + 3 \cdot completitud$$

   Esta función salud da buenos resultados aunque sus evaluaciones son relativas, es decir, la función salud no se computa con respecto a la mejor solución posible (absoluta) sino que evalúa cuál de los individuos de la raza actual es el mejor candidato utilizando valores normalizados. Es precisamente por esto que el valor máximo que puede tomar la función salud es 4, siendo los parámetros de

normalización los siguientes:

$$densidad = \frac{d_i}{d_b}$$
$$completitud = \frac{c_i}{c_b}$$

Donde d_i es la densidad del individuo a evaluar, d_b la mayor densidad de los individuos de la raza, c_i el número de puntos de la nube que engloba el individuo a evaluar y c_b el número de puntos del individuo que más puntos contiene dentro de esa misma raza. Con esto se consigue dar buenas calificaciones a individuos que engloben muchos de los puntos posibles pero manteniendo cierta densidad. Se descartan también los individuos que tengan prismas muy densos pero con pocos puntos, o bien que contengan muchos puntos pero muy dispersos.

2. **Función salud para razas con vecinos.**
Cuando sí se tienen razas cercanas el enfoque es algo distinto, pretendiendo mantener el representante de la raza en zonas densas para que no evolucione con puntos que no son suyos y no generar rupturas en el seguimiento. Esta función salud se aplica justo en el caso contrario del anterior, es decir, cuando los individuos a evaluar pertenecen a una raza la cual tiene alguna otra adyacente. Al contrario que la función salud anterior, en este caso, la densidad y la completitud tienen el mismo peso:

$$salud = densidad + completitud$$

De esta manera se consigue mantener una buena relación entre densidad y completitud, siendo los individuos mejor valorados aquellos con prismas densos pero con bastantes puntos de la nube englobada por la raza. Al contrario que en la función salud anterior el valor máximo que puede tomar la calidad de una raza es 2 y se utiliza la misma normalización explicada anteriormente.

Una restricción adicional, incluída para la evolución de razas con vecinos, es limitar a la mitad del espacio de exploración volumétrico. De esta forma, se sigue evolucionando la razas en posición de la forma convencional, sin embargo, se fuerza a que el aspecto de las razas sea más persistente en el tiempo, haciendo su evolución más lenta.

5.2.7. Fiabilidad de las razas

Una vez generada una raza, empieza a evolucionar con los datos de entrada de la aplicación para detectar y seguir las personas que se encuentren en el entorno aplicando el algoritmo detallado anteriormente. Con el fin de gestionar las razas existentes y optimizar el resultado generado por el algoritmo se han introducido una serie de restricciones:

- **Mínima persistencia de la raza.**
  Dado que la salida del algoritmo de seguimiento será utilizada como entrada del módulo de alarmas, es necesario que la información que éste genera tenga una garantía mínima. Para reducir la tasa de falsos positivos, aunque internamente se estén evolucionando todas las razas existentes en el sistema, sólo exportarán como personas válidas las posiciones de las razas que lleven más de x milisegundos presentes en el algoritmo.

- **Vida de las razas.**
  Para proteger al algoritmo de seguimiento frente a oclusiones totales de alguna raza, éstas siguen evolucionando cierto tiempo tomando como referencia el modelo de seguimiento estimado por el filtro de Kalman. Este proceso se repite hasta un tiempo máximo, tras el cual la raza será eliminada si no tiene más soporte desde la nube de puntos. Durante el tiempo en el que no se tienen datos sensoriales la raza estará marcada como *predicción* y esta información se exportará junto con los datos de posición para que, en caso de ser necesario, pueda ser utilizado desde el módulo de alarmas o cualquier otro componente que utilice como entrada la salida del módulo de seguimiento.

5.3. Módulo de alarmas y situaciones de riesgo

Una vez explicados cada uno de los pasos del módulo de detección y seguimiento multipersona basado en el algoritmo evolutivo multimodal, se detallarán cómo son detectadas las situaciones de riesgo y las diferentes circunstancias que es capaz de detectar el módulo de alarmas. Para comprobar el estado del sistema y determinar si se ha producido alguna situación de riesgo se utilizan tanto la nube de puntos original como la posición de cada una de las personas que el algoritmo ha detectado así como la estela temporal de posiciones de cada una de las razas.

Las situaciones de riesgo que este módulo es capaz de detectar son las siguientes:

- **Caídas**
  El sistema es capaz de detectar, utilizando la información 3D de cada persona, si alguien se ha caído en el entorno monitorizado. Una vez detectada esta situación el sistema es capaz de enviar una alarma de forma automática, bien en el momento justo de haberse detectado dicha situación, o tras esperar un cierto tiempo prudencial para asegurarse de que la situación es realmente una caída y que no es, por ejemplo, que una persona está recogiendo algo del suelo.

- **Proximidad a zonas prohibidas**
  Se pueden definir zonas de seguridad dentro del entorno y que el sistema avise si alguna persona está cerca de ellas. Esto permite que si alguien se acerca a zonas como el balcón, la puerta de salida o el cuadro eléctrico de la casa se produzca una situación de alarma.

- **Entrada y salida de estancias**
  El sistema conoce cuántas personas hay en cada estancia, utilizando este contador

se puede saber si alguien ha entrado o salido de ellas. Esto puede ser útil para monitorizar estancias prohibidas, como puede ser la sala de limpieza o de medicinas de una residencia.

- **Ausencia de paso**
  Una posibilidad extra que ofrece el sistema es poder poder fijar situaciones de riesgo por ausencia de paso. Esto permite que el sistema entre en situación de alerta si nadie ha pasado por una determinada estancia dentro de una franja horaria prefijada. Si nadie ha pasado en toda la mañana, por ejemplo, por la sala de estar, puede ser significativo de que se está produciendo una situación peligrosa.

- **Estancia prolongada en el baño**
  Con el fin de detectar posibles situaciones peligrosas que una persona puede sufrir en el baño, sin la necesidad de estar monitorizando directamente esa estancia (preservando su intimidad), se ha diseñado una funcionalidad que es capaz de estimar si alguien ha desaparecido pasando por una puerta que lleva a una estancia especial, como puede ser el aseo. Conociendo tanto la posición de la puerta, como la trayectoria que lleva la persona, mediante la estimación de algoritmo de seguimiento, esta alarma inicia un contador cuando se detecta que alguien entra en el baño (u otra estancia). En el caso de que no abandone esta estancia en un tiempo prudencial, saltará una alarma informando de esta situación.

- **Zona Volumétrica prohibida**
  Una funcionalidad adicional, intrínseca al sistema, es la posibilidad de utilizar la propia nube de puntos en 3D ofrecida por el sensor para definir zonas tridimensionales concretas como prohibidas. A diferencia de las alarmas de zonas prohibidas, donde la situación de riesgo se define por proximidad de una persona a una zona peligrosa, las alarmas volumétricas son detectadas por intrusión de un número relevante de puntos. Definida una zona 3D prohibida, la alarma volumétrica saltará si se ha detectado que hay *algo* dentro de esta zona, por ejemplo un brazo frente a un armario de medicinas.

Cada una de estas alarmas se puede configurar y activar en rangos horarios determinados, lo que permite gran flexibilidad, permitiendo incluso utilizar alguna de estas alarmas como si se tratase de un sistema de seguridad.

5.4. Módulo gráfico

El sistema desarrollado incorpora, de manera completamente opcional, una interfaz gráfica en la que se vuelca toda la información generada (figura 5.9). Esta interfaz gráfica, basada en GTK, ofrece la posibilidad de modificar varios parámetros de visualización y un visor OpenGL, donde se representa toda la información tridimensional:

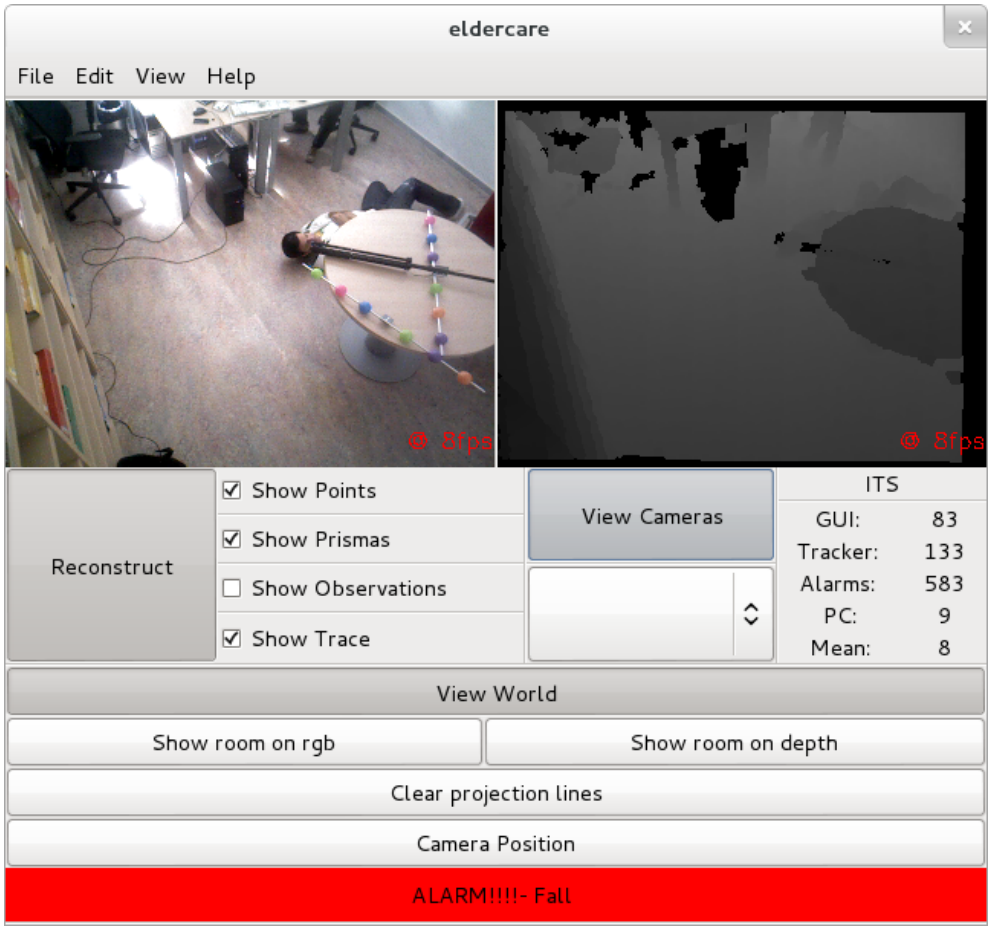


Figura 5.9: Interfaz gráfico de ElderCare.

- Nube de puntos filtrada.
- Observaciones del sistema.
- Posición actual de cada una de las personas, así como su rastro.

Este módulo se ejecuta como un módulo completamente independiente en una hebra dedicada. Toda la información que es representada se recoge del módulo de adquisición estando protegida frente a accesos concurrentes detrás de un *mutex*.

A través del propio visor se permite la conexión a los diferentes dispositivos instalados en el sistema y ver directamente tanto su imagen de color como el mapa de profundidad. A su vez, muestra una etiqueta verde si no se está produciendo ninguna situación de riesgo y en caso contrario una etiqueta roja con información de cada una de las situaciones detectadas.

5.5. Configuración del sistema

El sistema descrito define un sistema de seguimiento 3D de personas capaz de detectar situaciones de riesgo de forma totalmente autónoma. El sistema, una vez

montado, no requiere ningún tipo de intervención externa, sin embargo, sí que es necesario configurarlo adecuadamente para su correcto funcionamiento. Esta configuración se carga por la aplicación automáticamente desde un fichero XML, donde se pueden configurar los siguientes parámetros:

- Configuración de las razas
  - Tiempo mínimo que una raza debe existir en el sistema antes de ser tomada como una persona. Con el fin de reducir los falsos positivos que pueda dar el sistema y tener cierta seguridad de que no es una detección puntual espúrea.
  - Tiempo máximo que una raza puede estar evolucionando únicamente con predicciones del filtro de Kalman, sin corrección desde observaciones.
  - Tamaño de la población de individuos explotadores que contiene cada raza.
  - Número de evoluciones genéticas que se realiza a cada una de las razas por iteración.
- Configuración de los exploradores
  - Tamaño fijo que se asigna al prisma del individuo explotador para buscar posibles zonas donde crear nuevas razas.
  - Número de exploradores que se generan en cada iteración.
- Configuración de los explotadores
  - Rangos máximos sobre los que se aplica la *mutación térmica*. Es decir, cuando se le aplica ruido térmico a un individuo elegido con elitismo, los umbrales de variación que se van a fijar en cada una de las dimensiones, tanto en posición del centro del prisma como en tamaño.
- Fiabilidad
  - Número mínimo de puntos que debe contener un explorador para ser susceptible de convertirse en raza.
  - Tamaño mínimo de la envolvente de la nube de puntos de un explorador para ser susceptible de convertirse en raza.
  - Distancia mínima entre razas. Este parámetro sólo se tiene en cuenta al generar nuevas razas, no en el seguimiento.
  - Umbral para la aplicación de la función salud para razas aisladas o para razas con vecinos.

Además de estos parámetros del sistema, es necesario fijar las situaciones de riesgo que se quieren controlar. Estas alarmas también se cargan utilizando ficheros XML y se definen de forma sencilla:

- Caídas
  Sólo es necesario fijar el umbral de altura sobre el que se discierne si una persona está en el suelo o de pié y el tiempo de seguridad para controlar situaciones aisladas (persona recogiendo algo del suelo).

- Alarmas basadas en posición
  Es necesario fijar la posición de riesgo (x,y,z) y un radio alrededor del punto que será la *zona prohibida*.

- Alarmas de ausencia o de paso
  Para fijar este tipo de alarmas es necesario fijar la zona monitorizada, la franja horaria en la que estará activa la alarma y si la alarma es positiva o negativa (presencia o ausencia), es decir, si la alarma es de paso o de ausencia.

- Alarmas volumétricas
  En este caso es necesario fijar la zona prohibida en 3D utilizando la misma primitiva que con las personas, un prisma, junto con la franja horaria que debe estar activa.

5.6. Acceso remoto al sistema

En la práctica es necesario poder acceder al sistema de forma remota ya que, en un entorno teleasistencial, estará instalado en la propia casa del sujeto monitorizado. Este acceso al sistema debe ofrecer dos funcionalidades básicas. Por una parte es necesaria una herramienta que permita la conexión remota al sistema y visualizar la información de todos los sensores, así como el estado del sistema para, en caso necesario, verificar el estado de una situación de riesgo. Para dar solución a esta necesidad se ha implementado un componente llamado *RemoteViewer* (figura 5.10). Este componente muestra en se visor un histórico con las últimas alarmas que ha detectado el sistema.

Por otra parte, puede ser necesario modificar los parámetros de funcionamiento del sistema desde una localización externa para, por ejemplo, incluir una nueva alarma en tiempo de ejecución. Sin esta funcionalidad, cada vez que surja la necesidad de incluir una nueva alarma sería necesario desplazarse físicamente hasta el domicilio donde esté montado el sistema y hacer los cambios manualmente. Para ello, se ha creado el componente *RemoteConfigurator* el cual permite transmitir por completo el fichero de configuración del sistema, pudiéndolo cargar en tiempo de ejecución por la aplicación sin necesidad de parar el sistema.

Se han implementado dos nuevos interfaces ICE en el componente *ElderCare* con el fin de dar soporte a estas dos nuevas herramientas. En la figura 5.11, se puede apreciar el esquema de conexión entre los componentes remotos y el componente principal.

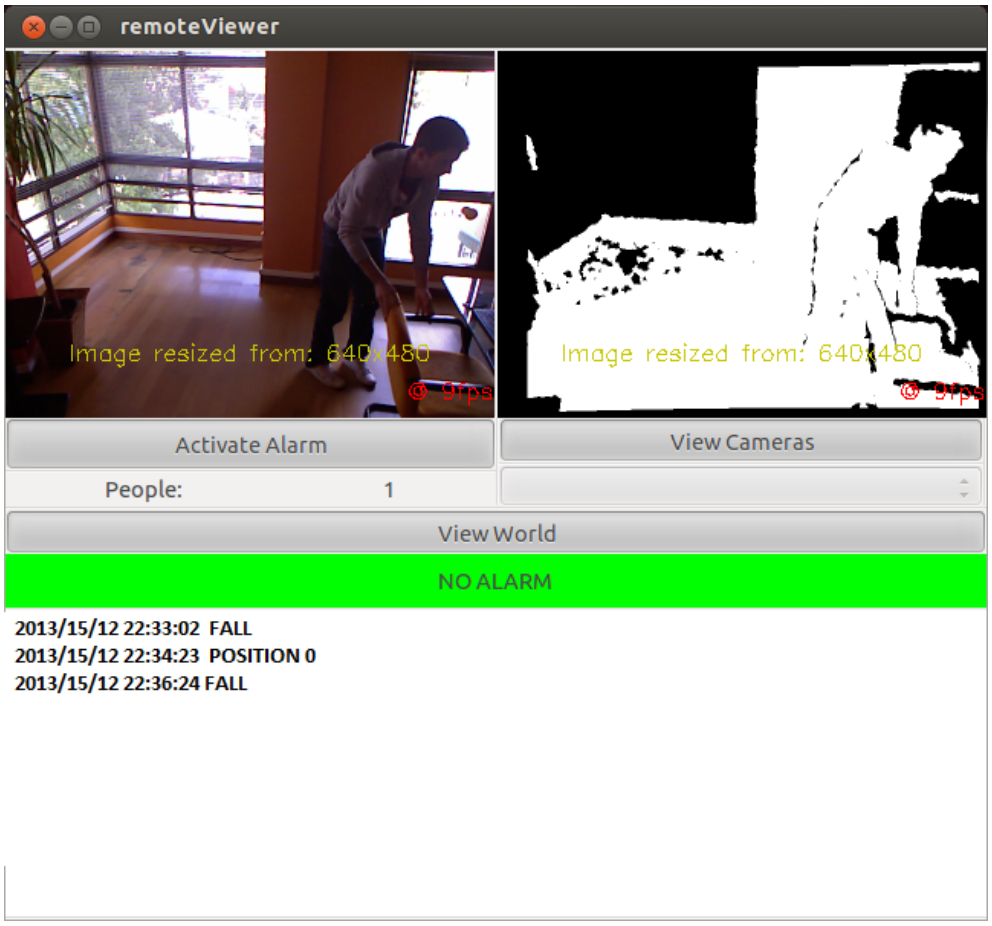


Figura 5.10: RemoteViewer

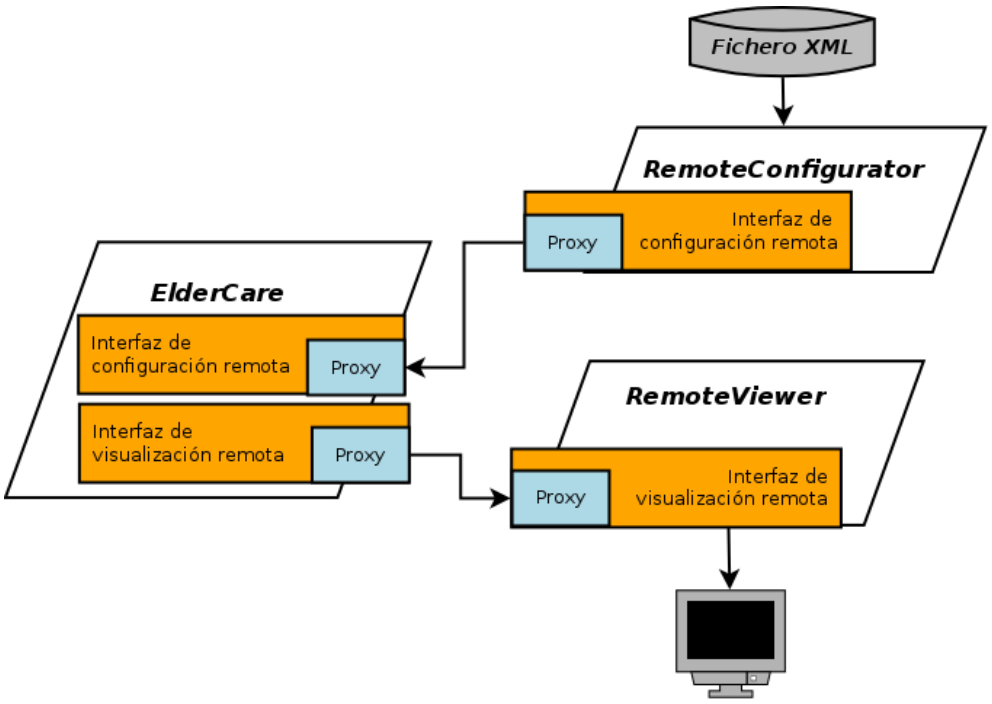


Figura 5.11: Conexión con componentes de acceso remoto

Capítulo 6
Experimentación y resultados

En este apartado se detallan las evaluaciones realizadas en las diferentes configuraciones del sistema. Se presentan los resultados obtenidos en los experimentos realizados en entornos reales, llevados a cabo con el fin validar experimentalmente tanto el conjunto de componentes como las librerías desarrollados durante este proyecto.

El entorno de pruebas utilizado consta de dos ordenadores de sobremesa convencionales. Uno de ellos muy potente con cuatro núcleos a 3.00GHz, y el segundo con dos núcleos a 1.8GHz. Las especificaciones más relevantes de ambos ordenadores están disponibles en las tablas 6.1 y 6.2 respectivamente. El sistema operativo utilizado es la última versión disponible de Debian Stable[1].

6.1. Segmentador de fondo

El componente *BackgroundSubstractor*, el cual implementa la segmentación de primer plano utilizada en este proyecto, integra la librería *DepthLib* para realizar un filtrado de los datos ofrecidos por el sensor y un muestreado dependiente de la distancia, ambos detallados en la sección 4.4. En este componente se puede configurar el tipo de muestreado aplicado y activar o desactivar el filtrado temporal de las imágenes, implementado dentro de *DepthLib*.

Con el fin de comprobar cómo se comportan las diferentes configuraciones disponibles en el componente *BackgroundSubstractor*, se ha medido el tiempo que tarda en realizar una iteración. Para unificar las pruebas llevadas a cabo, se han realizado las evaluaciones sobre exactamente los mismos datos utilizando los componentes *Recorder*

[1] https://www.debian.org/releases/wheezy/

	Descripción
Procesador	Intel Core i5, 4x3.00GHz
Memoria RAM	8GB a 1600 MHz
Disco Duro	SSD 550 MB/s de lectura y 510 MB/s de escritura

Tabla 6.1: Especificaciones del ordenador A

	Descripción
Procesador	Intel Core i3, 2x1.80 (con hyperthreading)
Memoria RAM	4GB a 1600 MHz
Disco Duro	SSD 550 MB/s de lectura y 550 MB/s de escritura

Tabla 6.2: Especificaciones del ordenador B

Tipo de filtro	**Tipo de Muestreo**	$\bar{\mathbf{X}}$ **(ms)**	σ **(ms)**
Ninguno	Uniforme	11.37	2.94
Ninguno	Dependiente de la distancia	11.40	2.81
Dinámico	Uniforme	23.67	3.30
Dinámico	Dependiente de la distancia	24.40	3.38

Tabla 6.3: Rendimiento de los agrupamientos

y *Replayer*, con un log de 15 minutos de duración. Los resultados, calculados sobre el ordenador A del entorno de pruebas, son los presentados en la tabla 6.3. La configuración deseable para nuestro sistema es la que combina tanto en filtrado como el muestreado dependiente de la distancia, ya que aporta una estabilización de la imagen fuente y un muestreo con una densidad estable, independientemente de la distancia del objeto al sensor. Los resultados obtenidos son prometedores, ya que solamente necesita unos 24 milisegundos para ejecutar una iteración completa.

La diferencia, en términos de tiempo de ejecución, entre la configuración básica, donde se emplea un muestreado homogéneo sin ningún tipo de preprocesado, y la configuración más avanzada, con un muestreado dependiente de la distancia y el preprocesado diseñado en este proyecto, es tan solo de 13ms. Sin embargo, las ventajas de la configuración avanzada, y por lo que merece la pena este mayor tiempo de ejecución, son numerosas, consiguiendo una mayor estabilidad en los datos del sensor y un muestreado mucho más representativo de la realidad.

6.2. Componente de reproducción enlatada

Es indispensable que los componentes de reproducción enlatada no consuman demasiada CPU. Es necesario que ofrezca capacidad de cómputo suficiente al resto de componentes que utilizan los datos que éste sirve. Con el fin de comprobar la carga generada por el componente de reproducción *Replayer*, se ha lanzado este componente sobre un *log* previamente almacenado con una configuración de 4 cámaras. Los resultados obtenidos son los presentados en la tabla 6.4.

Estos datos están normalizados por núcleos, donde cada núcleo constituye un 100 %, por lo que la carga máxima del ordenador de pruebas A sería un 400 % y el mismo valor para el ordenador B, ya que, aún sólo teniendo 2 núcleos, es capaz de ejecutar dos hilos de ejecución en cada uno de ellos. Estos resultados reflejan que el componente *Replayer* usa menos del 4 % de la capacidad de cómputo total del ordenador A y menos

	Replayer y Recorder		OpenniServer1		OpenniServer2	
PC	$\bar{\mathbf{X}}$	σ	$\bar{\mathbf{X}}$	σ	$\bar{\mathbf{X}}$	σ
Ordenador A	14.6 %	5 %	9.3 %	1 %	9 %	1 %
Ordenador B	30 %	4 %	26 %	5 %	22 %	7 %

Tabla 6.4: Carga de CPU generada por el componente Replayer

del 8 % en el ordenador B. Cumpliendo con creces el requisito de ser un componente que se pueda ejecutar con baja carga computacional. Además, la carga generada por este componente es ligeramente menor que la generada con los servidores reales, 14 % frente a 18 % para el ordenador de pruebas A, y 30 % frente a 48 % en el ordenador de pruebas B.

Sin embargo, en el caso del componente *Recorder*, su coste computacional no es crítico, siempre y cuando sea capaz de mantener una frecuencia de grabación ágil. La limitación de este componente no viene fijada por la CPU, ya que el procesamiento que realiza es muy ligero, sino por los tiempos de volcado a disco de todo el flujo de datos. Utilizando la misma configuración que la empleada para evaluar al componente *Replayer*(4 cámaras), grabando imágenes a una frecuencia de 15 fps, el tamaño de los logs generados es de 280MB por cada minuto de grabación.

6.3. Validación experimental del seguidor de personas en 3D

Con el fin de validar el algoritmo de seguimiento desarrollado se ha realizado, por una parte, una evaluación de su funcionamiento junto con el módulo de alarmas, comprobando el funcionamiento global del sistema en dos entornos reales donde se producen una serie de situaciones de riesgo. Por otra parte, se ha evaluado su capacidad de seguir a una serie de personas deambulando por el entorno monitorizado. Estas tres evaluaciones del sistema se han realizado de manera automática, utilizando los componentes de reproducción enlatada desarrollados en este proyecto. Gracias a estas herramientas, se puede reproducir, de forma automática y en bucle, un mismo *log* con datos reales del que se conocen de antemano las situaciones de riesgo que en él se producen o el número de personas que hay realmente en el entorno.

6.3.1. Escenario de pruebas 1: salón-comedor de un domicilio particular

El primer escenario de pruebas es un salón-comedor de un domicilio particular (figura 6.1), donde se han colocado dos sensores con el fin de monitorizar el mayor espacio posible. En la figura 6.2, se puede apreciar la representación completa con los datos obtenidos de ambos sensores, donde la vista obtenida con el sensor 1 son los datos representados en la figura 6.3(a), y la vista del sensor 2 los de la figura 6.3(b).

En esta primera validación, a modo ilustrativo, se han recreado 4 situaciones de

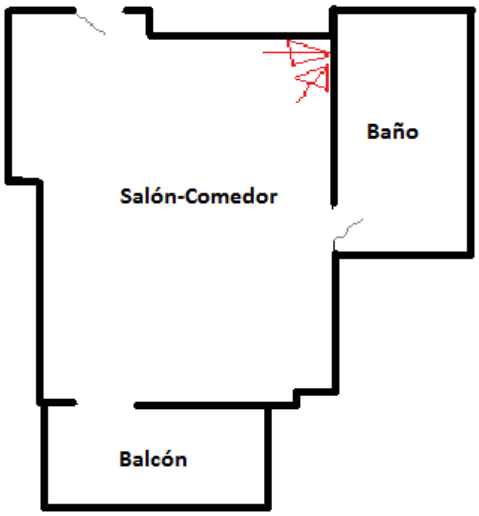


Figura 6.1: Mapa del entorno de pruebas 1

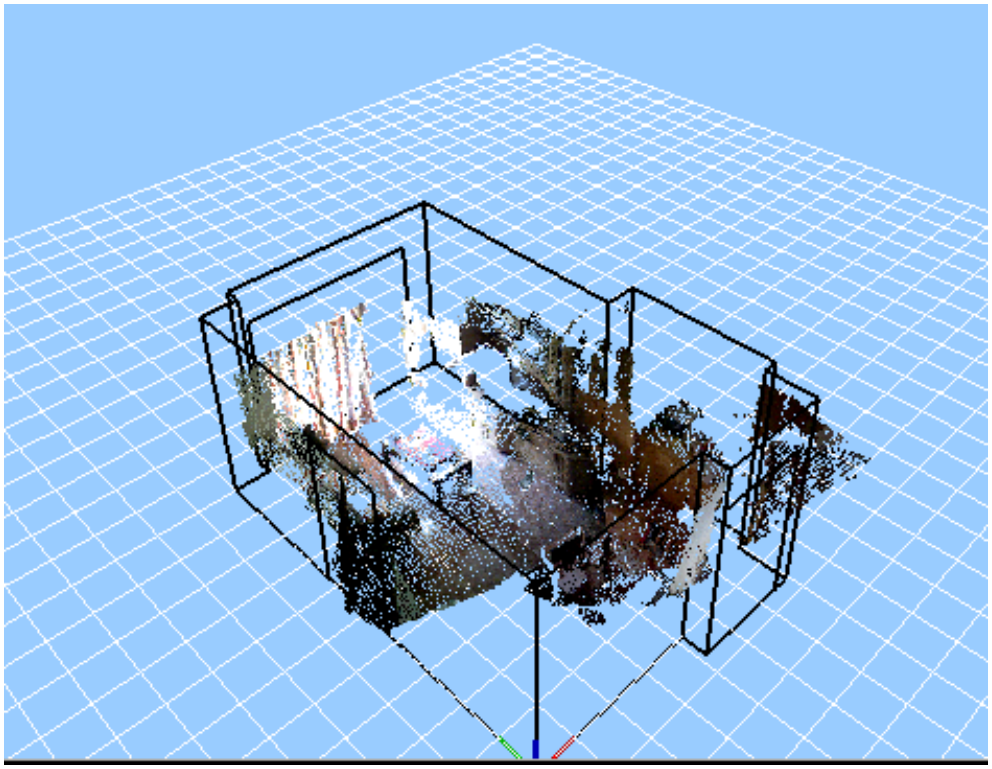


Figura 6.2: Reconstrucción del entorno de pruebas 1

(a) (b)

Figura 6.3: Rango de visión de los sensores en el entorno 2: (a) Imagen de color del sensor 1 (b) Imagen en color del sensor 2

riesgo junto con una alarma de detección de presencia. Las situaciones de riesgo corresponden a dos caídas (figuras 6.4(a), 6.4(b) y 6.4(c)), una situación en la que el sujeto se acerca a la puerta del balcón (figura 6.4(d)) y por último una situación en la que el sujeto ha pasado demasiado tiempo en el baño. La primera de las caídas se produce con una oclusión parcial, dado que el sujeto cae detrás de una mesa (figura 6.4(a)), mientras que la segunda caída se produce en una zona de buena visibilidad donde el sujeto es captado por ambos sensores (figuras 6.4(b) y 6.4(c)). La zona determinada como peligrosa, la puerta del balcón, está situada a unos 6.30 metros del sensor en una zona donde incide directamente la luz del sol (figura 6.4(d)). De forma simultánea, cada vez que se detecta una persona en la escena se lanzará una alarma de presencia. Los parámetros usados para la configuración del sistema en este experimentos son los incluidos en la tabla 6.5.

Gracias al componente *Replayer*, se ha reproducido el *log* detallado, de 92 segundos de duración, un total de 3.000 veces y los resultados obtenidos se han volcado en la tabla 6.6. La tasa de aciertos obtenida es el 100 % y no se ha producido ningún falso positivo. La única situación a destacar en este experimento es la discontinuidad que se ha producido en la alarma de posición peligrosa. Justo en esa zona de la habitación, la puerta al balcón, entra mucha luz solar. Esto hace que el sensor no funcione de forma adecuada, introduciendo más ruido en el sistema de lo habitual e incluso generando zonas en las que el sensor no es capaz de captar distancia. En todas las situaciones, el sistema ha sido capaz de detectar que el sujeto ha llegado a la zona peligrosa. Sin embargo, en ciertas ocasiones, no era capaz de mantener de forma correcta la ubicación del sujeto. Lo cual hacía creer al sistema que el sujeto había salido de la zona peligrosa y vuelto a entrar en ella en pocas interacciones. Esta situación se registró en 556 ocasiones.

Estos resultados, aunque obtenidos sobre una repetición de cuatro situaciones de riesgo, son satisfactorios y dan una idea inicial del potencial del sistema desarrollado.

(a)

(b)

(c)

(d)

Figura 6.4: (a) Situación de riesgo 1. (b)Situación de riesgo 2 (sensor 1). (c)Situación de riesgo 2 (sensor 2). (d)Situación de riesgo 3

Configuración de las razas	
Tiempo mínimo de existencia antes de clasificar como persona	200ms
Tiempo máximo de persistencia sin observaciones	500ms
Número de explotadores de cada población	50
Número de evoluciones en cada iteración	20
Configuración de los exploradores	
Tamaño inicial de los exploradores	300x300x300(mm)
Número de exploradores generados por iteración	5
Configuración de los explotadores	
Rangos de aplicación para la mutación térmica	
En posición	(400,400,400) (mm)
En volumen	(50,50,150) (mm)
Configuración de la fiabilidad del sistema	
Número de puntos de mínimo para generación de raza	100
Tamaño mínimo de la envolvente de un explorador	300x300x300(mm)
Distancia mínima entre nuevas razas	400
Umbral de distancia para la función salud	1000
Configuración del BackgroundSubstractor	
Configuración para el sensor 1	
Configuración del filtrado	
Tamaño del buffer	4
Umbral de variación	7
Configuración del muestreo	
Número de capas	6
Salto entre capas	0.5
Distancia máxima	6000 (mm)
Configuración para el sensor 2	
Configuración del filtrado	
Tamaño del buffer	4
Umbral de variación	7
Configuración del muestreo	
Número de capas	7
Salto entre capas	0.4
Distancia máxima	7000 (mm)

Tabla 6.5: Configuración del sistema empleado en la evaluación del entorno 1

Situación	**Verdaderos Positivos**	**Falsos negativos**
Caída 1	3000	0
Caída 2	3000	0
Posición peligrosa	3000	0
Estancia prolongada	3000	0
Presencia	3000	0

Tabla 6.6: Resultados de la detección en las diferentes situaciones de riesgo

	Ordenador A				**Ordenador B**			
	Tiempos		**% CPU**		**Tiempos**		**% CPU**	
	$\bar{\mathbf{X}}(ms)$	$\sigma(ms)$	$\bar{\mathbf{X}}(\%)$	$\sigma(\%)$	$\bar{\mathbf{X}}$(ms)	σ(ms)	$\bar{\mathbf{X}}(\%)$	$\sigma(\%)$
Background Substractor1	24.55	3.29	35	14	66.23	20.29	83	18
Background Substractor2	23.08	4.25	30	15	73.36	14.63	75	14
ElderCare	7.16	7.44	25	18	27.89	20.60	45	16
Alarmas	36	10	3	1	85	35	4	1

Tabla 6.7: Tiempo de CPU necesario para cada componente

De forma simultánea a las pruebas de validación, se ha realizado una evaluación del rendimiento de cada uno de los componentes implicados. Se ha medido tanto el tiempo que tarda en realizar cada iteración como la carga de CPU que generan. Estos resultados, expresados en la tabla 6.7, indican que el sistema desarrollado es capaz de ejecutarse a unas 40 iteraciones por segundo en el ordenador A, y a unas 13 en el ordenador B.

Como evaluación cualitativa de cómo se comporta el sistema y la fiabilidad que presenta desde el punto de vista computacional, se ha probado el sistema, funcionando las 24 horas del día, durante un mes completo en este entorno. Durante todo este tiempo el sistema ha funcionando de manera adecuda sin generar fallos que hayan hecho cerrar alguno de los componentes del mismo.

6.3.2. Escenario de pruebas 2: residencia de ancianos

El segundo entorno utilizado para la validación del sistema es una habitación de una residencia de ancianos. Esta habitación consta de un dormitorio con dos camas y un baño (figura 6.5). Al igual que en el primer experimento de validación, se usarán dos sensores para cubrir un mayor espacio. La representación en 3D de los datos capturados por ambos sensores se puede apreciar en la figura 6.6, donde el ángulo de visión del primer sensor captura datos en la perspectiva de la figura 6.7(a), y el ángulo del segundo sensor los de la figura 6.7(b).

En este segundo escenario se han planteado una serie de situaciones ilustrativas de riesgo tal y como se hizo en el primer experimento. En este caso, las situaciones que se han recreado son tres diferentes posiciones de caídas, una alarma de estancia prolongada en el baño y por último, también se ha evaluado el detector de presencia. La primera caída se produce en la puerta del baño (figura 6.8(a)), a unos 9 metros del sensor que capta esa zona. Una segunda caída, producida entre las dos camas de la habitación, zona donde se solapa el campo de visión de ambos sensores (figuras 6.8(b) y 6.8(c)). Y una tercera caída, al lado de una de las camas y en una zona donde incide directamente la luz solar (figura 6.8(d)). Los parámetros de configuración utilizados para realizar esta evaluación son los presentados en la tabla 6.8.

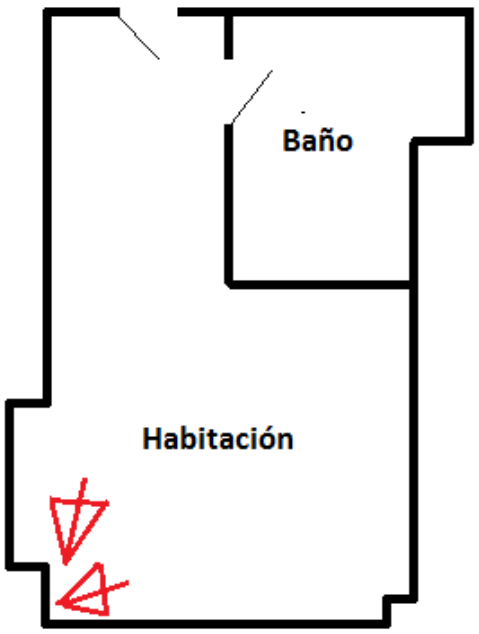


Figura 6.5: Mapa del entorno de pruebas 2

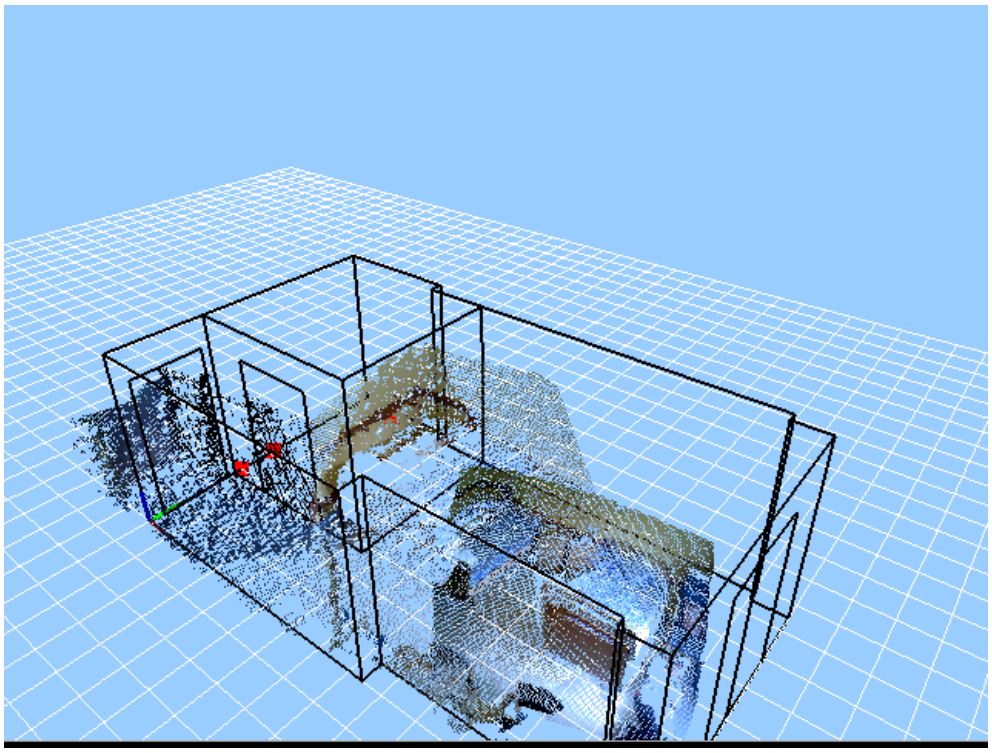


Figura 6.6: Reconstrucción del entorno de pruebas 2

(a) (b)

Figura 6.7: Rango de visión de los sensores en el entorno 2: (a) Imagen de color del sensor 1 (b) Imagen en color del sensor 2

Configuración de las razas	
Tiempo mínimo de existencia antes de clasificar como persona	300ms
Tiempo máximo de persistencia sin observaciones	500ms
Número de explotadores de cada población	50
Número de evoluciones en cada iteración	20
Configuración de los exploradores	
Tamaño inicial de los exploradores	300x300x300(mm)
Número de exploradores generados por iteración	2
Configuración de los explotadores	
Rangos de aplicación para la mutación térmica	
En posición	(300,300,300) (mm)
En volumen	(50,50,150) (mm)
Configuración de la fiabilidad del sistema	
Número de puntos de mínimo para generación de raza	120
Tamaño mínimo de la envolvente de un explorador	300x300x300(mm)
Distancia mínima entre nuevas razas	400
Umbral de distancia para la función salud	1000
Configuración del BackgroundSubstractor	
Configuración para el sensor 1	
Configuración del filtrado	
Tamaño del buffer	4
Umbral de variación	7
Configuración del muestreo	
Número de capas	10
Salto entre capas	0.3
Distancia máxima	10000 (mm)
Configuración para el sensor 2	
Configuración del filtrado	
Tamaño del buffer	4
Umbral de variación	7
Configuración del muestreo	
Número de capas	10
Salto entre capas	0.3
Distancia máxima	10000 (mm)

Tabla 6.8: Configuración del sistema empleado en la evaluación del entorno 2

(a)

(b)

(c)

(d)

Figura 6.8: (a) Situación de riesgo 1. (b)Situación de riesgo 2 (sensor 1). (c)Situación de riesgo 2 (sensor 2). (d)Situación de riesgo 3

Situación	Verdaderos Positivos	Falsos negativos
Caída 1	3000	0
Caída 2	3000	0
Caída 3	3000	0
Estancia prolongada	2759	241
Presencia	3000	0

Tabla 6.9: Resultados de la detección en las diferentes situaciones de riesgo

	Ordenador A				**Ordenador B**			
	Tiempos		**% CPU**		**Tiempos**		**% CPU**	
	$\bar{\mathbf{X}}(ms)$	$\sigma(ms)$	$\bar{\mathbf{X}}(\%)$	$\sigma(\%)$	$\bar{\mathbf{X}}$(ms)	σ(ms)	$\bar{\mathbf{X}}(\%)$	$\sigma(\%)$
Background Substractor1	25.53	4.01	38	12	60.32	21.29	84	20
Background Substractor2	22.18	4.37	30	12	72.65	12.65	74	16
ElderCare	7.19	8.02	25	20	25.68	21.65	43	13
Alarmas	35	12	3	1	86	30	3	1

Tabla 6.10: Tiempo de CPU necesario para cada componente

Utilizando el componente *Replayer*, se ha reproducido el log que contiene las situaciones descritas anteriormente, de 88 segundos de duración, un total de 3.000 veces. En la tabla 6.9, se pueden observar el resultado de las alarmas generadas por el sistema desarrollado. Como se puede apreciar, se ha conseguido una tasa de aciertos en caídas y presencia del 100 %, mientras que en la detección de estancias prolongadas en el baño es del 91.96 %. Los fallos en la detección de esta última alarma son debidos a que la entrada al baño está muy alejada del sensor, cerca de los 9 metros. A esta distancia los datos ofrecidos por el sensor son muy ruidosos. Para el cálculo de la situación de estancia prolongada en el baño, es necesario tener datos fiables tanto de la posición como de la trayectoria. Esta alarma entra en un estado de incertidumbre cuando se ha detectado que un sujeto ha desaparecido en dirección hacia el baño. Una vez detectada esta situación, se activa un contador y si la persona no sale del baño en un tiempo determinado se desencadena la alarma final. El problema principal es que los datos ruidosos hacen difícil el cálculo de la trayectoria real, originando que, en algunas situaciones, el algoritmo no detecte que la trayectoria fuese la del baño y por lo tanto descarte la alarma.

De forma análoga al primer experimento de validación, simultáneamente a la evaluación de la detección de situaciones de riesgo, se han realizado medidas tanto de tiempo de ejecución como de consumo de CPU de cada uno de los componentes del sistema. Tal y como se esperaba, los resultados son similares a los obtenidos en el experimento anterior, consiguiendo alrededor de 40 iteraciones por segundo con el ordenador de pruebas A, y unas 13 para el ordenador de pruebas en el entorno B.

Este sistema está siendo evaluado actualmente en la residencia de ancianos, donde lleva funcionando sin problemas las 24 horas del día aproximadamente 5 semanas, lo

cual nos ha servido en gran medida para depurar partes del algoritmo desarrollado.

6.3.3. Seguimiento de varias personas.

Para este último experimento sólo se ha utilizado el módulo de seguimiento ya que es el encargado de seguir a las personas que han sido detectadas en el entorno a lo largo del tiempo. En esta evaluación se ha empleado el mismo escenario de pruebas utilizado el primer experimento, el salón comedor de un domicilio particular. En este caso se grabó un log de 50 segundos de duración en la que se encuentran 3 personas deambulado por el entorno (figura 6.9). En un primer momento la estancia está completamente vacía (figuras 6.10(a) y 6.10(b)) y de una en una van entrando las tres personas al escenario (figuras 6.11(a), 6.11(b) y 6.11(c)). Para comprobar la capacidad de detección y seguimiento del sistema se ha verificado el número de personas que está detectando y siguiendo en 3D en cada momento. La situación real (verdad absoluta) es la representada en la figura 6.12, con una persona que entra en el instante 80, otra en el 120 y la tercera en el 180. A partir de ese momento las tres personas deambulan libremente hasta que una de ellas abandona la estancia en el instante 420.

Para comprobar el funcionamiento del sistema, al igual que en los experimentos anteriores, se ha alimentado el algoritmo con el mismo fichero log de datos reales utilizando las herramientas de reproducción enlatada un total de 3.000 veces. Los resultados obtenidos están representados en la figura 6.13, donde se puede ver un resultado acorde con los datos reales. A la hora de detectar personas existe cierta variación entre iteraciones. Este fenómeno es intrínseco al proceso de detección utilizado, basado en exploradores con cierto factor aleatorio, por lo que en unas evaluaciones puede tardar más iteraciones en conseguir un candidato que cumpla los requisitos que en otras.

En cuanto a la persistencia del algoritmo frente a oclusiones, es capaz de superar oclusiones temporales producidas durante cortos periodos de tiempo, como sucede entre los instantes 210 y 240, donde se produce una oclusión casi completa de uno de los sujetos con una duración de 3 segundos 6.14. Sin embargo, entre los instantes 300 y 380 se produce una oclusión exactamente igual que la anterior, pero esta vez con una duración de 8 segundos en la que, en algunos casos, el algoritmo no es capaz de mantener el seguimiento, descartando el sujeto y creando uno nuevo cuando vuelve a ser perceptible por el algoritmo. Esta última situación se ha producido un total de 300 veces de las 3.00 evaluaciones totales.

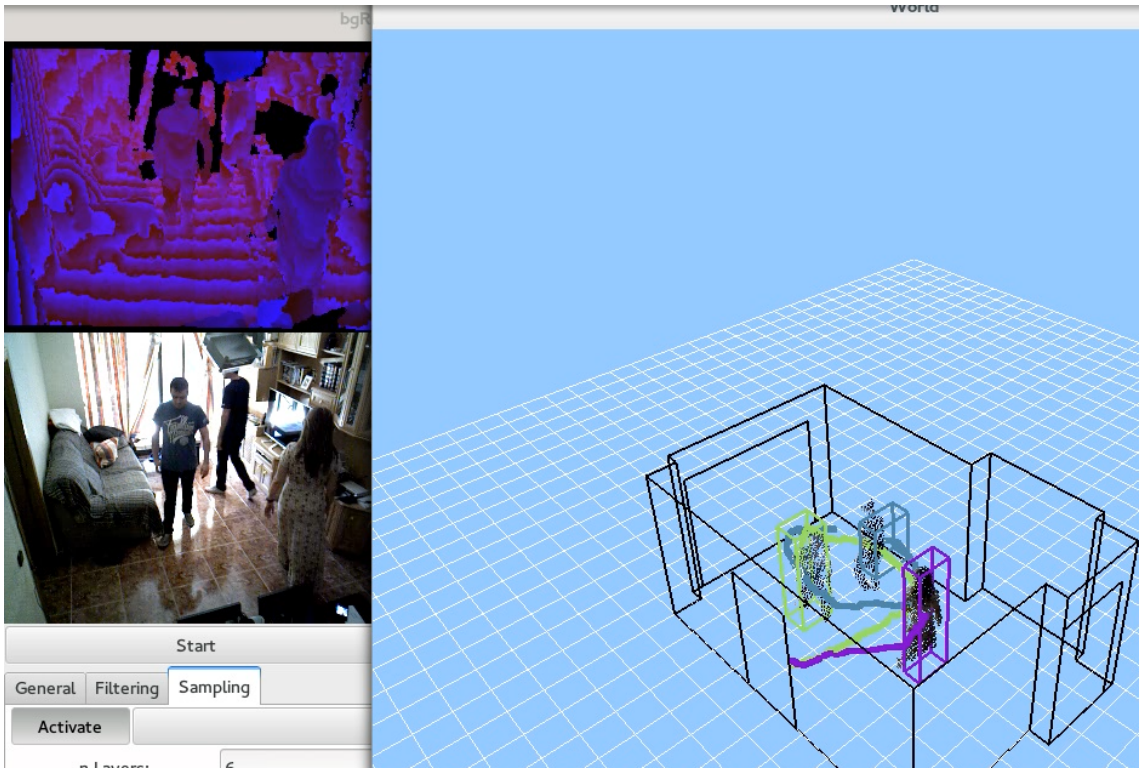


Figura 6.9: Representación del seguimiento de múltiples personas.

(a) (b)

Figura 6.10: (a) Vista del sensor 1 (b) Vista del sensor 2

(a) (b)

(c)

Figura 6.11: (a) Entrada de la primera persona (b) Entrada de la segunda persona (c) Entrada de la tercera persona

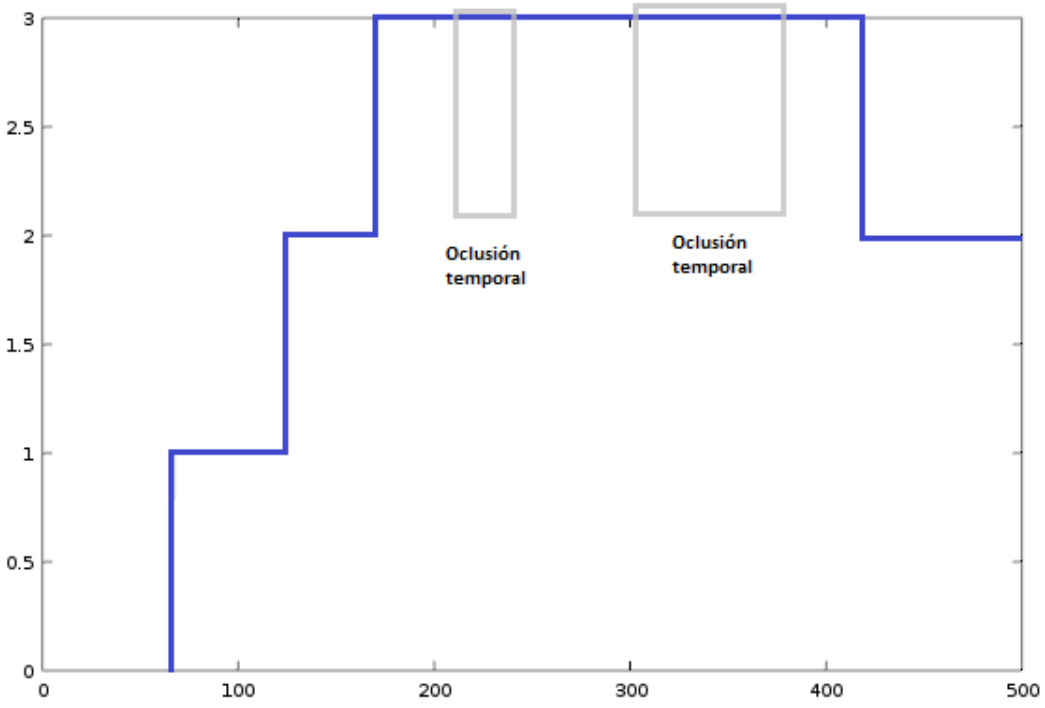


Figura 6.12: Datos reales del número de personas

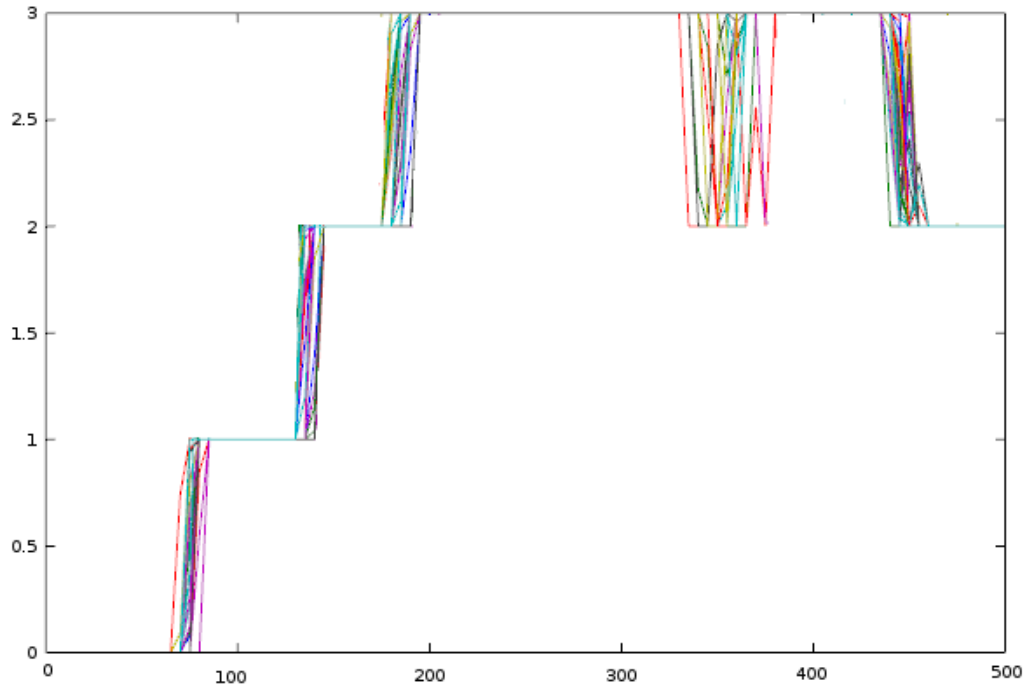


Figura 6.13: Resultados de la evaluación del seguimiento de varias personas

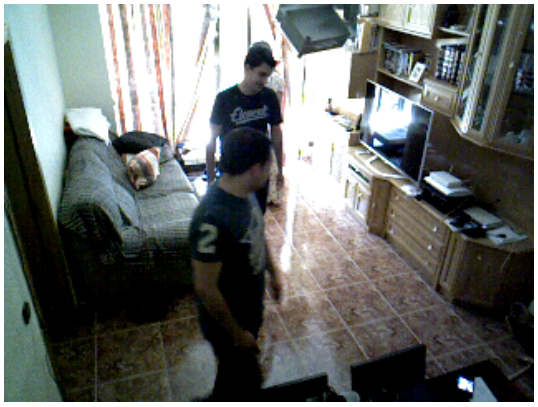


Figura 6.14: Ejemplo de oclusión de una de las personas.

Capítulo 7
Conclusiones y trabajos futuros

En los capítulos anteriores se ha realizado una amplia descripción del problema abordado y de las soluciones que se han desarrollado. Así mismo, se han expuesto los resultados obtenidos en los diversos experimentos llevados a cabo. En este capítulo se presentan las principales conclusiones del trabajo y se propondrán una serie de líneas futuras de investigación y desarrollo, planteadas a partir del conocimiento generado en este trabajo fin de máster.

7.1. Conclusiones

Tras un desarrollo de más de 13.000 líneas de código, divididos entre las diferentes librerías y componentes desarrollados, se ha conseguido dar solución a cada uno de los objetivos planteados en el capítulo 2. Con todo ello, se ha conseguido desarrollar un sistema completo capaz de detectar y seguir personas en 3D, basado en un algoritmo evolutivo multimodal. Además, a partir de las posiciones estimadas por el algoritmo de seguimiento, el propio sistema es capaz de detectar una serie de situaciones de riesgo de forma eficiente y robusta.

A continuación se repasan los distintos subobjetivos planteados, con el fin de recapitular la solución que se ha aportado a cada uno de ellos:

1. El primer subobjetivo del proyecto se centra en dar una solución a la segmentación de primer plano utilizando sensores RGB-D. Esta funcionalidad se ha implementado dentro de un componente denominado *Foreground2PointCloud*, explicado con detalle en la sección 4.3. Este componente es capaz de aprender el mapa de profundidad estático del entorno y segmentar aquellos datos que no concuerden con él, es decir, los objetos en movimiento.

   Con el fin de mejorar la usabilidad de este componente, así como su rendimiento, surgió la necesidad de crear una librería que ofreciera cierta funcionalidad relacionada con mapas de profundidad. Esta librería, llamada *DepthLib*, incorpora herramientas útiles para el segmentador de primer plano. Algunas de estas herramientas son el filtrado temporal de las imágenes (sección 4.4.2) y un muestreado de los datos dependiente de la distancia (sección 4.4.1).

2. El segundo subobjetivo consistía en diseñar y desarrollar un algoritmo que fuese capaz, por una parte, detectar personas en el entorno, y por otra parte, seguirlas a lo largo del tiempo. Para ello, se ha desarrollado un sistema basado en de un algoritmo evolutivo multimodal, explicado en detalle en el capítulo 5. En este capítulo se explican todos los pasos de este sistema así como las primitivas de evolución y seguimiento utilizadas, así como la función salud diseñada y todos los elementos empleados.

3. El tercer subobjetivo plantea la necesidad de, en función de la posición estimada de las personas, detectar una serie de situaciones peligrosas. Para ello, se ha desarrollado un módulo de alarmas, destallado en la sección 5.3, que es capaz de detectar caídas, proximidad a zonas peligrosas, entrada y salida de estancias, ausencia de paso, estancias prolongadas en el baño e intrusiones en zonas volumétricas prohibidas.

4. En el cuarto subobjetivo se plantea la creación de las herramientas necesarias para crear un sistema de reproducción enlatada. Con los componentes *Recorder* y *Replayer*, detallados en la sección 4.2, se pueden grabar una serie de datos sensoriales en disco y posteriormente reproducirlos como si se tratase del sensor original. Estas herramientas posibilitan la creación de un entorno de pruebas en el que validar sistemas de forma rigurosa y automática.

5. En el último subobjetivo se plantea llevar a cabo un proceso de validación experimental del sistema desarrollado. Para ello, se ha evaluado el sistemas sobre datos recogidos en dos entornos reales (capítulo 6). El primer entorno es salón de un domicilio particular, en la que se producen cuatro situaciones de riesgo diferentes. El segundo entorno es una habitación de una residencia de ancianos donde también se producen cuatro situaciones que son necesarias controlar.

7.1.1. Requisitos cumplidos

Los requisitos que debe cumplir el sistema desarrollado en este trabajo son los detallados en la sección 2.2.

Como se había establecido desde el comienzo del trabajo, la arquitectura software sobre la que se basa todo el desarrollo es *JdeRobot*. El hecho de haber utilizado esta plataforma como base para el sistema desarrollado, permite que todos los componentes se puedan disponer de forma distribuida, utilizando la red como vía de comunicación entre ellos. Tanto el desarrollo de las librerías, como de los componentes, se ha realizado por completo utilizando el lenguaje de programación C++.

Los componentes desarrollados específicamente para el sistema de teleasistencia son, por una parte, el segmentador de primer plano (*BackgroundSubstractor*) y, por otra parte, el componente *ElderCare*. Este segundo componente incorpora el seguimiento y detección de personas así como un módulo específico con un detector de situaciones de riesgo.

Durante la fase de experimentación, se han realizado pruebas del rendimiento general del sistema. Los resultados de estas pruebas sostienen que el sistema es capaz de ejecutarse de forma fluida. Uno de los ordenadores utilizados en el entorno de pruebas es capaz de ejecutar todo el sistema a 40 iteraciones por segundo, y en el segundo pc, menos potente, a 14, siendo ambos ordenadores convencionales.

Tras los resultados de la validación experimental, se puede afirmar que el sistema es fiable y robusto en la detección de situaciones de riesgo cumpliendo con los requisitos planteados inicialmente.

Todo el sistema desarrollado, al estar basado en comunicaciones ICE, es fácilmente integrable en prácticamente cualquier sistema informático. Esto se debe a las facilidades que ofrece este *middleware* de comunicaciones, ya que tiene soporte multiplataforma y multilenguaje.

7.1.2. Descripción software

El sistema desarrollado en este trabajo fin de máster, consta de una serie de componentes y librerías que han sido detallados a lo largo del presente proyecto y que son el aporte fundamental de este trabajo:

- *ParallelIce*
  Esta librería, con 428 líneas de código, ofrece un acceso en paralelo a una serie de sensores permitiendo paliar diversos problemas de sincronización y latencia presentes en sistemas distribuidos.

- *Recorder y Replayer*
  Estos dos componentes forman el sistema de reproducción enlatada desarrollado. El componente *Recorder*, con 1159 líneas de código, es capaz de captar datos de varios sensores y volcarlos a disco de una forma estructurada. Por su parte, *Replayer*(1085 líneas de código), se encarga de leer los datos del disco y ofrecerlos utilizando el mismo formato en el que se sirven los datos del sensor original. De esta forma, el cliente no tiene conocimiento de quién le está sirviendo datos, si *Replayer* o del propio sensor.

- *DepthLib*.
  Esta librería, con 975 líneas de código, incorpora la funcionalidad implementada para tratar mapas de profundidad, tanto para hacer filtrados temporales como para aplicar muestreados dependientes de la distancia.

- *BackgroundSubstractor*
  Este componente, encargado de aplicar una segmentación de primer plano a mapas de profundidad, consta de 1655 líneas de código. Una vez realizada la segmentación, convierte los datos resultantes a nube de puntos aplicando la calibración del sensor original, siendo esta nube de puntos los datos que servirán de entrada al componente *ElderCare*.

- *ElderCare*
  Este componente, con 6.677 líneas de código, incorpora la funcionalidad más importante del sistema, el algoritmo de detección y seguimiento de personas en 3D. A su vez, incorpora un módulo, ejecutable de forma independiente, que es capaz de detectar todas las situaciones de riesgo planteadas en este proyecto.

- *RemoteViewer*
  Este visualizador remoto, de 1.786 líneas de código, permite comprobar de forma remota el estado completo del sistema. Además, *RemoteViewer* ofrece un interfaz donde se presenta el estado actual del algoritmo de seguimiento, y donde también se puede ver, en tiempo real, las capturas de los diferentes sensores que se estén utilizando.

7.1.3. Aportes en este trabajo

Las novedades presentadas en este trabajo son principalmente cinco. La primera de ellas es el desarrollo de una librería de adquisición en paralelo para lidiar con los problemas de desincronización y latencia en sistemas distribuidos basados en ICE. La segunda novedad es el desarrollo de una librería (*DepthLib*), que integra funcionalidad útil para el desarrollo de aplicaciones basadas en sensores RGB-D. La dos funcionalidades principales que incluye esta librería son, por una parte, un filtrado temporal de los datos y, por otra parte, un muestreo que tiene en cuenta la distancia, con el fin de obtener datos con una densidad constante en el espacio.

El tercer aporte importante son las herramientas para la reproducción de datos enlatados, con un componente que es capaz de volcar en disco datos ofrecidos desde varios sensores, y un segundo componente que reproduce los datos guardados como si se tratase del sensor original.

El cuarto aporte es una herramienta de segmentación sobre sensores RGB-D. Este componente es capaz de segmentar los datos de primer plano de una escena captada con mapas de profundidad, teniendo en cuenta el ruido intrínseco del sensor. Adicionalmente, es capaz de aplicar la misma segmentación a cualquier otro sensor que esté captando datos sobre la misma perspectiva, como puede ser la cámara de color.

El último y principal aporte de este proyecto fin de máster es la aplicación de sensores RGB-D para implementar un sistema de teleasistencia. El sistema se basa en la detección y seguimiento de personas en 3D, implementado con un algoritmo evolutivo multimodal. Utilizando esta información, el sistema es capaz de detectar una serie de situaciones de riesgo habituales en personas con algún tipo de dependencia.

En el presente trabajo, se ha creado un sistema de teleasistencia totalmente autónomo en el que las personas monitorizadas no necesitan llevar ningún dispositivo o atuendo especial, solamente utilizando sensores RGB-D. Esto supone un gran avance con respecto a la utilización de cámaras convencionales, ya que permite disminuir el número de sensores utilizados, pudiendo incluso utilizar un solo sensor para monitorizar

una habitación de tamaño medio. La información tomada como entrada del algoritmo evolutivo es únicamente la de los datos espaciales 3D ofrecidos por el sensor. Esto permite, al no utilizar información adicional de color, que el sistema funcione incluso en total oscuridad y que sea mucho más robusto y estable.

7.1.4. Conocimientos adquiridos

Los conocimientos adquiridos durante la realización de este trabajo son muy numerosos debido a su carácter heterogéneo y las múltiples tecnologías manejadas. Desde el punto de vista la programación, la consolidación en el uso de C++ como lenguaje sobre el que asentar futuros trabajos y el uso de ICE para la comunicación de componentes distribuidos. Se han adquirido importantes conocimientos en la sincronización entre múltiples hebras y el acceso concurrente a datos.

En lo referente a los sensores RGB-D, se han adquirido conocimientos sobre las diferentes técnicas de filtrado y de muestreado aplicables a este tipo de sensores. De igual manera, se han asentado conocimientos previos sobre procesado de información 3D, tales como conversiones geométricas, proyecciones o calibración de cámaras, así como modelos de seguimiento, como el filtro de Kalman.

Se ha profundizado en los algoritmos evolutivos, así como los operadores genéticos y su aplicación en el seguimiento de personas en 3D.

7.2. Trabajos futuros

A lo largo de este trabajo fin de máster ha aumenta la funcionalidad de *JdeRobot* para realizar futuros proyectos con sensores RGB-D. Ofrece un segmentador de primer plano para cualquier tipo de aplicación que lo necesite, así como nuevas técnicas de acceso en paralelo a sensores distribuidos.

El seguimiento 3D de personas en interiores, genera numerosas líneas futuras sobre las que poder avanzar en nuevos proyectos. El primero de ellos, fundamental para la implantación del sistema desarrollado en entornos reales de manera factible, es la creación de una herramientas de calibración totalmente automática de sensores RGB-D. De esta forma, la instalación del sistema sería completamente *plug&play* sin necesidad de pasar por el proceso de calibración semi-automático usado en este proyecto.

Una limitación de este trabajo es la segmentación utilizada para detectar el primer plano de la escena. Con este procedimiento cualquier objeto móvil es tomado como primer plano y podría generar falsos positivos. Un posible trabajo futuro es diseñar un proceso de clasificación que sea capaz de discernir entre objetos y personas reales.

Un trabajo interesante, útil para el sistema desarrollado, podría consistir en desarrollar una herramienta visual o interfaz web multidispositivo., por ejemplo

mediante el uso de *html5*, que muestre en una página web todos los datos generados por el sistema. Esta puerta queda abierta por la propia arquitectura de *ElderCare* y *JdeRobot*, ya que es capaz de servir toda la información generada con interfaces ICE. Otro complemento, al hilo de este proyecto, sería desarrollar una herramienta visual con la que poder establecer y configurar las diferentes situaciones de riesgo, de forma sencilla e intuitiva, por usuario no técnicos.

Bibliografía

[Andersen *et al.*, 2012] M. Andersen, T. Jensen, P. Lisouski, A. Mortensen, M. Hansen, T. Gregersen, y P. Ahrendt. Kinect depth sensor evaluation for computer vision applications. Technical report, Technical report, Dept. of Engineering, Aarhus University, Denmark, 2012.

[Calderita *et al.*, 2012] L. V. Calderita, L. Manso, y P. Nuñez. Assessment of primesense rgb-d camera for robotic applications. In *WAF 2012, Workshop of Physical Agents 2012*, pages 147–153, 2012.

[Davison *et al.*, 2007] A.J. Davison, I.D. Reid, N.D. Molton, y O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[Dostal *et al.*, 2013] Jakub Dostal, Per Ola Kristensson, y Aaron Quigley. The potential of fusing computer vision and depth sensing for accurate distance estimation. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 1257–1262, New York, NY, USA, 2013. ACM.

[Einhorn *et al.*, 2010] Erik Einhorn, Christof Schröter, y Horst-Michael Gross. Can't take my eye off you: Attention-driven monocular obstacle detection and 3d mapping. In *IROS*, pages 816–821. IEEE, 2010.

[Fischler y Bolles, 1981] Martin A. Fischler y Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[García Martínez, 2007] Iván García Martínez. Reconstrucción 3d visual con algoritmos evolutivos. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2007.

[Gonzalez-Jorge *et al.*, 2013] H. Gonzalez-Jorge, B. Riveiro, E. Vazquez-Fernandez, J. Martínez-Sánchez, y Arias P. Metrological evaluation of microsoft kinect and asus xtion sensors. *Measurement*, 46(6):1800–1806, July 2013.

[Hartley y Zisserman, 2004] R. I. Hartley y A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Inc,] ASUSTeK Computer Inc. Asus Xtion. `http://www.microsoft.com/en-us/kinectforwindows/`. Accedido: 15-05-2013.

[Kalman *et al.*, 1960] Kalman, Rudolph, y Emil. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[Kaplan, 1991] W. Kaplan. *Advanced Calculus*. Advanced Book Program, Addison-Wesley Publishing Company, 1991.

[Khoshelham y Elberink, 2012] Kourosh Khoshelham y Er Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. In *Sensors 2012, 12, 1437?1454. 2013*, page 8238, 2012.

[Klein y Murray, 2007] G. Klein y D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234, 2007.

[Martínez de la Casa, 204 205] Marta Martínez de la Casa. Sistema de atención visual en escena. Proyecto fin de carrera, Universidad Rey Juan Carlos, 204-205.

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

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

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

[Matyunin *et al.*, 2011] Sergey Matyunin, Dmitry Vatolin, Yury Berdnikov, y Michail Smirnov. Temporal filtering for depth maps generated by kinect depth camera. In *IEEE 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, 2011.

[msk, 2013] Microsoft kinect. `http://www.xbox.com/en-us/kinect`, 2013. Accedido: 17 de octubre de 2013.

[OpenNI,] OpenNI. OpenNI. `http://www.openni.org/`. Accedido: 15-05-2013.

[Perdices García, 2010] Eduardo Perdices García. Autolocalización visual en la robocup con algoritmos basados en muestras. Proyecto fin de master, Universidad Rey Juan Carlos, 2010.

[Pineda, 2006] Antonio Pineda. Aplicación de seguridad basada en visión. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2006.

[Plaza, 2003] José María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis doctoral, Universidad Politécnica de Madrid*, 2003.

[pri, 2013] Primesense. `http://www.primesense.com`, 2013. Accedido: 17 de octubre de 2013.

[Rivas Montero, 2010] Francisco Miguel Rivas Montero. Caminata basada en ondas acopladas para el robot humanoide nao. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2010.

[Rivas Montero, 2014] Francisco Miguel Rivas Montero. Eldercare 5.0. Proyecto fin de grado, Universidad de León, 2014.

[Rosenfeld, 1969] Azriel Rosenfeld. *Picture processing by computer*. Computer science and applied mathematics. Academic press, New York, London, 1969. Réimpression : 1973.

[Shumway-Cook *et al.*, 2009] Anne Shumway-Cook, Marcia A. Ciol, Jeanne Hoffman, Brian J. Dudgeon, Kathryn Yorkston, y Leighton Chan. Falls in the Medicare population: incidence, associated factors, and impact on health care. *Physical therapy*, 89(4):324–32, April 2009.

[Stoyanov *et al.*, 2012] Todor Stoyanov, Rasoul Mojtahedzadeh, Henrik Andreasson, y Achim J. Lilienthal. Comparative evaluation of range sensor accuracy for indoor mobile robotics and automated logistics applications. *Robotics and Autonomous Systems*, 2012.

[Suzuki y Be, 1985] S. Suzuki y K. Be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, April 1985.

[Viager, 2011] Mikkel Viager. Analysis of Kinect for mobile robots, 2011.

[Zhang *et al.*, 2013] Quan Zhang, Lingmei Ren, y Weisong Shi. Honey: A multimodality fall detection and telecare system. *Telemed J E Health*, 2013.