universidad de león

Escuela de Ingenierías I. I.

Industrial, Informática y Aeroespacial

Curso académico 2013-2014

Trabajo de Fin de Grado

Seguimiento de personas en 3D basado en sensores RGB-D

Autor: Francisco Miguel Rivas Montero
Tutores: Adolfo Rodríguez de Soto
José María Cañas Plaza

El secreto de la existencia no consiste solamente en vivir, sino en saber para qué se vive

Fiódor Dostoyevski

A todo aquél que se ha contribuido de una u otra forma a enriquecer mi vida.

Agradecimientos

Quiero agradecer a todas las personas que me ha apoyado y ha confiado en mí durante todo este tiempo, gracias a ellos este trabajo ha sido un poco menos duro.

En primer lugar quiero agradecer a los tutores de este trabajo fin de grado, por una parte a Adolfo Rodríguez de Soto por su predisposición a ayudar y todas las facilidades que me ha brindado y, por otra parte, a José María Cañas Plaza ya que sin su dedicación este trabajo no habría salido adelante. Gracias por enseñarme todo lo que se, por tu empeño y buen hacer.

A todos los miembros del laboratorio de robótica de la Universidad Rey Juan Carlos, es un placer formar parte de un grupo tan brillante y poder compartir grandes momentos con vosotros.

Agradecer, como no, a todos mis compañeros del Departamento de Fisioterapia, Terapia Ocupacional, Rehabilitación y Medicina Física de la Universidad Rey Juan Carlos por toda la paciencia que habéis tenido conmigo y por haberme ayudado, sobre todo en la fase experimental del proyecto, paseando por el laboratorio siempre sin una mala cara. En especial quería destacar a Paco, Alicia, María, Esther, Lola y a mi gran amigo Javi, sois los mejores.

También quiero reconocer el apoyo de mi novia Noelia, sin ti todo esto no tendría ningún sentido. Gracias por estar siempre ahí y por ser como eres.

Y cómo no, a mi familia, en especial a mis tíos y a mis padres. Sois los verdaderos artífices de este trabajo porque sin vosotros todo esto no habría sido posible. Gracias por todo lo que hacéis y por todo lo que habéis hecho, son tantas cosas que no hay forma de recompensaros, tendréis para siempre un hueco en mi corazón.

¡Gracias a todos!

Resumen

El auge del mundo de los videojuegos en los últimos años ha hecho que se inviertan muchos recursos para el desarrollo de nuevos interfaces para mejorar la experiencia del jugador. Uno de los dispositivos más revolucionarios de los últimos años comercializado 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 o la visión artificial.

En este trabajo fin de grado se presenta un sistema de teleasistencia basado en este tipo de sensores. El sistema desarrollado consta de un algoritmo de seguimiento 3D de personas y un módulo de alarmas que es capaz de detectar una serie de situaciones de riesgo en función de las posiciones estimadas por el algoritmo. El algoritmo desarrollado utiliza únicamente la información 3D ofrecida por el sensor lo que permite que el sistema funcione de manera ininterrumpida tanto con luz como en total oscuridad.

Para el desarrollo del sistema se ha utilizado el lenguaje de programación C++ bajo la plataforma JdeRobot en Debian 7.5 de 64 bits. El uso de JdeRobot permite que los componentes desarrollados para este trabajo se puedan ejecutar de manera distribuida, facilitando el acceso remoto a los datos del entorno desde el exterior.

Índice general

1. Introducción **1**
- 1.1. Teleasistencia . . . 1
- 1.2. Visión artificial . . . 3
- 1.3. Sensores y sistemas de seguimiento 3D . . . 6
- 1.4. ElderCare . . . 11

2. Objetivos **15**
- 2.1. Descripción del problema . . . 15
- 2.2. Requisitos . . . 16
- 2.3. Metodología . . . 16
- 2.4. Planificación . . . 18
- 2.5. Esfuerzo Temporal del proyecto . . . 20

3. Plataforma de desarrollo **21**
- 3.1. JdeRobot . . . 21
- 3.2. Sensores RGB-D . . . 23
  - 3.2.1. Descripción técnica . . . 23
  - 3.2.2. Acceso software para desarrolladores . . . 29
- 3.3. PCL . . . 32
- 3.4. OpenCV . . . 33
- 3.5. Glade y librerías GTK+ . . . 34
- 3.6. OpenGL . . . 35
- 3.7. libxml++ . . . 36
- 3.8. Progeo . . . 37

4. Infraestructura desarrollada **39**
- 4.1. Componente OpenNIServer . . . 39
  - 4.1.1. Diseño software . . . 40
  - 4.1.2. Conversión a nube de puntos . . . 41
  - 4.1.3. Segmentación de personas . . . 45
  - 4.1.4. Configuración del componente . . . 46
- 4.2. Componente RgbdViewer . . . 47
- 4.3. Componente RgbdManualCalibrator . . . 51
- 4.4. Componente RemoteConfigurator . . . 53

5. Seguimiento 3D **57**
5.1. Diseño . . . 57
5.2. División de la nube de puntos en prismas . . . 58
5.3. Agrupamiento en prismas . . . 62
5.3.1. RANSAC . . . 62
5.3.2. Basado en histogramas . . . 65
5.4. Detección de observaciones no válidas . . . 68
5.5. Fusión de observaciones . . . 69
5.6. Módulo de seguimiento . . . 70
5.6.1. Por distancias . . . 70
5.6.2. Filtro de Kalman . . . 71
5.7. Módulo de alarmas y situaciones de riesgo . . . 74
5.8. Módulo gráfico . . . 75
5.9. Configuración del sistema . . . 77
5.10. Acceso remoto al sistema . . . 78

6. Experimentación y resultados **81**
6.1. Comparativa de los diferentes algoritmos de agrupamiento . . . 81
6.1.1. Comparativa de los algoritmos de seguimiento . . . 83
6.2. Validación experimental del sistema . . . 86

7. Conclusiones **91**
7.1. Conclusiones . . . 91
7.2. Trabajos futuros . . . 94

Bibliografía **97**

Índice de figuras

1.1. Sistema de teleasistencia con pulsador 1
1.2. (a) Clasificación de objetos (b) Realidad aumentada (c) Imagen de una FRMi (d) Filtrado de imágenes 5
1.3. (a) Wiimote de Nintendo (b)Eyetoy de Sony (c) Kinect de Microsoft . 5
1.4. Reconstrucción por triangulación 6
1.5. (a) Vicon para medicina (b) Vicon para animación 7
1.6. Cámara Bumblebee 7
1.7. Configuración canónica estéreo 8
1.8. (a) D-IMager de Panasonic Electric Works (b) SwissRanger de Mesa Imaging 9
1.9. Patrón de luz estructurada de Kinect 10
1.10. Versiones anteriores de ElderCare (a) Versión 1.0 (b) Versión 4.6 . . . 12
1.11. ElderCare en entornos reales 12

2.1. Modelo en espiral 17
2.2. Esfuerzo temporal del proyecto 20

3.1. Estructura de la interconexión de componentes utilizando JdeRobot. (S) Servidor (C) Cliente 22
3.2. (a) Xtion Pro Life. (b) Primesense Carmine 1.09. (c) Caprie 24
3.3. Representación de los valores del sensor de profundidad en diferentes distancias 25
3.4. Variación de las medidas de profundidad de un píxel 26
3.5. Precisión espacial. 26
3.6. Escenario de pruebas de precisión espacial. 27
3.7. Configuración del sensor 27
3.8. Tiempo de estabilización de las mediciones 28
3.9. (a) Imagen de color (b) Mapa de profundidad (c) Imagen de profundidad registrada y sobrepuesta sobre la imagen de color 30
3.10. Ejemplo de segmentación con Point Cloud Library 32
3.11. Captura del juego Rally Master Pro desarrollado con OpenGL. 35
3.12. Modelo de cámara Pinhole 38

4.1. Estructura del componente OpenNIServer 40
4.2. Cálculo del punto 3D con la información de distancia 41
4.3. Configuración del mapa de profundidad 42
4.4. Explicación de la corrección de distancia 43

4.5. (a) Reconstrucción con la distancia directa (b) Reconstrucción con la distancia corregida (c) Imagen desde la cámara de color 44
4.6. Sistema de referencia 3D solidario con la cámara 45
4.7. (a) Cámara de color. (b) Mapa de segmentación de NITE 46
4.8. (a) Imágenes (b) Nube de puntos 48
4.9. Conexión del componente RgbdViewer y estructura interna 50
4.10. Interfaz gráfico del componente rgdbViewer 50
4.11. (a) Píxeles seleccionados (b) Rectas de retroproyección en 3D 50
4.12. Interfaz gráfico del calibrator 51
4.13. Proyección del entorno en la imagen de color. 52
4.14. Nubes de puntos con distintos colores. 53
4.15. Conexión del componente RgbdManualCalibrator y estructura interna . 54
4.16. Interfaz gráfico a partir del xml de entrada 54
4.17. Esquema de la configuración remota de componentes 56

5.1. Interconexión de los componentes desarrollados en ElderCare 58
5.2. Estructura interna del componente ElderCare 59
5.3. Secuencia del algoritmo del módulo de seguimiento 3D 59
5.4. (a) Imagen cámara 1 (b) Imagen cámara 1 segmentada (c) Imagen cámara 2 (d) Imagen cámara 2 segmentada 61
5.5. Representación 3D de la segmentación de las personas en todos los sensores 62
5.6. Datos crudos englobados en prismas 63
5.7. Modelado de un prisma en 3D 64
5.8. Resultado del filtrado con RANSAC 65
5.9. Representación de los datos en histogramas 66
5.10. Histogramas acotados 67
5.11. Resultado del filtrado con histogramas 68
5.12. Falso positivo en la detección de personas de NITE 69
5.13. Resultado de combinar observaciones coincidentes. 70
5.14. Diagrama del filtrado de Kalman. 73
5.15. Interfaz gráfico de ElderCare. 76
5.16. RemoteViewer. 79

6.1. Volumen del prisma de una persona con las diferentes técnicas de agrupamiento 82
6.2. Posición 3D de una persona con las diferentes técnicas de agrupamiento 83
6.3. Comparación del motor de seguimiento por distancias con el agrupamiento de Kalman 84
6.4. Rupturas de seguimiento de personas en las diferentes configuraciones . 85
6.5. Entorno de pruebas 1. 86
6.6. (a) Situación de caída 1 (b) Situación de caída 2(c) Situación de caída 3 (d) Situación de salida 87
6.7. Entorno de pruebas 2. 88
6.8. (a) Situación de salida 1 (b) Situación de salida 2 (c) Situación de caída 1 89

Índice de tablas

3.1. Especificaciones de Kinect 23

6.1. Rendimiento de los agrupamientos 81
6.2. Rendimiento de las diferentes configuraciones del motor de seguimiento 85
6.3. Resultados de la detección en las diferentes situaciones de riego 88
6.4. Resultados de la detección en las diferentes situaciones de riesgo 89

Capítulo 1
Introducción

En este primer capítulo vamos a explicar, de forma general, el contexto en el que se encuadra este proyecto. Desarrollaremos los conceptos básicos de teleasistencia y visión artificial, presentaremos las dificultades a la hora de desarrollar un sistema de seguimiento tridimensional de personas y haremos una descripción de 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 caída

Figura 1.1: Sistema de teleasistencia con pulsador

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 es 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, evaluando si ha habido algún cambio brusco en los datos del sensor. Con esta información se puede determinar si se ha producido una caída.

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 personas con algún tipo de afectación cognitiva como pueden ser los enfermos 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. 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 donde una persona se encargará de

[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

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.

Estos sistemas, a pesar de mejorar 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 con poca luz 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, 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:

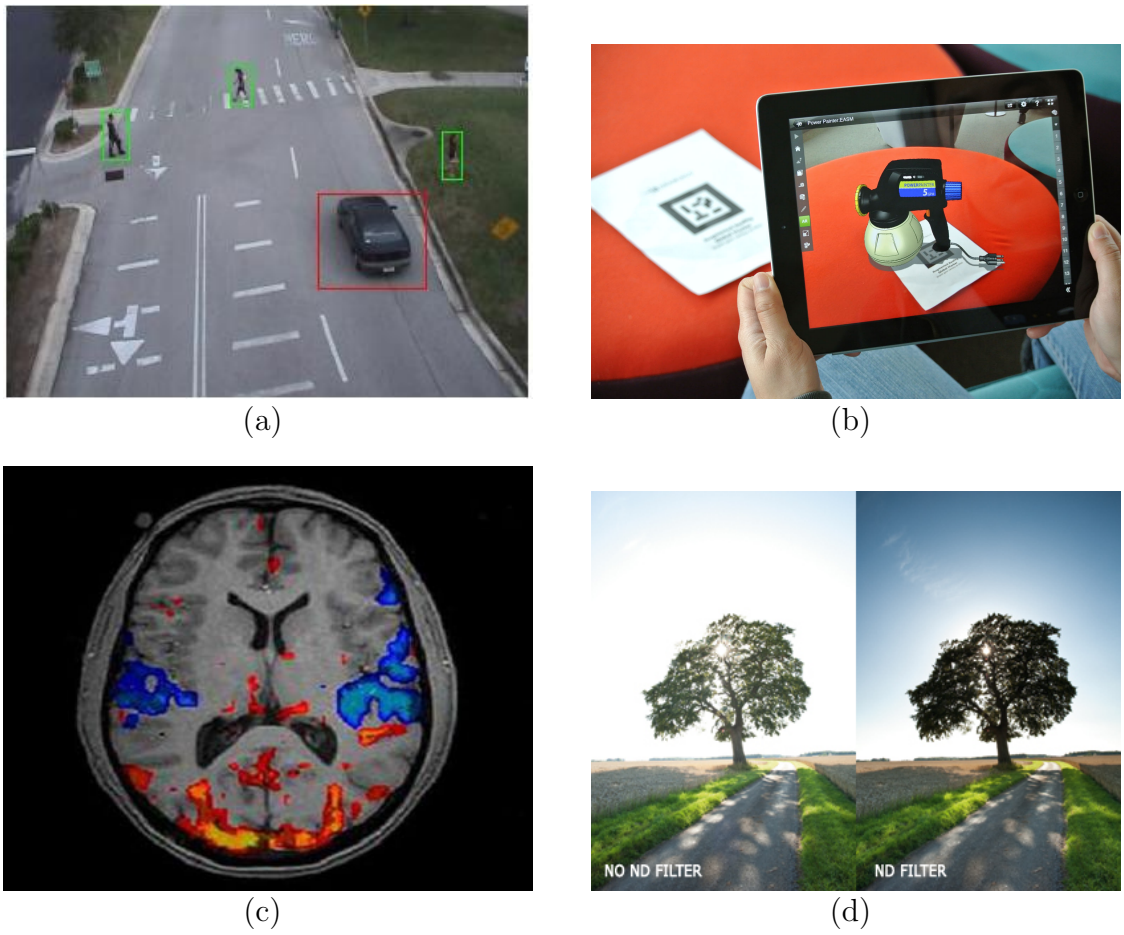
- Captura de imágenes en diferentes rangos del espectro electromagnético (imágenes infrarrojas, térmicas...).
- Procesado de imágenes con el fin de conseguir reducción de ruido, mejora de contraste, realzado...
- Detección y clasificación de objetos.
- Analizar o seguir objetos en movimiento.
- Registro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

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 dispositivo Eyetoy (figura 1.3(b)), que básicamente incorpora 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 se disparen 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.



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

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

(a) (b) (c)

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

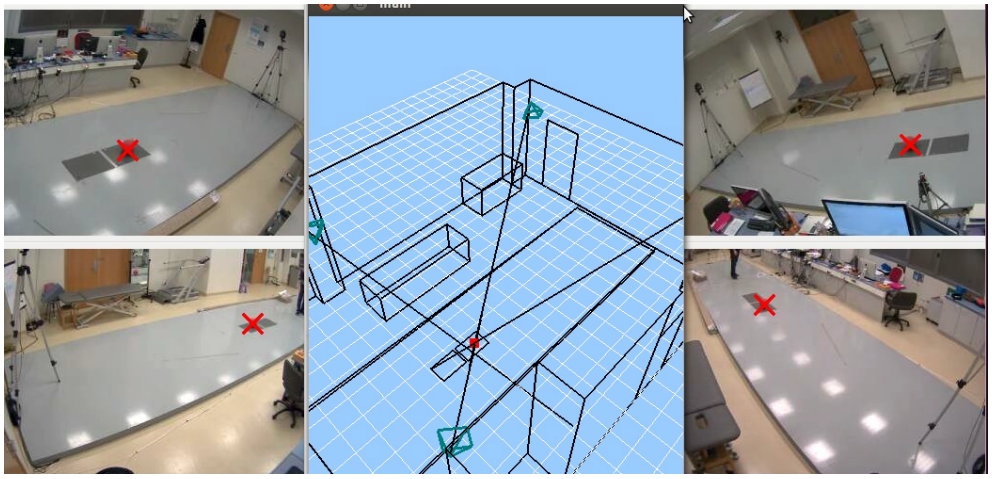


Figura 1.4: Reconstrucción por triangulación

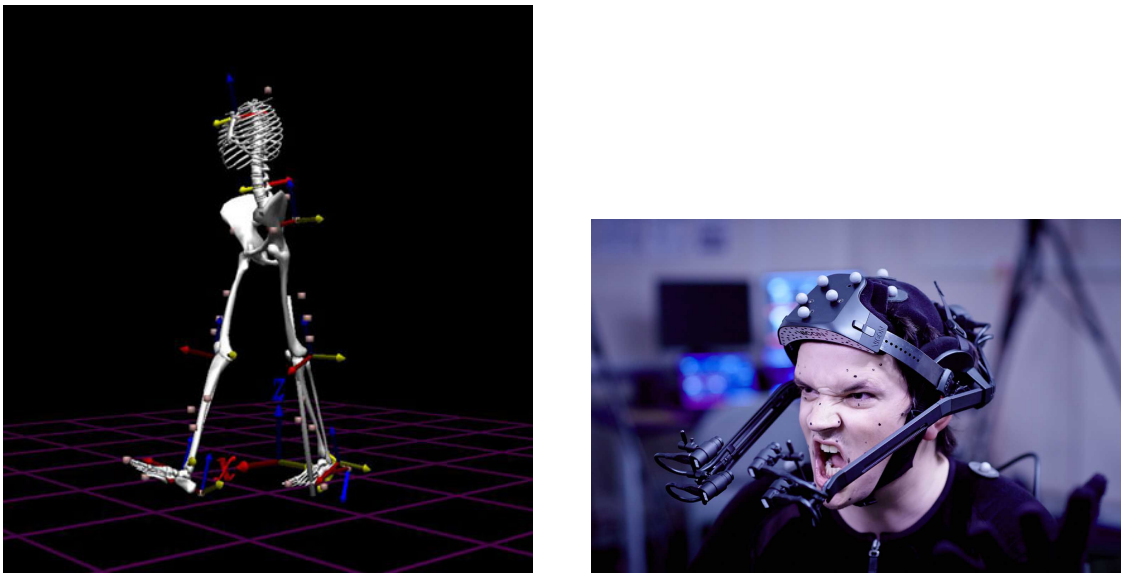
1.3. Sensores y sistemas de seguimiento 3D

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 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 médicos (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



(a) (b)

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

Figura 1.6: Cámara Bumblebee

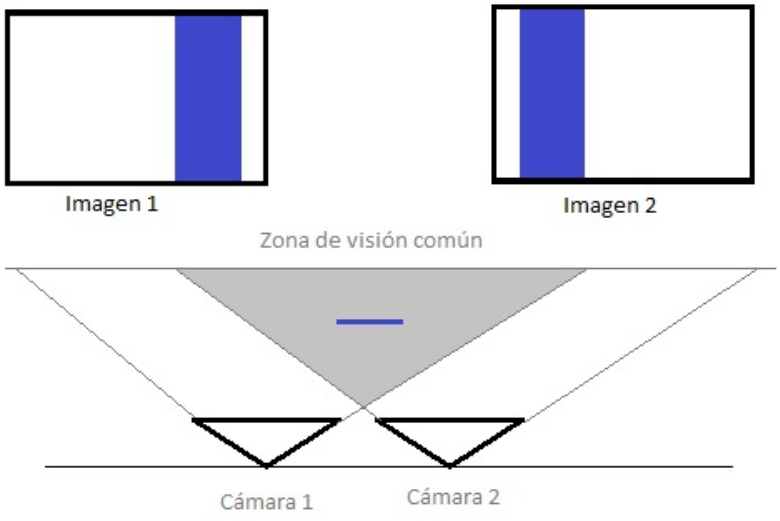


Figura 1.7: Configuración canónica estéreo

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 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 se basa en estudiar las deformidades de las sombras que inciden en la escena. Se calcula el campo de

(a) (b)

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

normales asociada a esa escena y a partir de ella se calculan las distancias. Uno de los principales problemas de esta técnica es la dificultad que tiene 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 y consiguiendo muy buenas aproximaciones suponiendo que se están capturando superficies Lambertianas (tienen el mismo brillo en todas las direcciones), que la fuente de luz se encuentra en el infinito y que no existe una distorsión fotométrica.

Los sistemas basados en una arquitectura con una cámara suelen estar más orientados a aplicaciones de autolocalización 3D, como en *Monocular Simultaneous Localization And Mapping* (Mono SLAM) [Davison *et al.*, 2007], asentado sobre filtros extendidos de *kalman*, aunque también existen trabajos sobre mapeado 3D basado en esta técnica [Einhorn *et al.*, 2010]. Una evolución de este sistema monocular de localización es el Parallel Tracking and Mapping (PTAM) [Klein y Murray, 2007], que separa el seguimiento del mapeo para permitir 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.

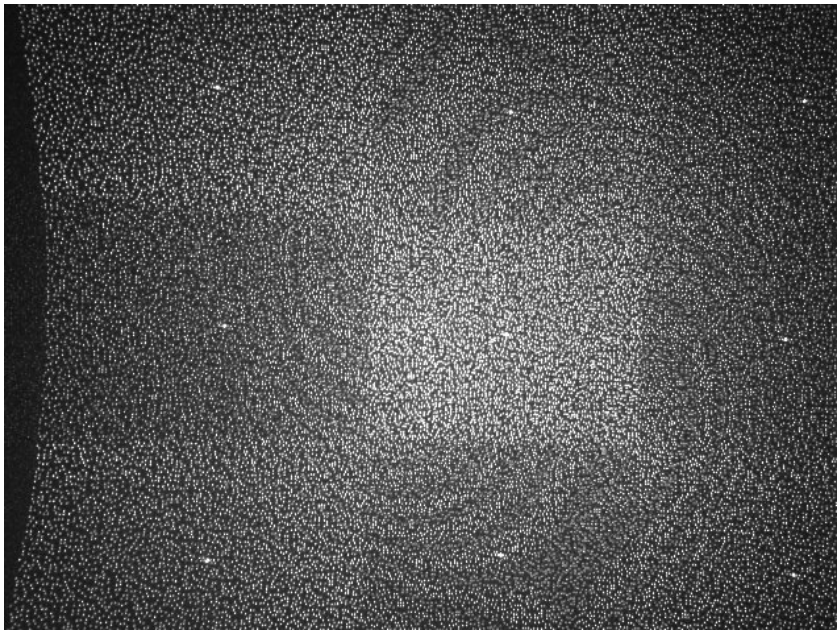


Figura 1.9: Patrón de luz estructurada de Kinect

- 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 de 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 caídas.

El primer antecedente de las técnicas de localización y seguimiento aplicadas en *ElderCare* es el proyecto fin de carrera con el título *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.

Esta tecnología se empieza a orientar hacia el campo de la teleasistencia con el trabajo fin de carrera *Seguimiento 3D visual de múltiples personas utilizando un algoritmo evolutivo multimodal* [Marugán Alonso, 2007]. En este trabajo se presentan mejoras en las técnicas de seguimiento con el fin de localizar a múltiples sujetos y donde el propio sistema es capaz de aprender automáticamente el color de cada uno de ellos. La misma autora realizó dos evoluciones más de *ElderCare* [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 del soporte en Android para visualizar el entorno monitorizado desde el exterior utilizando dispositivos móviles.
- Pruebas piloto en entornos reales (figura 1.11).

El presente trabajo supone la quinta evolución de *ElderCare*, el cual conlleva un cambio completo del sistema, teniendo en cuenta que el sensor base, que hasta la fecha habían sido cámaras de color convencionales, será reemplazado por sensores de profundidad tipo Kinect. 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 las versiones anteriores, beneficiándose del uso de estos novedosos sistemas de captación tridimensionales. 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 base de desarrollo. A

[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

continuación, en el capítulo 5 se define la solución planteada 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, así como los requisitos que debe cumplir nuestra solución.

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 éste el elegido, debido a que da 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 lo hemos articulado en varios subobjetivos:

1. Crear un *servidor software* que se conecte a los dispositivos RGB-D y que ofrezca todos sus datos, tanto la cámara de color, como del mapa de profundidad y de la nube de puntos en referencia a posición del sensor como al origen de coordenadas de nuestro entorno.
2. Diseñar y desarrollar un *algoritmo de estimación 3D* que reciba una nube de puntos desde diferentes sensores, analice toda la información y que sea capaz de detectar la posición tridimensional de *todas las personas* presentes en la escena.

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 la *configuración* remota del sistema,

5. Llevar a cabo un proceso de validación experimental del sistema desarrollado para estudiar tanto la fiabilidad como la viabilidad de su uso en entornos reales.

2.2. Requisitos

Para cumplir los objetivos presentados en el punto anterior, nuestro proyecto deberá, además, 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, y detectar diversas situaciones de riesgo planteadas en nuestro proyecto.

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

2.3. Metodología

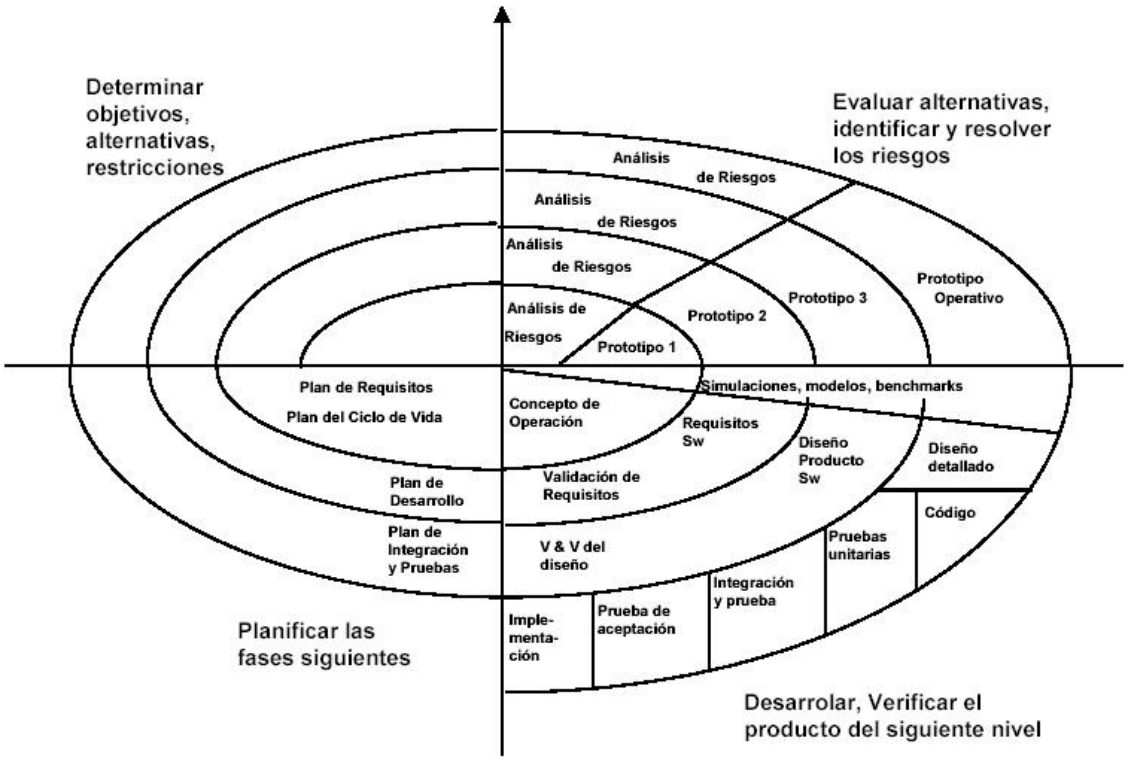
En el desarrollo de los componentes software de nuestro trabajo, el modelo de ciclo de vida utilizado ha sido el modelo en espiral basado en prototipos, ya que permite desarrollar el proyecto de forma incremental, aumentando la complejidad progresivamente y haciendo posible la generación de prototipos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

nos 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 2011 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 ido proponiendo 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:

- **Familiarización con el entorno software**
  Esta primera fase, nos centraremos en la familiarización con el entorno software *JdeRobot* (versión 5.1) utilizado para la realización del proyecto. Esta familiarización no ha sido del todo compleja, puesto que he podido aprovechar mis conocimientos previos adquiridos en este entorno de desarrollo al haber realizado mi proyecto fin de carrera [Rivas Montero, 2010] bajo este mismo entorno aunque en una versión anterior (4.2.1). La principal tarea en esta fase ha sido aprender el lenguaje *slice*, utilizado para la definición de los interfaces que utiliza *JdeRobot* para la comunicación de sus componentes así como el tipo de conexiones soportadas entre componentes y la manera de establecer las mismas.

- **Análisis de los dispositivos RGB-D y sus entornos de programación**
  En esta segunda fase, se hizo un estudio sobre los diferentes dispositivos RGB-D de consumo, tipo Kinect, disponibles en el mercado así como los diferentes SDKs disponibles para el acceso a dichos dispositivos. Finalmente, nos decantamos por el sensor Xtion de Asus debido a su precio similar al de Kinect pero con una serie de ventajas.

- **Creación de un servidor RGB-D**
  Tras haber definido tanto el sensor a utilizar como el SDK con el que accederemos a él, se desarrolló un componente bajo *JdeRobot* capaz de servir la imagen de color, el mapa de profundidad y la nube de puntos 3D con su color correspondiente. Este componente recibe el nombre de *OpenNIServer* y es capaz de ofrecer la información tridimensional anclada al propio sensor o bien a un origen de coordenadas externo a través de un fichero de calibración. Para ello utiliza una matriz K para los parámetros extrínsecos y una matriz RT para la rotación y traslación.

  Para poder comprobar el correcto funcionamiento del servidor, se desarrolló un componente llamado *RgbdViewer* capaz de presentar gráficamente la información de los datos ofrecidos por el servidor. Este componente muestra en un interfaz

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

desarrollado con GTK la imagen de color y el mapa de profundidad y se apoya en un visor de OpenGL para mostrar la información 3D de la nube de puntos.

- **Creación de una herramienta de calibración de sensores RGB-D**
  Una vez conseguido el funcionamiento básico tanto del servidor que se conecta al dispositivo y el cliente es que capaz de representar la información, surgió la necesidad de crear una herramienta de calibración para anclar la referencia de la información tridimensional a la de nuestro entorno.

- **Estudio de la segmentación de personas**
  Con todo el entorno preparado, empezamos a comprobar el funcionamiento de la detección de personas incluido en la librería de NITE. Finalmente tras estudiar varias opciones, se incorporó la opción de la segmentación de personas en el propio *OpenNIServer* para que fuese capaz de servir los datos ya segmentados.

- **Implementación de un seguidor de personas**
  Ya con la segmentación de personas implementada, se empezó a desarrollar un algoritmo que filtrase los datos 3D obtenidos y fuese capaz de explicar cada una de las personas utilizamos primitivas volumétricas, en concreto utilizando prismas.

- **Desarrollo de un módulo de alarmas**
  Llegados a este punto, se fijaron cuáles serían las situaciones de riesgo que se querían llegar a detectar con nuestro sistema. Paralelamente, se comenzó a desarrollar un módulo que fuese capaz de detectar estas situaciones basándose en la posición 3D de cada una de las persona detectadas en el entorno.

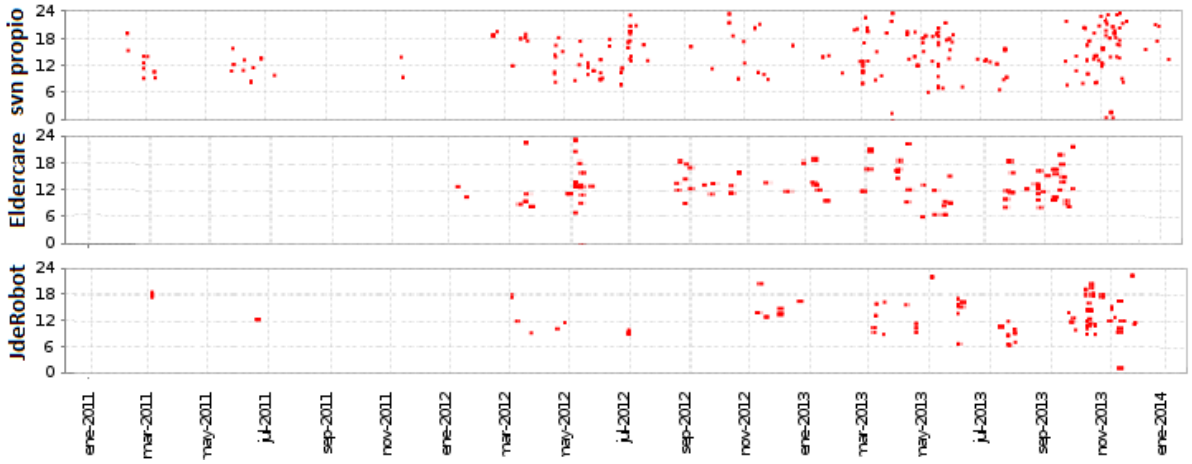
- **Desarrollo de las herramientas remotas**
  Con el sistema ya funcionando, en las últimas etapas surgió la necesidad de integrar una serie de herramientas remotas. La primera herramienta remota desarrollada ofrece la posibilidad de saber qué está pasando en el lugar monitorizado pudiendo ver la posición de las personas detectadas, así como las imágenes de color de cada uno de los dispositivos instalados. Este componente se puede ejecutar remotamente en otra ubicación distinta a la zona monitorizada.

  Otra herramienta adicional es un configurador remoto que permite cambiar a distancia y en tiempo de ejecución los parámetros del sistema sin necesidad de desplazarnos a lugar donde esté instalado el equipo principal.

- **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 tanto en el laboratorio como en la oficina de una empresa colaboradora.

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 grado, 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 vamos a describir 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 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 [Cañas 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 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 y se interconectan entre sí utilizando la abstracción de *proxies*. Esto permite que cada componente pueda ser ejecutado en un computador distinto, con una arquitectura

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

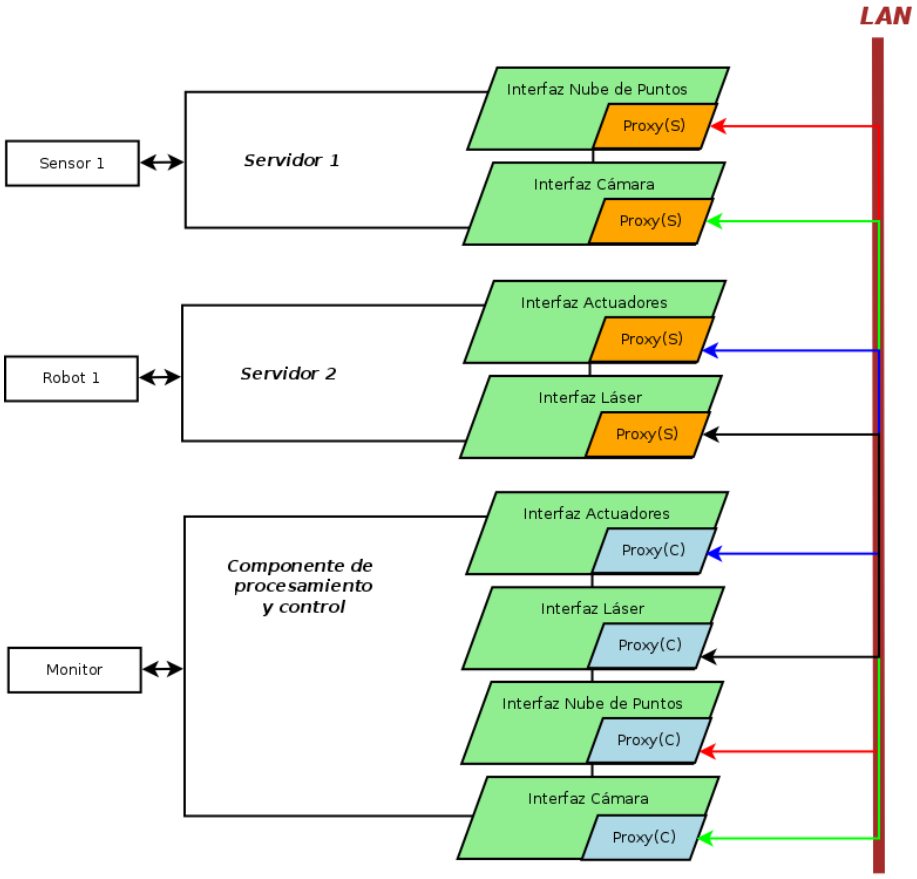


Figura 3.1: Estructura de la interconexión de componentes utilizando JdeRobot. (S) Servidor (C) Cliente

distinta e incluso con sistemas operativos distintos. Si dos componentes se ejecutan 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 este conectado.

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 los datos para procesarlos y trabajar con ellos.

La versión de *JdeRobot* utilizada para realizar este proyecto es la última versión estable, la 5.2.1.

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.

3.2.1. Descripción técnica

El fabricante que está detrás de la tecnología sobre la que se apoya Kinect es PrimeSense[3]. 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 4.7(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

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

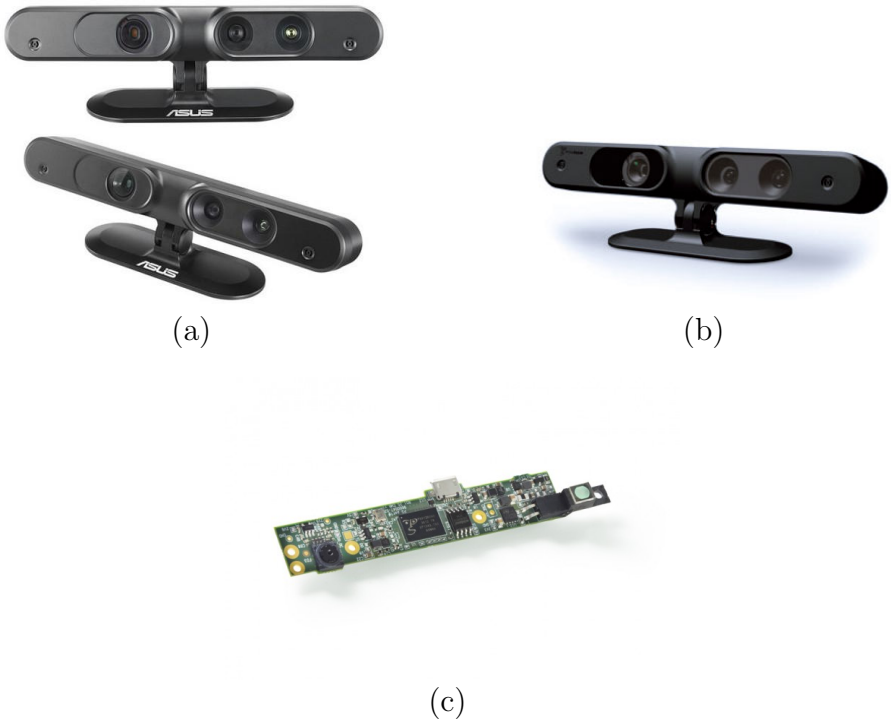


Figura 3.2: (a) Xtion Pro Life. (b) Primesense Carmine 1.09. (c) Caprie

2.0 alimentado e incorporando todas las herramientas necesarias para acceder a él.

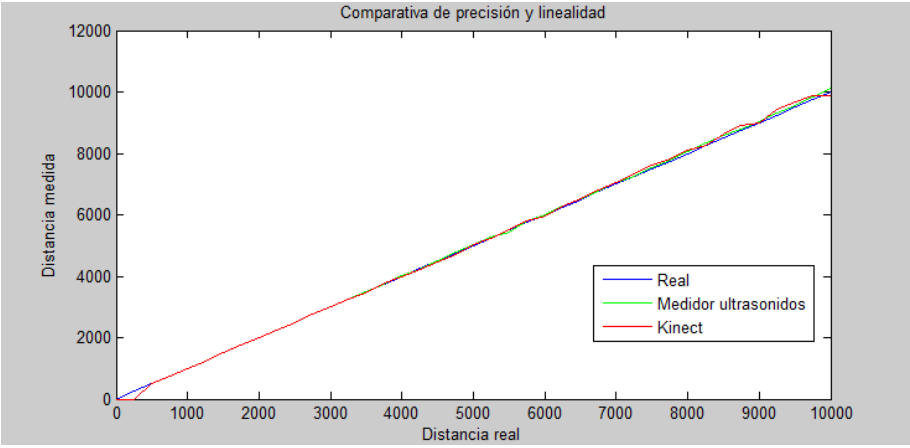


Figura 3.3: Representación de los valores del sensor de profundidad en diferentes distancias

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 3.3, 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 en sensor a diferentes distancias de la superficie plana dentro de su propio 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 un punto en concreto apuntando a un objeto a una distancia fija este valor oscila bastante alrededor de la distancia real (figura 3.4).

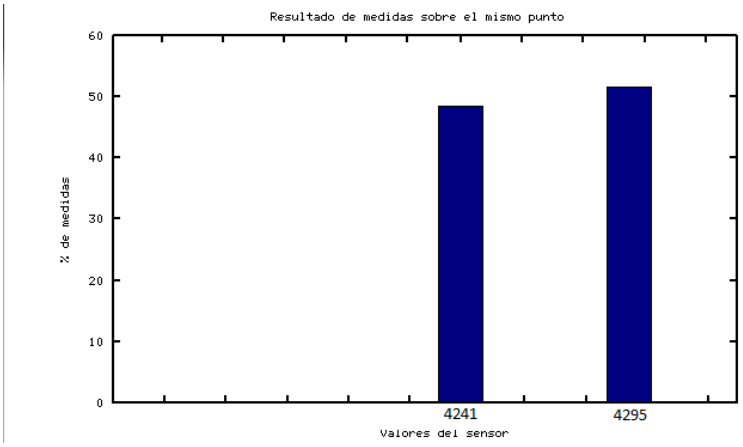


Figura 3.4: Variación de las medidas de profundidad de un píxel

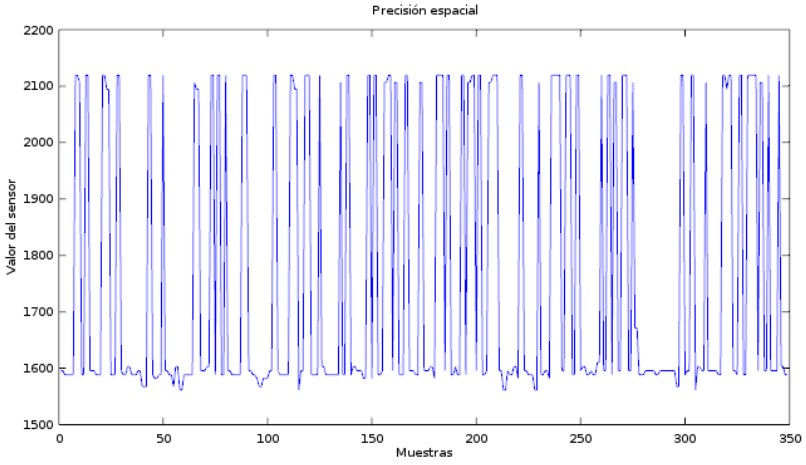


Figura 3.5: Precisión espacial.

- **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 3.5 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 3.6). El píxel sobre el que se obtiene el valor de distancia es uno de los píxeles en el borde superior de la silla.

- **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 3.7.

- **Tiempo de calentamiento**
  Un factor a tener el cuenta a la hora de utilizar estos sensores es que tardan cierto

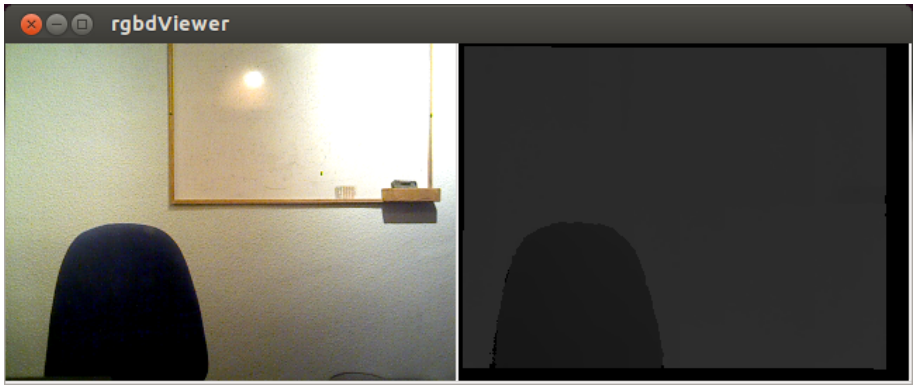


Figura 3.6: Escenario de pruebas de precisión espacial.

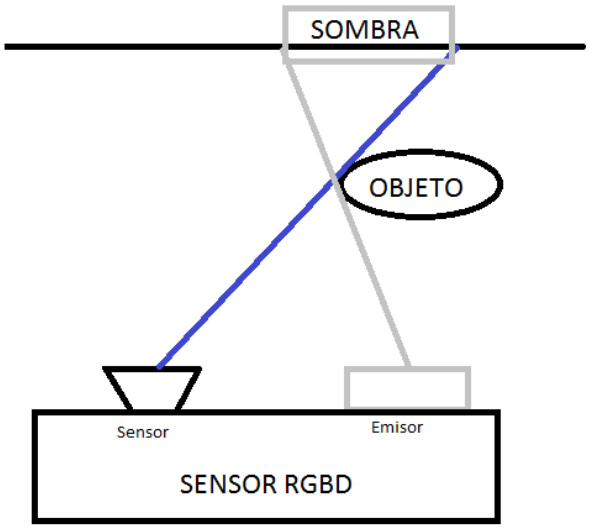


Figura 3.7: Configuración del sensor

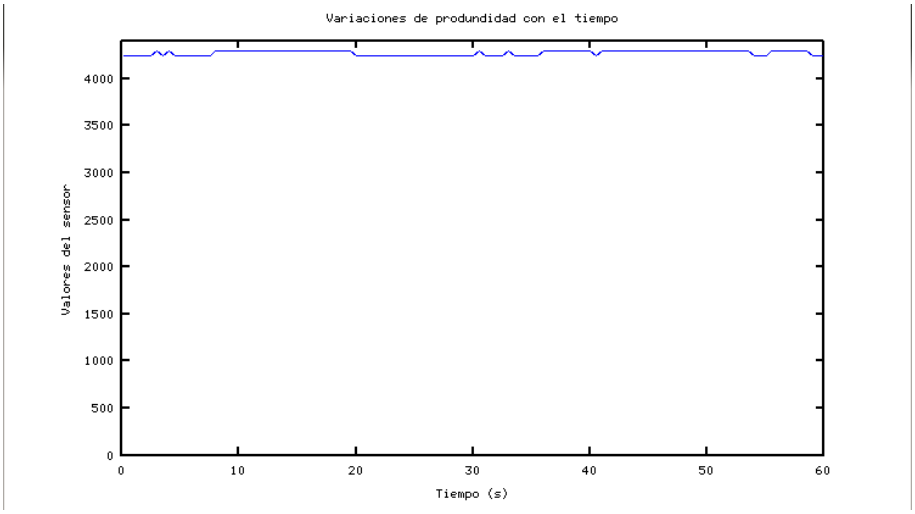


Figura 3.8: Tiempo de estabilización de las mediciones

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 3.8).

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*[4].

Freenect y OpenKinect

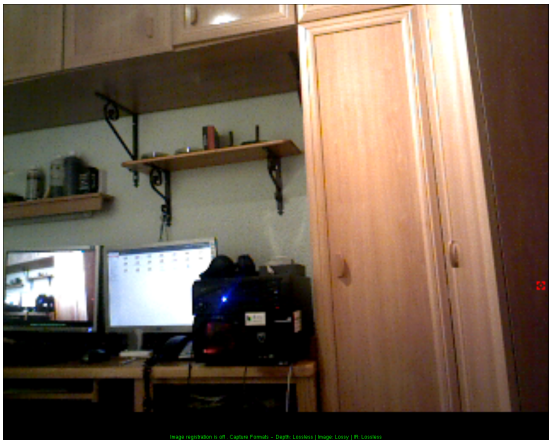
Como primer API de software libre para los sensores Kinect, *libfreenect* tuvo mucho éxito, ya que ofrecía las funciones básicas para poder acceder tanto la imagen de color de su cámara RGB como a los valores del sensor de profundidad. Los principales problemas de esta librería eran su alto consumo de 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

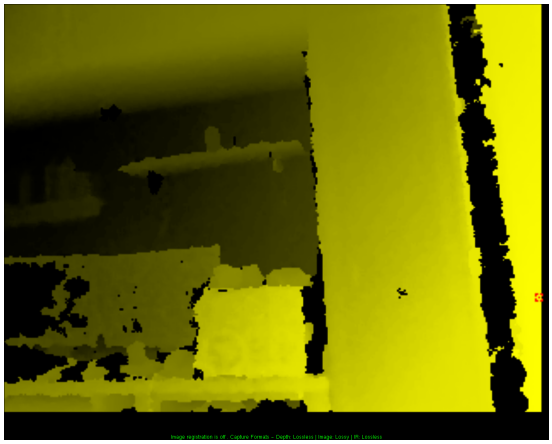
Un mes después de la aparición de *libfreenect*, PrimeSense publicó su propia librería para acceder a los sensores basados en el chip que ellos mismos fabricaban. Esta librería se liberó con licencia Apache 2.0, bajo el nombre de OpenNI[5]. Esta librería, no sólo ofrece acceso a los datos de los diferentes 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 3.9).
- **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.

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



(a)



(b)

(c)

Figura 3.9: (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, que es 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 la combinación de OpenNI y NITE, en su versión más actual (2.2). Las razones por la que se han elegido estas librerías son su mantenimiento directo por los propios fabricantes del sensor, su difusión en una amplia comunidad de desarrolladores[6][7], tener soporte para linux (x86 y x64) y no consumir una gran cantidad de recursos en su ejecución.

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

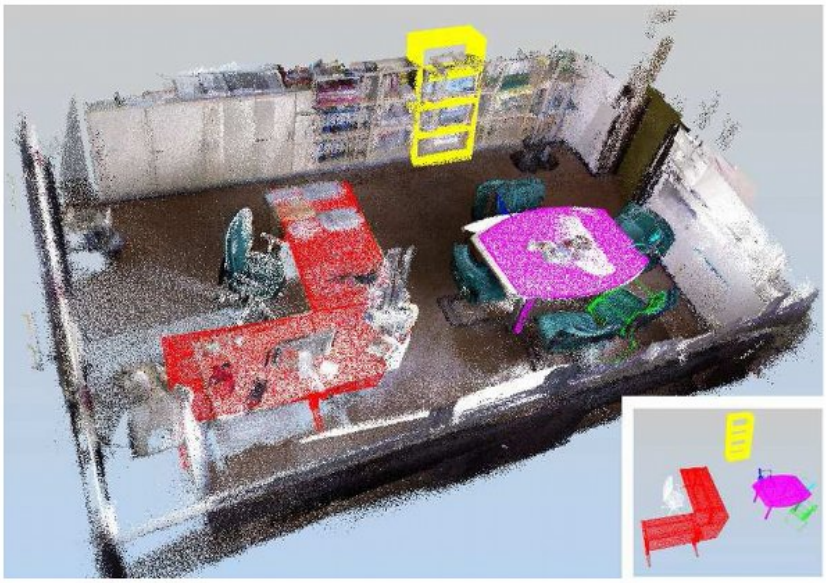


Figura 3.10: Ejemplo de segmentación con Point Cloud Library

3.3. PCL

Point Cloud Library (PCL[8]) 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[9]) 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 algoritmos de filtrado, extracción de características, reconstrucción de superficies, registro, ajuste de modelos y segmentación (figura 3.10).

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

En este proyecto se ha utilizado PCL para hacer segmentación de planos y descartar observaciones no válidas que puedan producir un mal comportamiento del sistema.

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

3.4. OpenCV

OpenCV[10] 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 más actual es la 2.4.6, 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 [11] 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 y en el tamaño de las imágenes.

[10] http://opencv.org/
[11] 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 aunque 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, 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 al utilizar

Figura 3.11: Captura del juego Rally Master Pro desarrollado con OpenGL.

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 [12] 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 3.11). 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.

[12] 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 este proyecto, se ha 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 3.12. 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*).

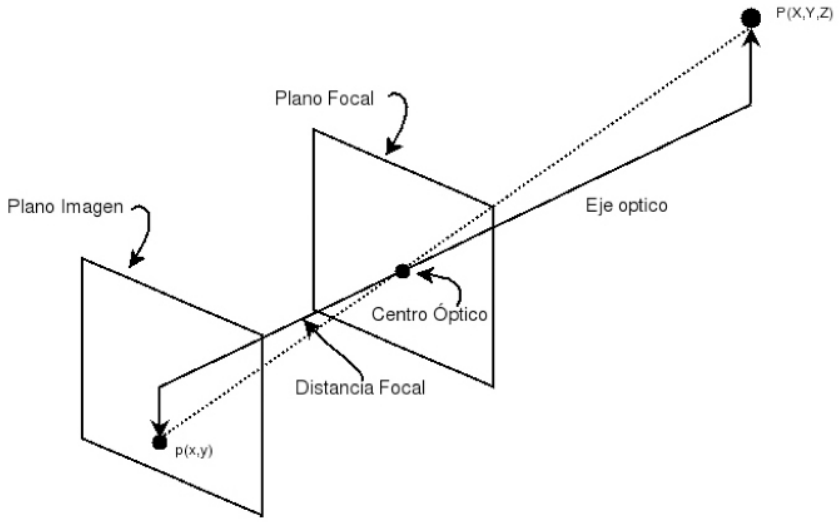


Figura 3.12: Modelo de cámara Pinhole

Capítulo 4
Infraestructura desarrollada para sensores RGB-D y configuraciones remotas

Una vez explicado tanto el contexto del proyecto como los objetivos que se afrontan en él, en este capítulo se detallarán los nuevos componentes y funcionalidades que se han añadido a *JdeRobot* como infraestructura útil. En primer lugar, los componentes relacionados directamente con el soporte para poder trabajar con dispositivos RGB-D y en segundo lugar, aquellos componentes genéricos que permiten acceso remoto para visualización de resultados y configuración de parámetros.

4.1. Componente OpenNIServer

OpenNIServer es el nombre recibe el servidor que se ha desarrollado bajo el entorno de OpenNI para dar soporte a los sensores RGB-D dentro de la plataforma de *JdeRobot*. Este servidor ofrece la imagen de color y el mapa de profundidad como imágenes con el interfaz genérico de cámaras de *JdeRobot* y la nube de puntos sobre un interfaz específico, desarrollado para trabajar con este tipo de datos:

Los parámetros *(x,y,z)* corresponden con la posición 3D del punto y *(r,g,b)* a su color correspondiente tomado de la imagen de color. Este interfaz tiene un parámetro extra, el *id*. Este valor se utiliza cuando la segmentación de personas de NITE está activada dentro de *OpenNIServer*. Los valores que toma esta variable serán explicados en profundidad en el apartado 4.1.3.

A continuación, explicaremos detalladamente las funcionalidades más destacadas de este componente, así como los diferentes modos que se pueden fijar variando su configuración.

```
module jderobot{

    struct RGBPoint{
      float x;
      float y;
      float z;
      float r;
      float g;
      float b;
      int id;
    };
    sequence<RGBPoint> RGBPointsPCL;

    class pointCloudData{
        RGBPointsPCL p;
    };
    interface pointCloud{
        idempotent pointCloudData getCloudData();
    };
};
```

Cuadro 4.1: Interfaz pointCloud

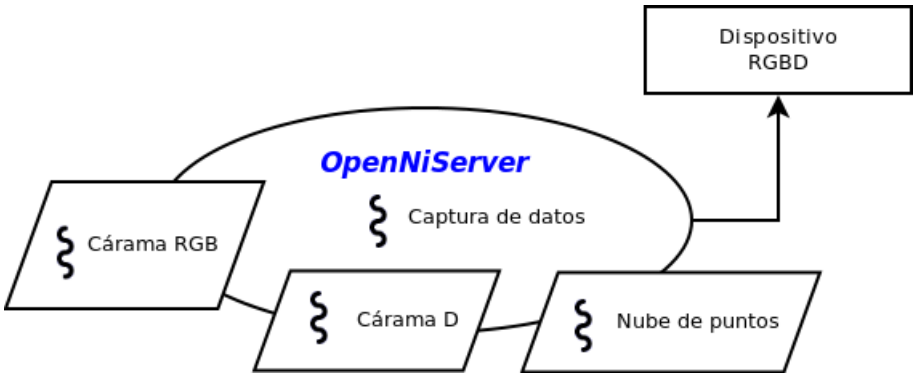


Figura 4.1: Estructura del componente OpenNIServer

4.1.1. Diseño software

El diseño que sigue el componente *OpenNIServer* es multihebra, donde cada ejecución realiza una tarea concreta. La primera se encarga de pedir datos al sensor de forma periódica y alojar esta información en memoria compartida, protegida detrás de un *mutex*. Para exportar estos datos recogidos por el hilo principal se han implementado otras tres hebras: una para la cámara de color, una segunda que toma los datos del mapa de profundidad y los codifica como una imagen de profundidad, y una tercera que ofrece estos mismos datos pero como una nube de puntos directamente en 3D. Esta estructura de diseño se puede apreciar en la figura 4.1.

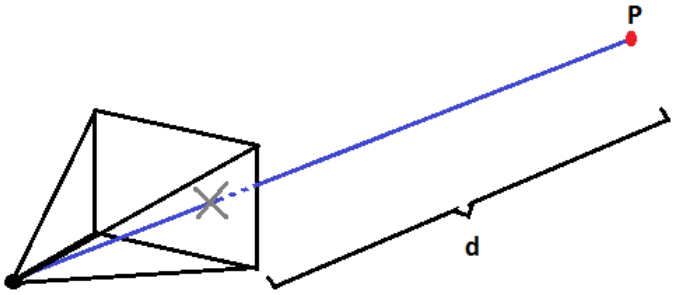


Figura 4.2: Cálculo del punto 3D con la información de distancia

4.1.2. 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 píxeles de la imagen de color (imagen D).

Para representar esta información conjunta en 3D, se utilizó *Progeo* y su modelo pinHole 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 4.2).

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 un sitio diferente por lo que no ven exactamente lo mismo. Para que esta asociación sea directa lo más sencillo 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. De la misma forma, 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 4.2.

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

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

Cuadro 4.2: Funciones ofrecidas por OpenNI para el registro y sincronización de imágenes

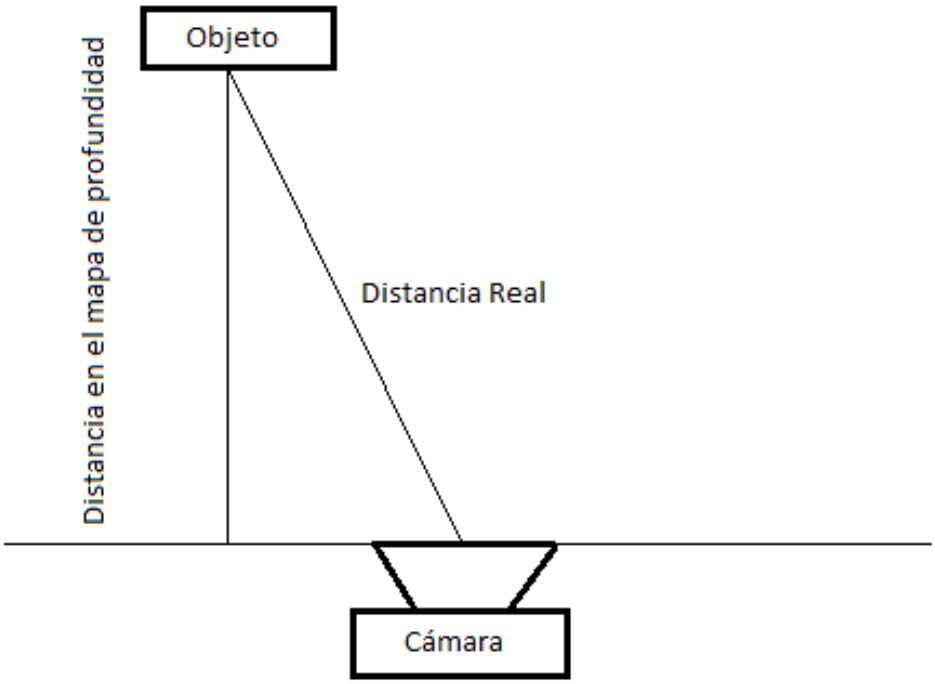


Figura 4.3: Configuración del mapa de profundidad

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 4.4. 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 4.3). 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 4.4, 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 vector $\overrightarrow{k}$, 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**:

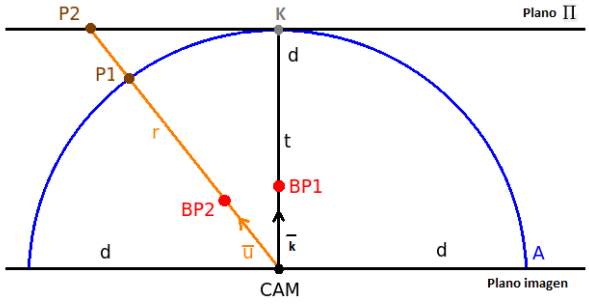


Figura 4.4: Explicación de la corrección de distancia

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

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, y pudiendo calcular el vector unitario como:

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

Por lo tanto:

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

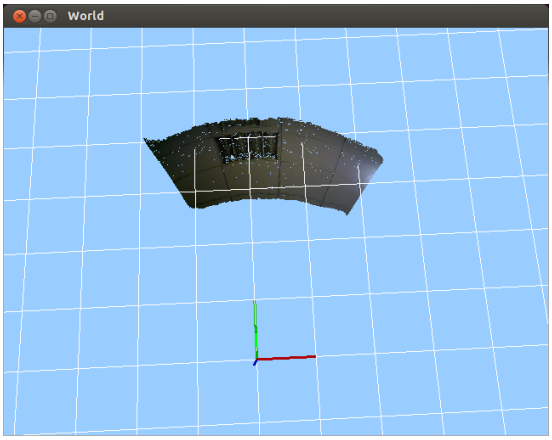
Como sabemos que el punto **P2** se encuentra en el plano **Π** podemos aplicar la ecuación del plano: $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$. Utilizando como puntos $P(x_0, y_0, z_0) = K$ y **P'**(x,y,z)=**P2** y el vector normal al plano $\overrightarrow{N}$(A,B,C)=$\overrightarrow{k}$:

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

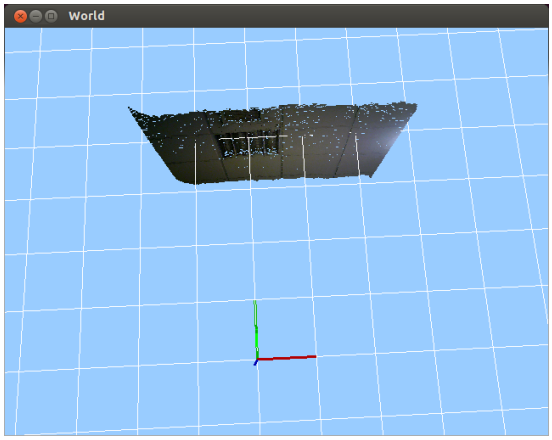
De esta fórmula podemos despejar la **t**:

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

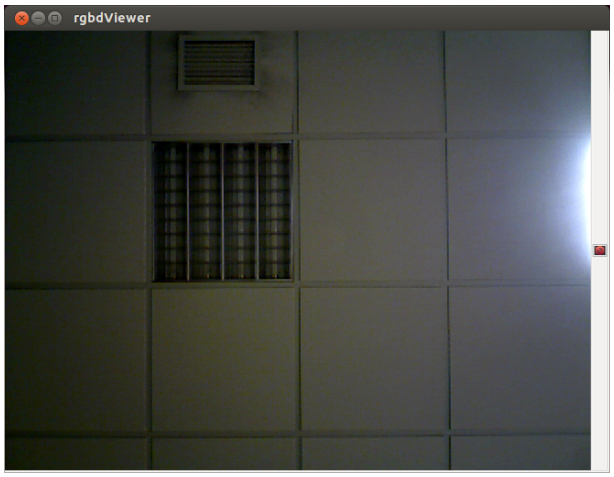
Aplicando el resultado de **t** sobre la función 4.1 obtenemos el punto real buscado. En la figura 4.5, podemos ver en un ejemplo real la diferencia entre la reconstrucción con la distancia que nos da directamente el sensor y la reconstrucción con la distancia corregida. La cámara enfoca directamente a un plano, el techo del laboratorio (figura 4.5(c)), y como podemos ver en la figura 4.5(a), la reconstrucción del plano es esférica alrededor de la cámara, mientras que una vez corregida la distancia (figura 4.5(b)), la reconstrucción 3D representa fielmente la realidad.



(a)



(b)



(c)

Figura 4.5: (a) Reconstrucción con la distancia directa (b) Reconstrucción con la distancia corregida (c) Imagen desde la cámara de color

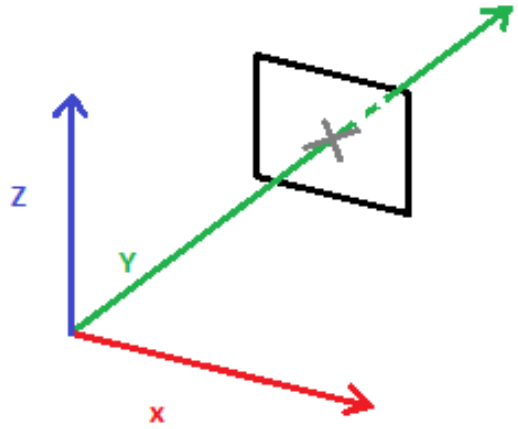


Figura 4.6: Sistema de referencia 3D solidario con la cámara

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 esta calibración es válida para cualquiera de las dos.

A su vez, este componente *OpenNIServer* es capaz de trabajar sin parámetros de calibración introducidos externamente. En este caso, *OpenNIServer* fijará los parámetros intrínsecos de la cámara como si fuese una cámara ideal y los parámetros extrínsecos los fijará con los valores por defecto siguientes (figura 4.9):

- **Posición:** es fijada en el punto (0,0,0).
- **Foco de atención:** es fijado al punto (0,1,0).
- **Roll:** fijado a 0.

Con esta configuración, conseguimos anclar como referencia de los puntos en 3D la propia posición de la cámara, fijando el plano imagen en un plano paralelo al plano XZ del mapa.

4.1.3. Segmentación de personas

Como se detalló en el apartado 3.2.2, NITE incluye una funcionalidad de detección de personas que es muy útil para el sistema que se pretende abordar en este proyecto. Por ello, se han incluido funcionalidades de NITE también en este componente, siendo esta integración bastante sencilla, ya que es directamente compatible con OpenNI. NITE ofrece la segmentación a través de un nuevo mapa, directamente correspondiente con la imagen de profundidad, como se puede apreciar en la figura 4.7.

(a) (b)

Figura 4.7: (a) Cámara de color. (b) Mapa de segmentación de NITE

La codificación de este mapa es muy sencilla, NITE asigna a cada píxel un identificador diferente para cada una de las personas detectadas, y 0 si ese píxel ha sido detectado como fondo del entorno. Es justo el valor de este identificador el que se le asigna al parámetro *id* del interfaz *pointCloud*. De esta forma, nos permite saber a qué pertenece cada uno de los puntos de una nube.

4.1.4. Configuración del componente

Como todos los componentes de *JdeRobot*, este componente permite ser configurado utilizando un fichero de configuración que es analizado por el propio componente. *OpenNIServer* permite la siguiente configuración:

- **PlayerDetection:** Activando esta opción, se activará la parte del código del componente que pide a NITE el mapa de identificaciones para la detección de las personas.
- **Mode:** Con este parámetro podemos fijar el modo de vídeo del dispositivo. Los modos disponibles para *OpenNIServer* son:
  - 0: 320x240@30
  - 2: 320x240@60
  - 4: 640x480@30
  - 6: 320x240@25
  - 8: 640x480@25
- **ImageRegistration:** con este parámetro se puede activar la opción del registro entre la imagen de color y el mapa de profundidad, con lo que simplificamos al máximo la correspondencia entre color y profundidad.
- **deviceId:** este es el identificador del dispositivo al que se va a conectar el componente. *OpenNIServer* permite ser ejecutado en un mismo ordenador varias veces y que cada uno se conecte a un dispositivo diferente.

- **calibration:** si utilizamos un fichero de calibración externo obtenido con *progeo* desde otro componente, se le puede pasar la ruta del fichero en este parámetro.
- **CameraRGB.Mirror:** permite espejar la imagen obtenida con el sensor, muy útil para algunas aplicaciones.
- **CameraRGB.PlayerDetection:** activando este parámetro, *OpenNIServer* ofrecerá las imágenes de color segmentadas, dejando su color original a las zonas donde NITE ha detectado que hay personas y dejando el resto de la imagen negra.
- **CameraDEPTH.Mirror:** este parámetro ofrece la misma funcionalidad que CameraRGB.Mirror, pero con la imagen de profundidad.
- **CameraDEPTH.PlayerDetection:** este parámetro ofrece la misma funcionalidad que CameraRGB.PlayerDetection, pero con la imagen de profundidad.

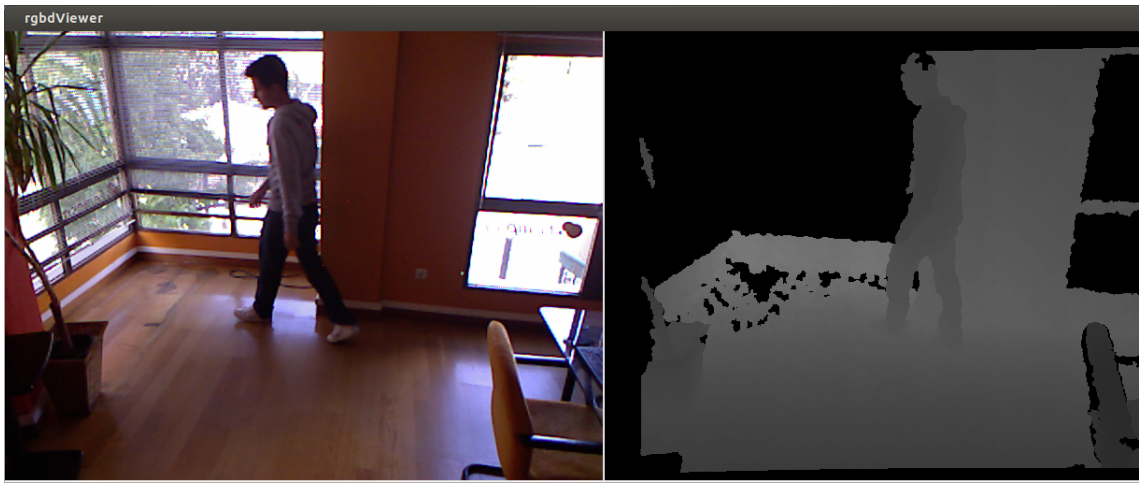
4.2. Componente RgbdViewer

Este componente se comporta fundamentalmente como un visor capaz de representar la información ofrecida por *OpenNIServer*. Muestra la imagen de color y el mapa de profundidad en un visor GTK(figura 4.8(a)) y la nube de puntos sobre un visor OpenGL(figura 4.8(b)).

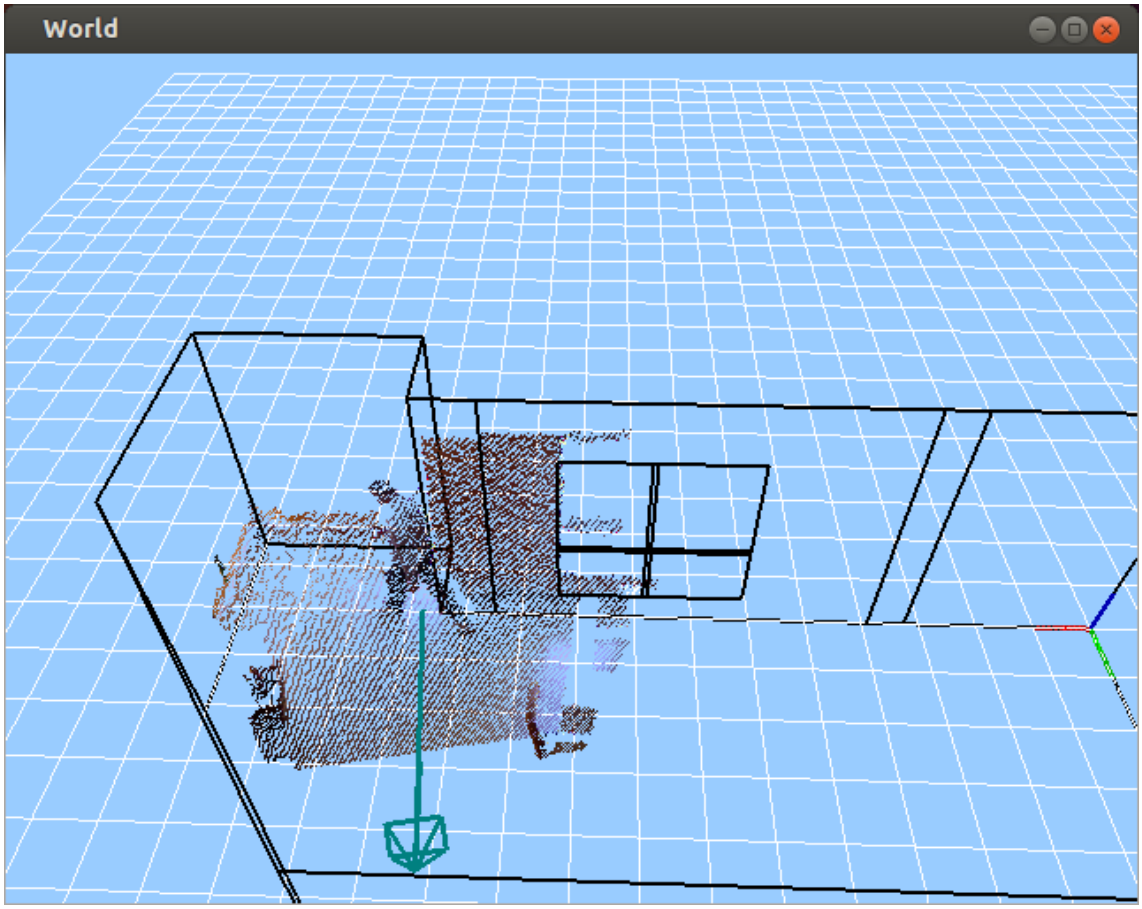
RgbdViewer nos ofrece algunas funciones útiles para verificar la correcta calibración de nuestro sistema:

- **Visualizar imágenes RGB y D**
  La funcionalidad básica de este componente es representar gráficamente en un interfaz de usuario, tanto la imagen de color, como la imagen de profundidad, ambas ofrecidas por el componente *OpenNIServer*. Esta representación se realiza de forma transparente, sin realizar ningún tipo de filtrado a las imágenes, sin embargo, si que realiza un redimensionamiento de las mismas si el tamaño es superior a 320x240, indicando este proceso con un texto superpuesto en las imágenes. Este cambio de tamaño se lleva a cabo para poder visualizar todas las imágenes de forma correcta ya que, si estas son muy grandes, puede que no se vean de forma completa en el monitor. Este proceso es totalmente configurable y puede ser desactivado. Junto con la información del redimensionamiento también se superpone la frecuencia a la que se están capturando las imágenes.

- **Representar la estructura del entorno en 3D**
  Este componente ofrece una serie de funcionalidades para realizar representaciones 3D utilizando OPENGL. La primera de ellas es poder cargar ficheros con descripciones del mundo y representarlos en el visor. Este fichero es fácilmente editable, ya que está basado únicamente en segmentos definidos por dos puntos (cuadro 4.3).



(a)



(b)

Figura 4.8: (a) Imágenes (b) Nube de puntos

```
#Puerta
#worldline P1x P1y P1z P1h P2x P2y P2z P2h
worldline 6715 0 0 1 6715 0 3000 1
worldline 4850 0 0 1 4850 0 3000 1
worldline 6715 0 2035 1 4850 0 2035 1
worldline 5815 0 0 1 5815 0 2035 1
worldline 5819 0 0 1 5819 0 2035 1
```

Cuadro 4.3: Ejemplo de fichero con descripciones del mundo

La segunda funcionalidad útil en el ámbito del presente proyecto es poder representar los puntos puntos 3D calculados por el componente *OpenNIServer* con los datos del sensor en el propio visor junto con la descripción del mundo (figura 4.8(b)). Esto nos permite tomar como referencia el mundo para situar objetos y comprobar el correcto funcionamiento de todo el sistema.

- **Proyectar el entorno sobre las imágenes ofrecidas por el sensor**
  A través de *progeo*, *rgbViewer* es capaz de proyectar el mundo representado en ficheros, explicados en el punto anterior, en cada una de las imágenes del visor. Esta funcionalidad es útil a la hora de comprobar lo buena que es la calibración que estamos utilizando (figura 4.13).

- **Representar en 3D la recta de retroproyección al hacer *click* en un píxel**
  Otra manera muy visual para ver la calidad de la calibración de un sistema es seleccionar en un píxel en la imagen RGB y comprobar si la recta de retroproyección encaja en nuestro mundo. En el ejemplo de la figura 4.11(a), se han seleccionado 3 píxeles y como se puede confirmar en la figura 4.11(b), las rectas encajan con nuestra representación del entorno 3D.

RgbdViewer está también diseñado para optimizar su funcionamiento con sistemas con varios núcleos, por lo que su funcionamiento ha sido dividido en dos hilos: el primero de ellos, encargado de pedir los datos al componente *OpenNIServer* y alojar estos datos en memoria compartida protegida tras un mutex, y un segundo hilo, dedicado a mostrar los datos en el interfaz gráfico (figura 4.10).

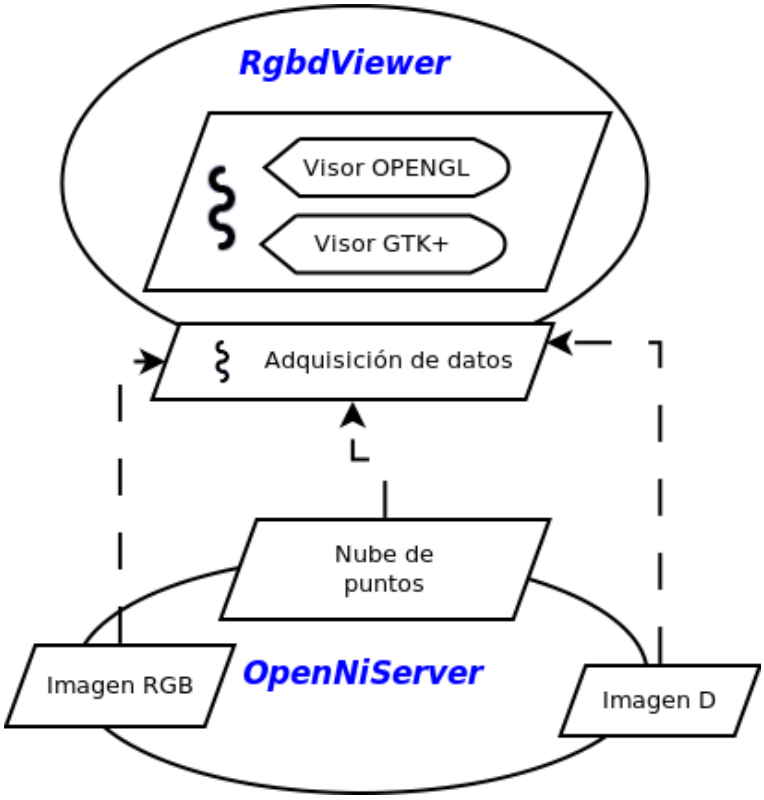


Figura 4.9: Conexión del componente RgbdViewer y estructura interna

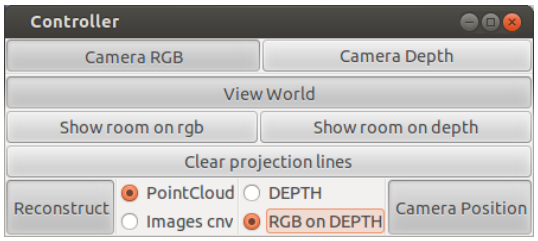
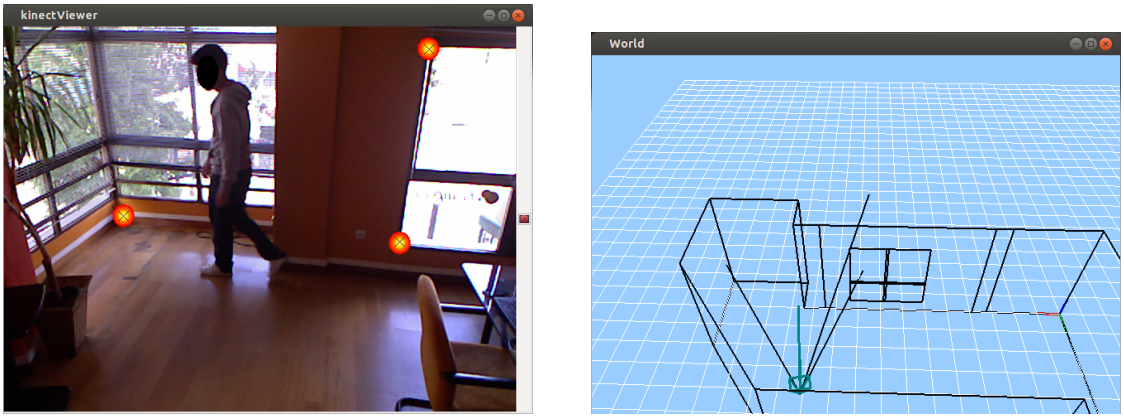


Figura 4.10: Interfaz gráfico del componente rgdbViewer



(a) (b)

Figura 4.11: (a) Píxeles seleccionados (b) Rectas de retroproyección en 3D

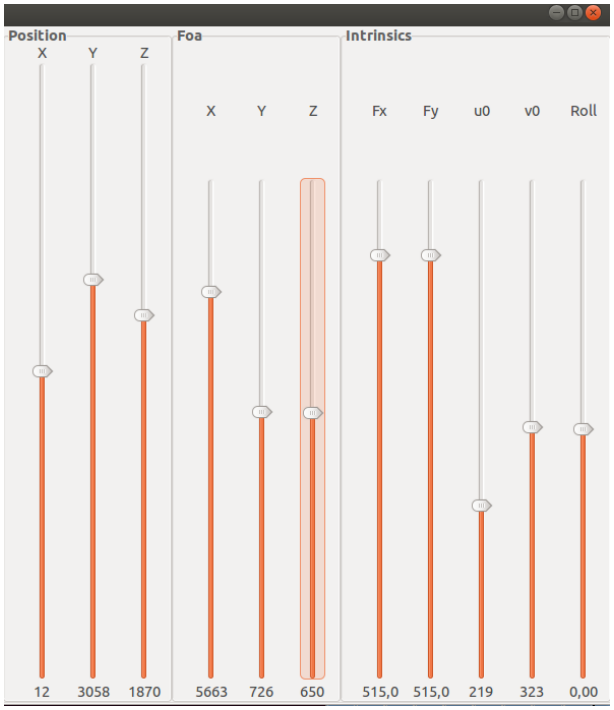


Figura 4.12: Interfaz gráfico del calibrator

4.3. Componente RgbdManualCalibrator

Como ya hemos introducido en el apartado 3.8, los parámetros de calibración de una cámara se pueden clasificar en intrínsecos y extrínsecos. Los intrínsecos son aquellos que definen el funcionamiento interno del sensor, así como la geometría de la discretización. Por otra parte, los extrínsecos son los que describen la posición y orientación del sistema de referencia de la cámara en 3D. Hay multitud de formas de definir estos parámetros, *progeo* utiliza los siguientes:

- **Parámetros intrínsecos:** Distancia focal (f_x,f_y) y la localización del punto central de la imagen (u0,v0).
- **Parámetros extrínsecos:** Posición de la cámara (x,y,z), foco de atención (x,y,z) y giro.

El componente *rgbdManualCalibrator* es una herramienta de calibración completamente manual que permite ajustar cada uno de los parámetros de la cámara. Para ello, es necesario conocer previamente la geometría del entorno en el que se van a calibrar los sensores. Utilizando la funcionalidad de cargar entornos en 3D, presentada en la sección anterior y que se incluye el componente *RgbdViewer*, simularemos una cámara que esté situada en ese escenario y con los parámetros de calibración que el usuario seleccione desde el interfaz gráfico (figura 4.12). Estos parámetros se pueden variar interactivamente, y el propio componente *RgbdManualCalibrator* se encarga, en cada iteración, de simular una nueva cámara. Para comprobar la correcta calibración de la cámara se retropoyectan todos los segmentos que se verían desde esta cámara simulada en la imagen proveniente del sensor (figura 4.13). Si las proyecciones

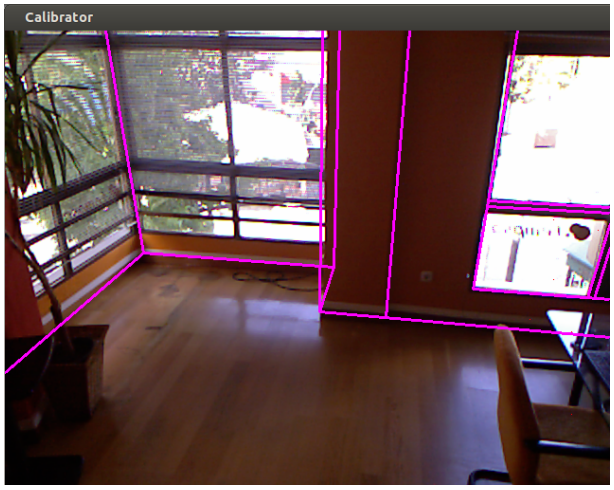


Figura 4.13: Proyección del entorno en la imagen de color.

concuerdan, entonces significará que la cámara está bien calibrada. Como hemos comentado en secciones anteriores, la imagen de color y el mapa de profundidad están registradas. Por lo que bastaría con realizar este proceso de calibración con cualquiera de las dos imágenes. *rgbdManualCalibrator* permite ver las proyecciones en cualquiera de las dos imágenes.

Una vez terminado este proceso inicial de calibración, *rgbdManualCalibrator* ofrece una serie de herramientas para comprobar su calidad. La primera de ellas es la posibilidad de calcular la nube de puntos a partir de las imágenes que recibe el componente en función de los parámetros seleccionados. Estos puntos se muestran en un visor de OpenGL junto con el mapa del entorno, de modo similar al componente *RgbdViewer*, y podemos ver si nuestra reconstrucción encaja o no.

Este componente es capaz de trabajar con múltiples dispositivos, siendo necesario realizar el proceso de calibración manual en cada una de ellas. A su vez, ofrece herramientas para comprobar cómo se ajusta la nube de puntos de un sensor con la de otro en zonas de solapamiento (zonas que son reconstruidas desde dos sensores simultáneamente). Para facilitar este proceso se puede representar cada nube con un color distinto (figura 4.14).

Una vez dada por terminada la calibración, *RgbdManualCalibration* es capaz de volcar los resultados de la calibración a un fichero en disco con un formato xml, fácilmente digerible por cualquier componente que necesite utilizar cámaras calibradas.

Al igual que en los componentes explicados en los apartados anteriores, el componente *RgbdManualCalibrator* ha sido implementado con dos hilos de ejecución (figura 4.15), uno que se encarga de la captura de datos de todos los sensores, en este caso, sólo se captura la imagen de color y la imagen de profundidad, alojando todos estos datos en memoria compartida, protegiéndolos frente al acceso concurrente. El segundo hilo se encarga de calcular iterativamente la nube de puntos correspondiente a

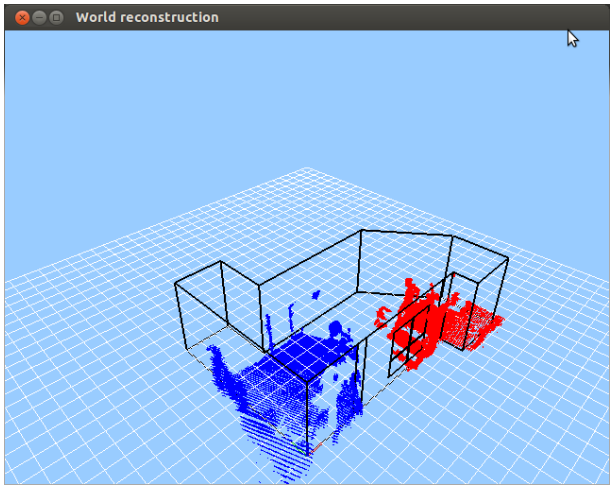


Figura 4.14: Nubes de puntos con distintos colores.

cada sensor, con los parámetros de calibración seleccionados en el interfaz de usuario, y representarlos en el propio interfaz.

4.4. Componente RemoteConfigurator

Todos los componentes de *JdeRobot* tienen una serie de parámetros de configuración que son cargados al inicio de su ejecución. En algunos casos, es necesario variar algunos de ellos para modificar su comportamiento en tiempo de ejecución. La opción que se utilizaba hasta la fecha era añadir un interfaz gráfico al propio componente e incorporarle una serie de entradas. Este tipo de configuración es útil siempre y cuando, tengamos acceso al equipo local donde se está ejecutando el componente, y dispongamos de una pantalla y un teclado como mínimo. La complicación viene precisamente al querer configurar un componente que está corriendo en un servidor sin pantalla, o bien, en un equipo remoto al que no tenemos acceso físico.

remoteConfigurator es un componente que permite la configuración de cualquier componente de *JdeRobot* de forma remota. Para que este componente sea lo más genérico posible, se ha basado su configuración en dos ficheros xml. El primer fichero xml, define la estructura y jerarquía de los parámetros a configurar, así como su tipo de dato (se puede apreciar un ejemplo en el código 4.4). A partir de este fichero, se genera de forma dinámica un interfaz gráfico. En él, el usuario puede fijar cada uno de los parámetros (figura 4.16). Al terminar la configuración *remoteConfigurator* genera un fichero con la misma estructura del primer xml pero esta vez con los datos que ha introducido el usuario gráficamente (código 4.5). Los tipos de datos soportados en este componente son dos:

1. **Booleanos:** son representados en el interfaz con una casilla de verificación.

2. **Numéricos:** son convertidos al tipo correspondiente desde una entrada de texto en el interfaz.

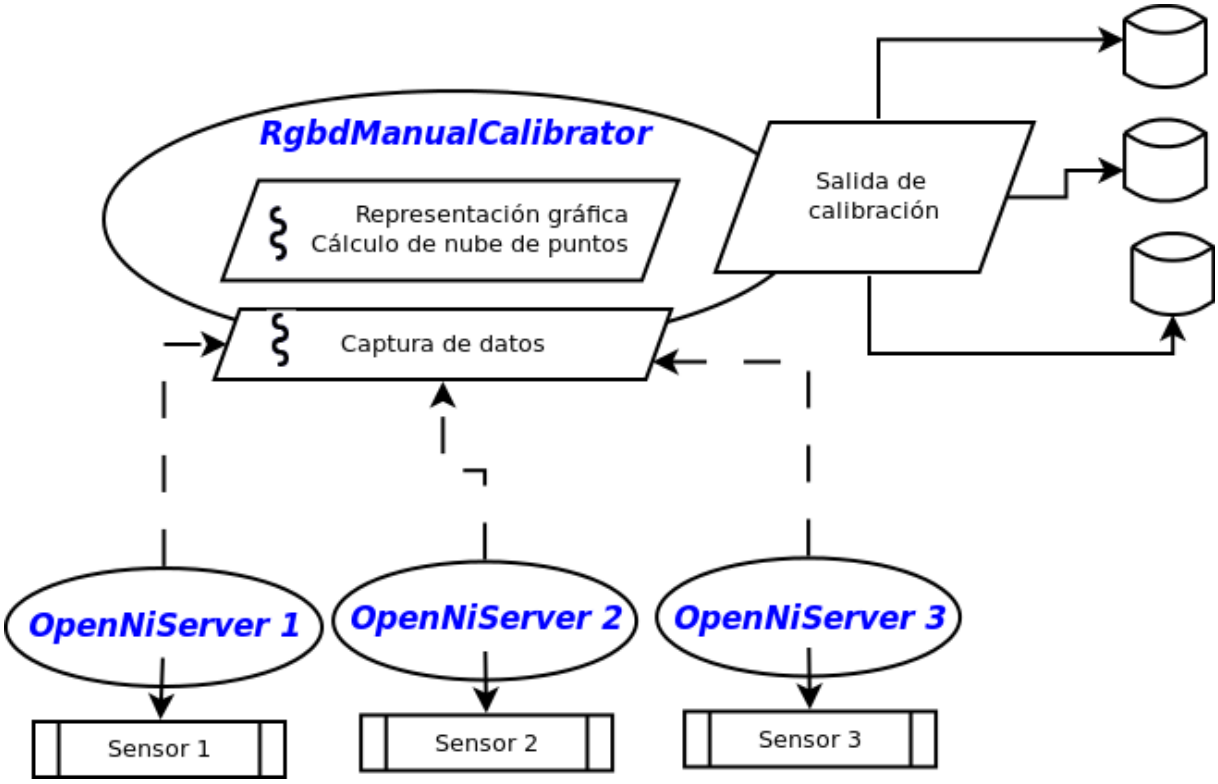


Figura 4.15: Conexión del componente RgbdManualCalibrator y estructura interna

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<myApp>
    <Configuration>
        <GUI_options>
            <GUI>toggle</GUI>
        </GUI_options>
        <MyParameter1>inputbox</MyParameter1>
    </Configuration>
    <ExtraConfig>
        <Position>
            <x>inputbox</x>
            <y>inputbox</y>
            <r>inputbox</r>
            <save>button</save>
        </Position>
    </ExtraConfig>
</myApp>
```

Cuadro 4.4: Fichero xml de entrada para remoteConfigurator

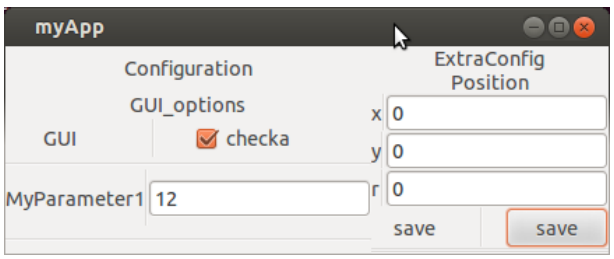


Figura 4.16: Interfaz gráfico a partir del xml de entrada

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<myApp>
    <Configuration>
        <GUI_options>
            <GUI>1</GUI>
        </GUI_options>
        <MyParameter1>a</MyParameter1>
    </Configuration>
    <ExtraConfig>
        <Position>
            <x>0</x>
            <y>0</y>
            <r>0</r>
        </Position>
    </ExtraConfig>
</myApp>
```

Cuadro 4.5: Fichero xml de salida con la configuración obtenida del gui

Una vez generado el segundo xml, el propio *remoteConfigurator* se encarga de mandarlo al componente remoto, el cual se encargará de analizarlo y modificar su comportamiento según los parámetros que en él se establezcan. El envío del documento se realiza utilizando un interfaz Ice dedicado (cuadro 4.6), siendo necesario seguir los siguientes pasos:

1. **Inicializar la configuración** Es necesario comunicar el componente a configurar que se le va a enviar un fichero de configuración, para asegurarnos de que el componente que ha iniciado la configuración es el que nos va a mandar el fichero se le asigna un número aleatorio, que es el que devuelve el método de inicialización.

2. **Envío del documento** El documento se envía línea por línea junto con el identificador. El componente que va recibiendo las líneas las guarda en el un fichero temporal.

3. **Envío de confirmación** Una vez se ha mandado el documento completo, es necesario llamar a un método con el identificador correspondiente para confirmar la configuración. Una vez que el componente a configurar recibe esta confirmación, cierra el fichero temporal y lo analiza fijando su comportamiento en función de los parámetros que ha recibido en el fichero.

```
module jderobot{

    interface remoteConfig{

        int initConfiguration();
        int write (string data, int id);
        int setConfiguration(int id);
    };
};
```

Cuadro 4.6: Interfaz Ice para la configuración remota

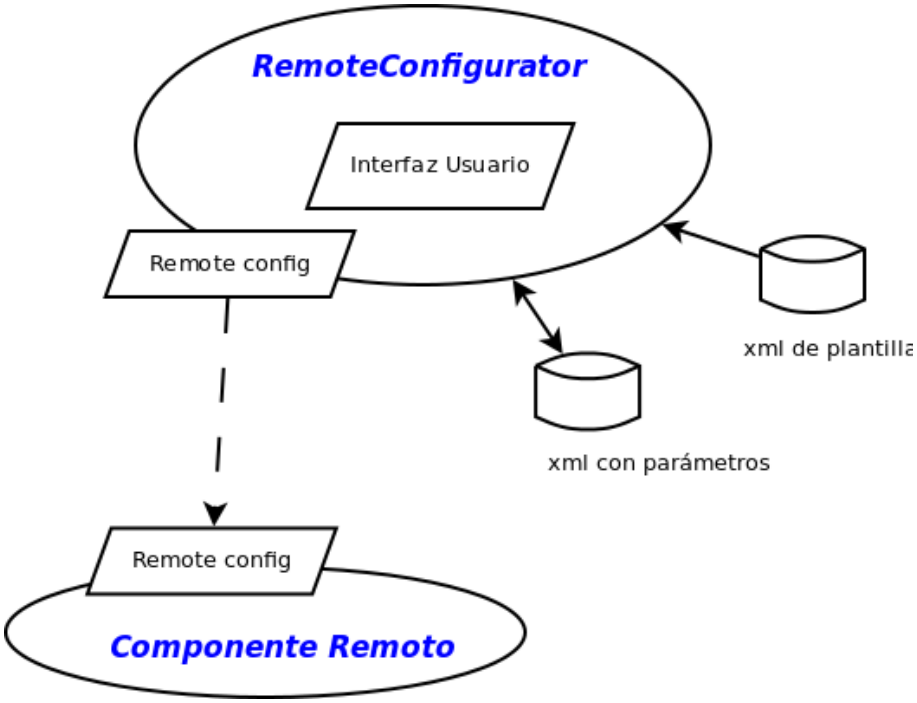


Figura 4.17: Esquema de la configuración remota de componentes

El esquema genérico utilizando *RemoteConfigurator* para configurar componentes remotamente se puede apreciar en la figura 4.17, donde el componente se apoya en dos ficheros xml auxiliares, uno con la plantilla para el interfaz gráfico y otro con los parámetros fijados, siendo este último enviado al componente remoto a configurar utilizando el interfaz *remoteConfig*.

Capítulo 5
Seguimiento 3D

Después de la primera parte del documento, en la que se han detallado las herramientas creadas para dar soporte a este proyecto, se detallará en este capítulo cómo se ha implementado el sistema de detección y seguimiento de personas 3D desarrollado, así como las diferentes situaciones de riesgo que es capaz de detectar.

El sistema se basa en el procesamiento de información 3D tratada como una nube de puntos. Estos datos son obtenidos desde *OpenNIServer*, y como hemos explicando anteriormente, este componente es capaz de ofrecer una nube de puntos con un identificador único para cada una de las personas detectadas en el entorno. La idea fundamental del algoritmo desarrollado es tratar de explicar con prismas de volumen variable a cada una de las personas detectadas en el entorno para realizar un seguimiento 3D de las mismas continuado en el tiempo.

5.1. Diseño

La estructura del sistema implementado sigue un modelo distribuido, utilizando la red para conectar cada uno de los componentes. Esto permite tener múltiples nodos satélites situados en diferentes estancias de una casa, que transmiten la información de cada sensor a un nodo principal, el cual se encarga de procesarla y detectar las diferentes situaciones de riesgo (figura 5.1).

En la sección 3.2, se han descrito las características de los sensores utilizados para este proyecto. Estos sensores tienen una serie de problemas con los que se ha tenido que lidiar. Los que más pueden afectar a esta aplicación son el ruido intrínseco del sensor, las oscilaciones en las mediciones y la precisión espacial. Por ello, es necesario introducir un filtrado para que estos fenómenos afecten lo menos posible a los resultados y conseguir que sea lo más fiable y robusto posible.

Dentro del componente *ElderCare*, la funcionalidad se ha dividido en módulos tal y como se puede apreciar en la figura 5.2. Un módulo se encarga de pedir los datos a los diferentes componentes *OpenNIServer* y alojarlos en memoria compartida. Tres módulos principales son el módulo del interfaz gráfico o GUI, el módulo que realiza todas las tareas de seguimiento tridimensional y finalmente el módulo que genera

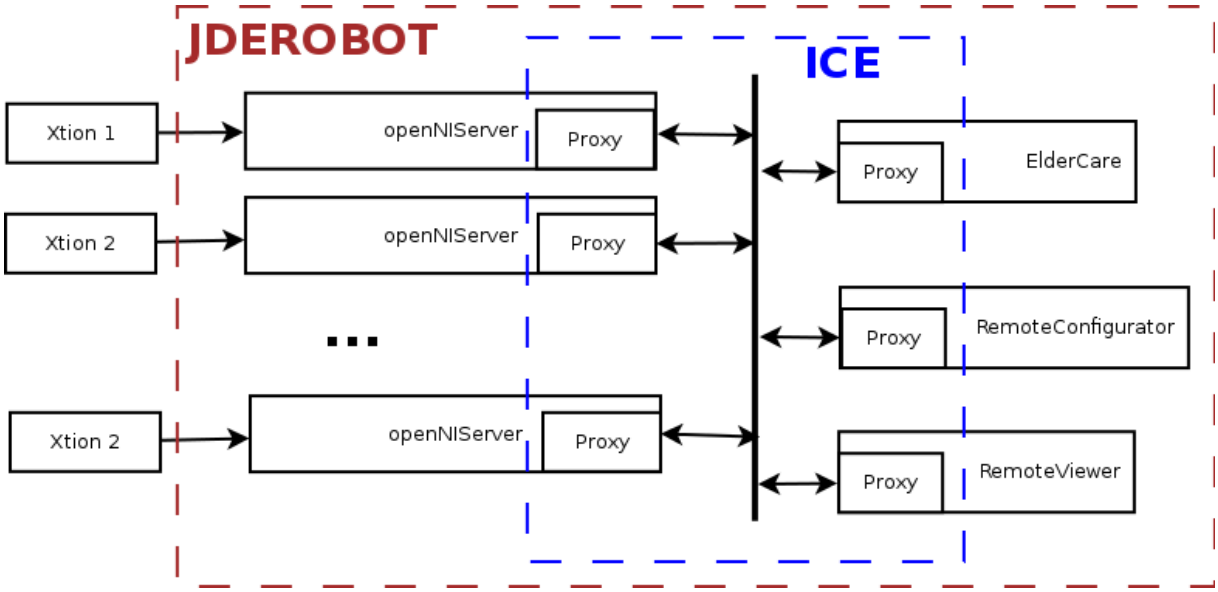


Figura 5.1: Interconexión de los componentes desarrollados en ElderCare

y gestiona las diferentes situaciones de riesgo y alarmas. Por otra parte, *ElderCare* incluye dos módulos dedicados exclusivamente a la comunicación con componentes externos de *JdeRobot*: el módulo de configuración y el módulo de visualización remota que se detallará en este mismo capítulo. 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 multicore, la aplicación desarrollada pueda ser ejecutada de forma muy ligera llegando al tiempo real.

En el módulo de seguimiento de personas 3D se ha implementado un algoritmo que sigue los pasos indicados en la figura 5.3. Este algoritmo constituye el corazón del sistema y sus pasos serán explicados en detalle en los siguientes apartados.

5.2. División de la nube de puntos en prismas

En este primer paso del algoritmo, el programa recibe datos de todos los sensores que estén conectados al sistema y divide los datos en nubes de puntos individuales. Por cada sensor recibiremos una única nube de puntos con toda la información. Cada punto tiene información de color, de posición 3D y un identificador. El sistema presentado sólo utilizará el identificador y la posición 3D, con la finalidad de que el seguimiento siga funcionando, aun

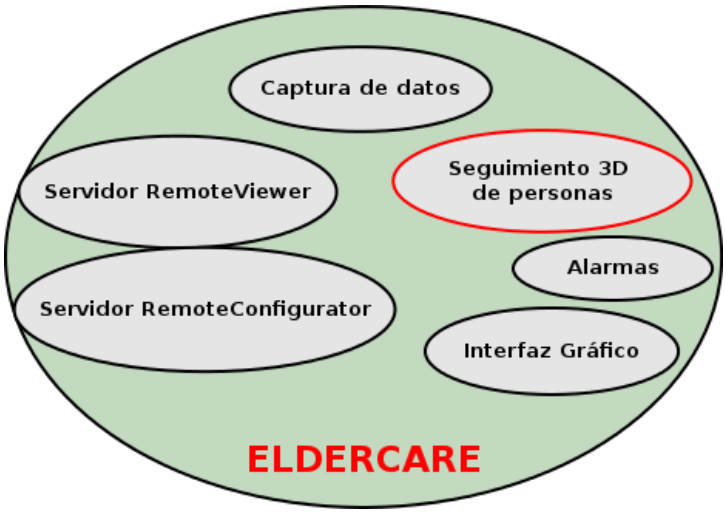


Figura 5.2: Estructura interna del componente ElderCare

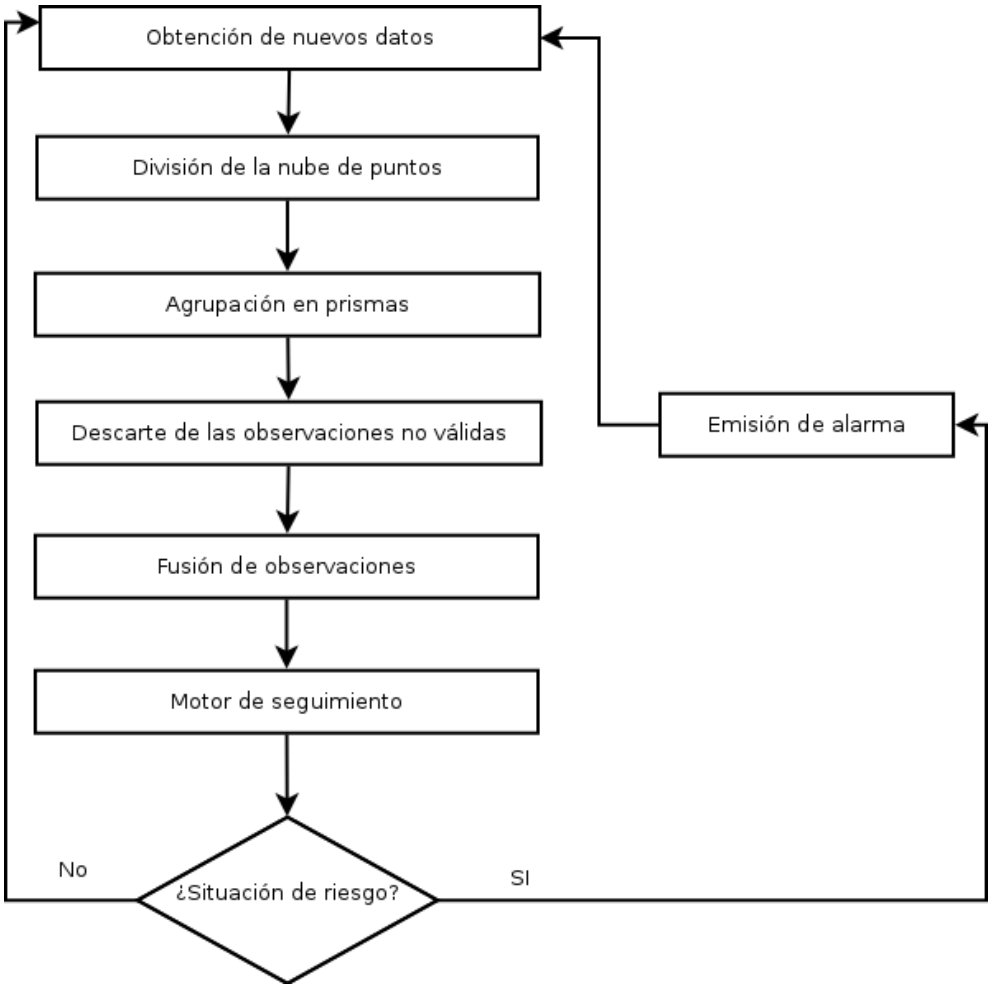
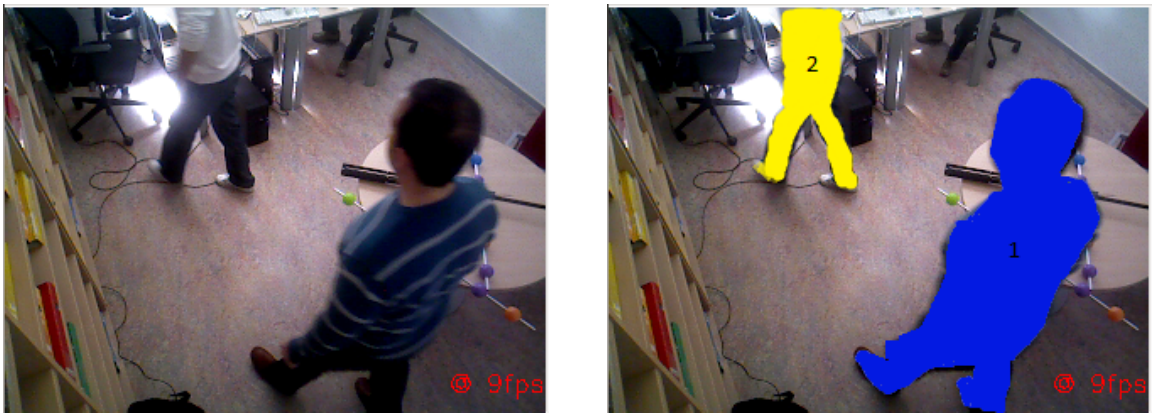


Figura 5.3: Secuencia del algoritmo del módulo de seguimiento 3D

En la figura 5.4, se puede ver un ejemplo gráfico de lo explicado anteriormente. El entorno expuesto es una habitación normal en la que se han colocado dos sensores, siendo las imágenes de la figura 5.4(a) y la figura 5.4(c) las imágenes de color obtenidas por los sensores. En las figuras 5.4(b) y 5.4(d), podemos ver la segmentación de NITE superpuesta sobre las imágenes fuente, utilizando la funcionalidad integrada en el componente *OpenNIServer*. Se puede comprobar que hay dos personas en el entorno y están siendo captadas de forma simultánea por los dos sensores. Para comprobar el funcionamiento y visualizar los pasos seguidos por el algoritmo se utilizará una representación 3D de la nube de puntos con los mismos colores utilizados en la imágenes segmentadas. En la figura 5.5, se puede observar el resultado de reconstruir la escena en 3D con estas premisas, rechazando todos los puntos que no corresponden a ninguna persona (cualquier punto no asociado a alguna de las personas detectadas en las imágenes). Aunque la detección de NITE detecta de forma correcta, con gran robustez, las personas que se encuentran en el entorno monitorizado, la segmentación espacial no es del todo precisa. Como se puede ver en la propia figura 5.5, hay numerosos díscolos, puntos que NITE asigna a una persona pero que realmente no son parte de ella. Estos datos incorrectos se suelen generar alrededor de la silueta de las personas y es notorio cuando la persona en cuestión se mueve de forma veloz.

Como se ha introducido al inicio de este capítulo, el sistema presentado trata de explicar con prismas a cada una de las personas que se encuentren en el entorno y utilizar esta primitiva tridimensional para implementar un motor de seguimiento. Al aplicar esta idea sobre las nubes independientes (cada una con un color diferente en la figura 5.6), se consigue un prisma para cada nube que engloba todos y cada uno de los puntos de esa nube. El resultado de aplicar este procedimiento sobre los datos crudos se puede visualizar en la figura 5.6.

El paso siguiente consiste en combinar las nubes que sean susceptibles de pertenecer a la misma persona. Debido a que nuestro sistema puede estar utilizando múltiples dispositivos, una misma persona puede ser captada por varios sensores simultáneamente. Para no tener múltiples descripciones de la misma persona, es necesario combinar los datos que pertenezcan a un mismo sujeto. Una forma básica de combinar observaciones es unir las nubes que estén muy cercanas una de otra o incluso se solapen. Aplicar este proceso sobre los resultados de la figura 5.6 es harto complicado. Como se puede observar, numerosos prismas se solapan, incluso prismas de personas distintas. Para poder aplicar algún tipo de combinación por distancia, es necesario un proceso de filtrado que permita que los prismas que albergan a las nubes independientes estén más concentrados. Este proceso elimina puntos que realmente no pertenecen al sujeto, al cual están asociados, mejorando la precisión general del algoritmo.



(a) (b)

(c) (d)

Figura 5.4: (a) Imagen cámara 1 (b) Imagen cámara 1 segmentada (c) Imagen cámara 2 (d) Imagen cámara 2 segmentada

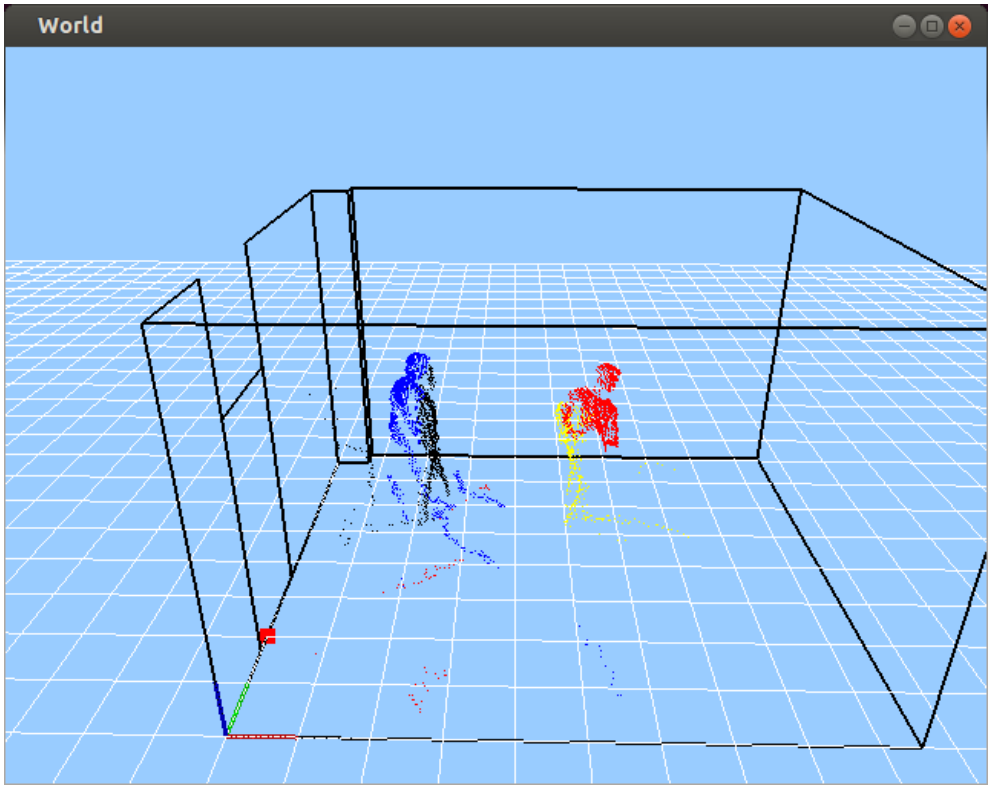


Figura 5.5: Representación 3D de la segmentación de las personas en todos los sensores

5.3. Agrupamiento en prismas

En este punto, tras aplicar el proceso explicado en el apartado anterior, el resultado es una serie de nube de puntos independientes correspondiente a las diferentes detecciones en cada uno de los sensores. Como se ha explicado, tratar de utilizar un prisma que englobe directamente a todos los puntos de cada nube independiente no es viable debido a que las propias observaciones son afectadas por el ruido del sensor. Por ello, se estudiaron una serie de técnicas de filtrado para tratar de reducir el tamaño de estos prismas al mínimo, pero manteniendo un número de puntos suficientemente grande para ser representativo.

5.3.1. RANSAC

El primer método estudiado fue *RANdom SAmple Consensus* (RANSAC [Fischler y Bolles, 1981]). Este método es muy utilizado a la hora de estimar parámetros de modelos matemáticos a partir de una serie de datos observados. Esta técnica estima el modelo buscado utilizando un subconjunto de los datos observados seleccionados aleatoriamente. A cada uno de estos modelos le asigna un valor de calidad y devuelve aquel modelo que haya obtenido una mayor puntuación.

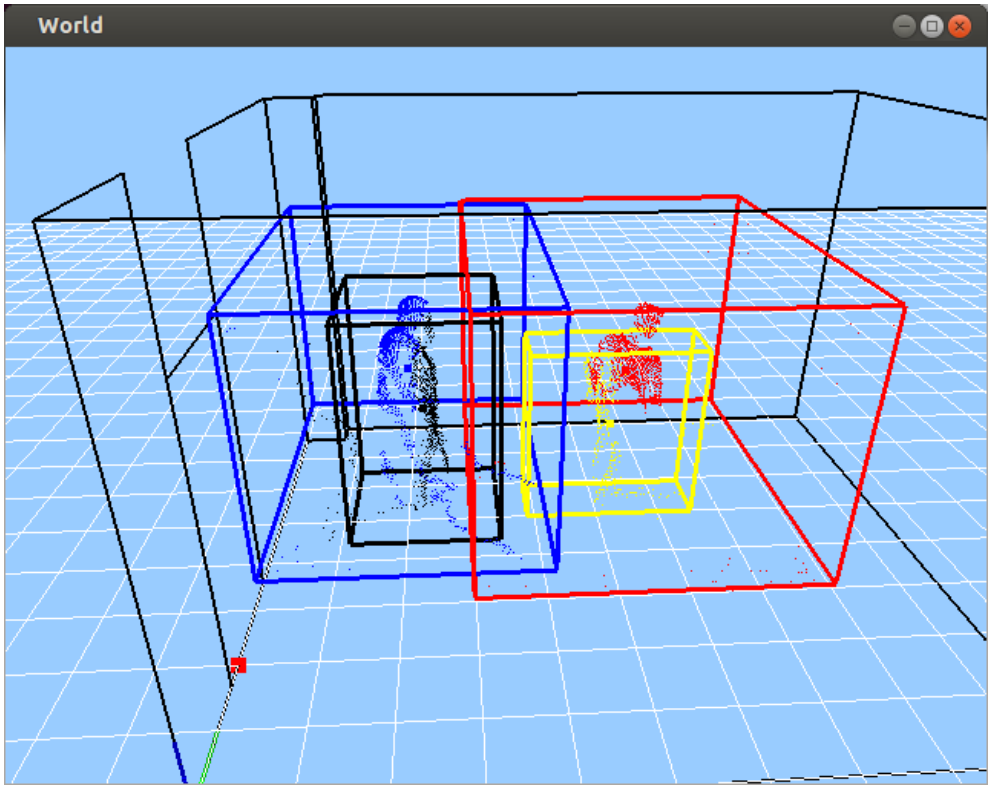


Figura 5.6: Datos crudos englobados en prismas

Uno de los motivos principales por lo que *RANSAC* está muy extendido en el ámbito de la robótica y la visión computacional, es la robustez que presenta frente al ruido existente en las observaciones. Al evaluar el modelo con varios subconjuntos de los datos fuente, RANSAC es capaz de descartar las posibles soluciones que han sido muy afectadas por los *outliers*. Para ello, es necesario fijar una función que evalúe qué calidad tiene el modelo calculado con cada una de las soluciones tentativas.

En nuestro caso en particular, el modelo matemático que se pretende evaluar es un prisma rectangular o cuboide, modelado como un punto en tres dimensiones (x,y,z), que corresponde al punto central del prisma, y su volumen definido como la desviación en cada una de las dimensiones (dx,dy,dz), tal y como refleja la figura 5.7.

Para que *RANSAC* funcione de forma adecuada es necesario fijar una función de calidad que sea robusta y que ajuste el método a la solución esperada. Esta solución debe filtrar el modelo de *outliers* condensando el prisma a la zona donde están concentrados la mayoría de los puntos. Una función con la que se han conseguido buenos resultados y que se ajusta bien a nuestras necesidades es la presentada en la ecuación (5.1), donde la *densidad* es el número de puntos por unidad de volumen, la *dispersión* es calculada como la desviación típica de todos los puntos que engloba y la *completitud* es el ratio entre los puntos totales y los puntos englobados por el prisma en cuestión. Este valor de calidad prima la densidad del prisma sobre la dispersión pero manteniendo cierta completitud de la observación inicial. Es decir, la solución

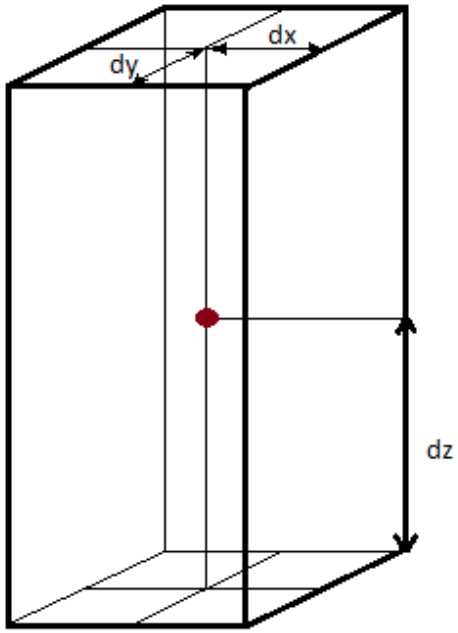


Figura 5.7: Modelado de un prisma en 3D

que obtiene este método es aquel prisma que, siendo evaluado sobre la nube de puntos inicial, es denso, tiene poca dispersión pero que sigue englobando bastantes puntos de ella.

$$calidad = \frac{densidad}{dispersion} * completitud \tag{5.1}$$

Esta función de calidad es evaluada sobre **n** subconjuntos (**n** iteraciones de RANSAC) de puntos de la siguiente manera:

1. Se seleccionan **k** puntos aleatoriamente.
2. Se calcula el prisma de menor tamaño que contiene a dichos puntos.
3. Se evalúa la función de calidad sobre la nube inicial. Donde los valores de *densidad*, *dispersión* y *completitud* de este prisma son calculados con todos los puntos de la observación individual.

Una vez terminado este proceso iterativo, el resultado del filtrado será aquel prisma que obtenga un valor más alto en la función de calidad.

En la figura 5.8, se puede observar el resultado sobre el escenario tomado como ejemplo en el apartado anterior. En él, se puede comprobar cómo aplicando este proceso de filtrado obtenemos los resultados deseados, prismas que engloban la gran mayoría de los puntos pertenecientes a la persona de la observación, sin verse afectados por

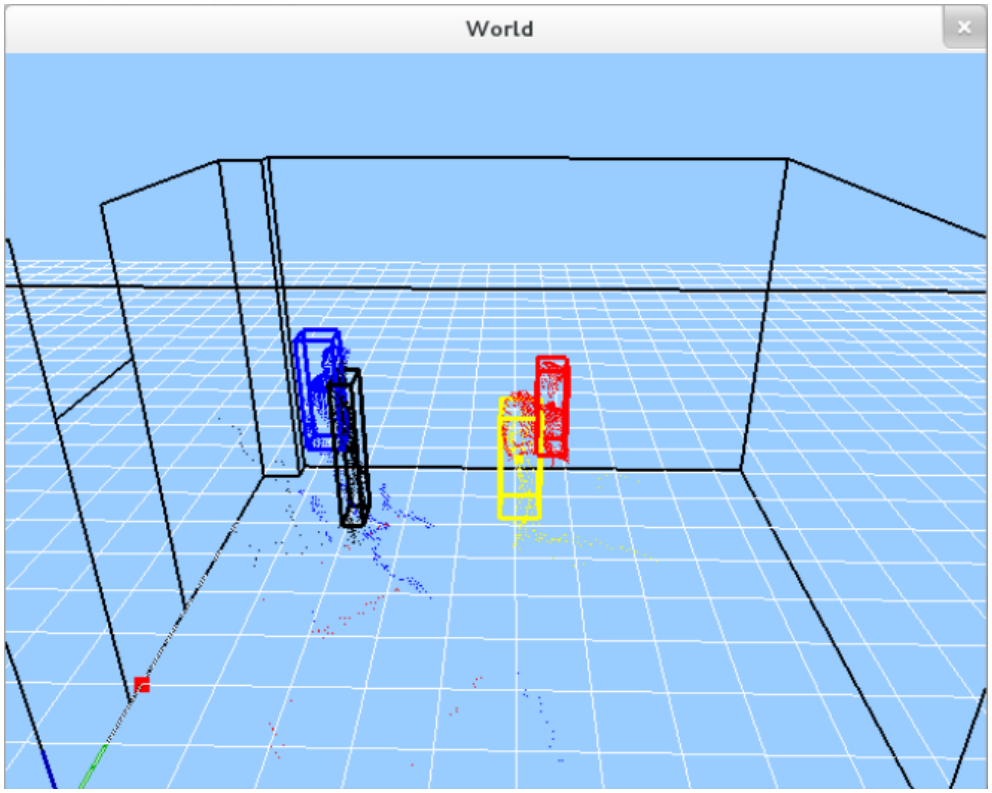


Figura 5.8: Resultado del filtrado con RANSAC

puntos espurios. En esta figura, al contrario que en la figura 5.6, se puede comprobar de forma intuitiva qué prismas son susceptibles de pertenecer a la misma persona.

5.3.2. Basado en histogramas

Otro método explorado para tratar dispersión de las nubes de puntos individuales y la forma en las que se generaba el ruido, generalmente disperso fuera del núcleo de concentración, es la utilización de histogramas. Conociendo las dimensiones del entorno monitorizado, se puede representar la dispersión de los puntos de cada nube individual utilizando un histograma para cada dimensión y utilizar algún tipo de umbralizado para acotarlo a zonas densas.

En la figura 5.9, se representa una nube individual en un histograma. En verde están delimitadas las dimensiones del prisma que englobaría todos los datos del nube. Como se puede apreciar a simple vista, los histogramas x e y están afectadas por puntos espurios (representados en rojo), lo cual influye en gran medida en el resultado final a la hora de calcular las dimensiones del prima.

El resultado deseado es un histograma acotado que devuelva únicamente zonas densas del entorno que contengan un número significativo de datos, descartando puntos alejados del núcleo. Para conseguir esta meta, se abordó dicha problemática pero desde una perspectiva inversa, es decir, implementar un sistema de umbralizado que no trate

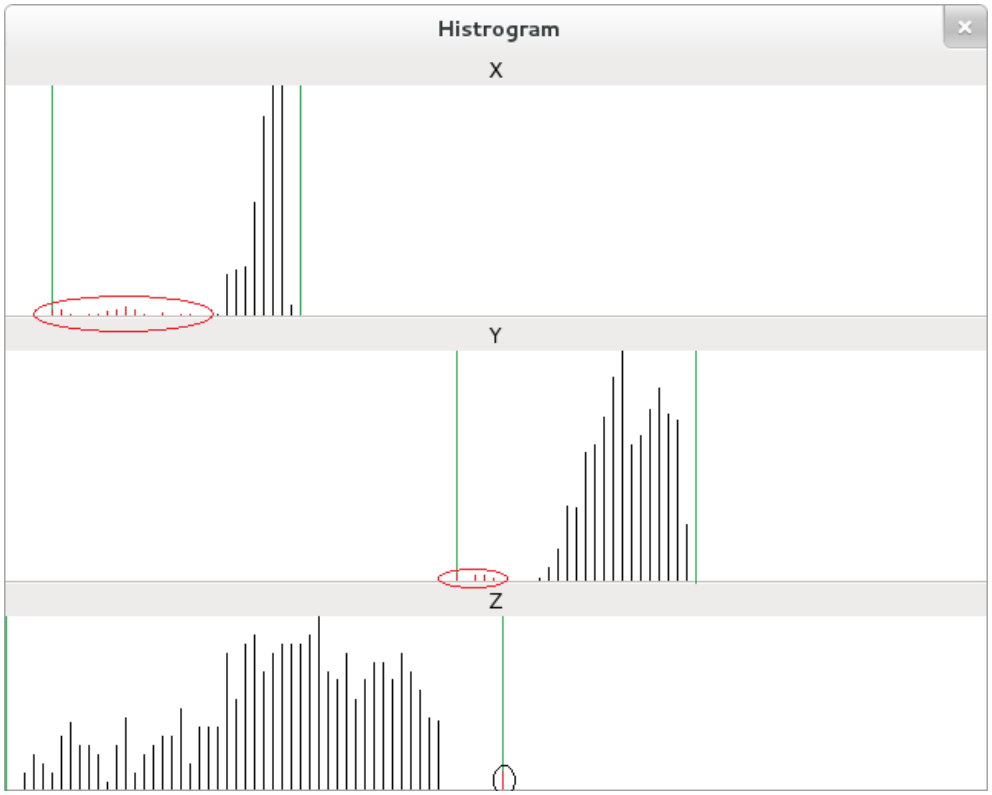


Figura 5.9: Representación de los datos en histogramas

de acotar el histograma partiendo de todos los datos sino justo lo contrario, tratar de expandir el histograma partiendo de una zona densa. Los pasos seguidos para implementar este umbralizado por expansión, son los siguientes:

1. Se calcula el histograma de acumulación normalizado.

2. Se acota la zona en la que se encuentra el 40 % de los datos, desde el 30 % al 70 % en el histograma de acumulación. Esta zona corresponde con la franja amarilla en la figura 5.10.

3. Se expande el histograma intervalo a intervalo, primero por la cota inferior, y luego por la superior, siempre y cuando la ganancia de incorporar ese *bin* sea mayor que cierto umbral.

4. Este proceso se repite para cada dimensión. El resultado obtenido de aplicar este algoritmo sobre la figura 5.9, es el representado en la figura 5.10. En esta figura se puede ver el foco de partida (la zona en amarillo) y el resultado tras las expansión (la zona verde), el resto es tomado como ruido y no se tendrá en cuenta a la hora de fijar el prisma. El umbral utilizado para aceptar una expansión es que el bin a incorporar englobe al menos el 5 % de los puntos totales de la nube, lo que significa que estamos incorporado a una zona que presuponemos densa, un intervalo con bastantes puntos.

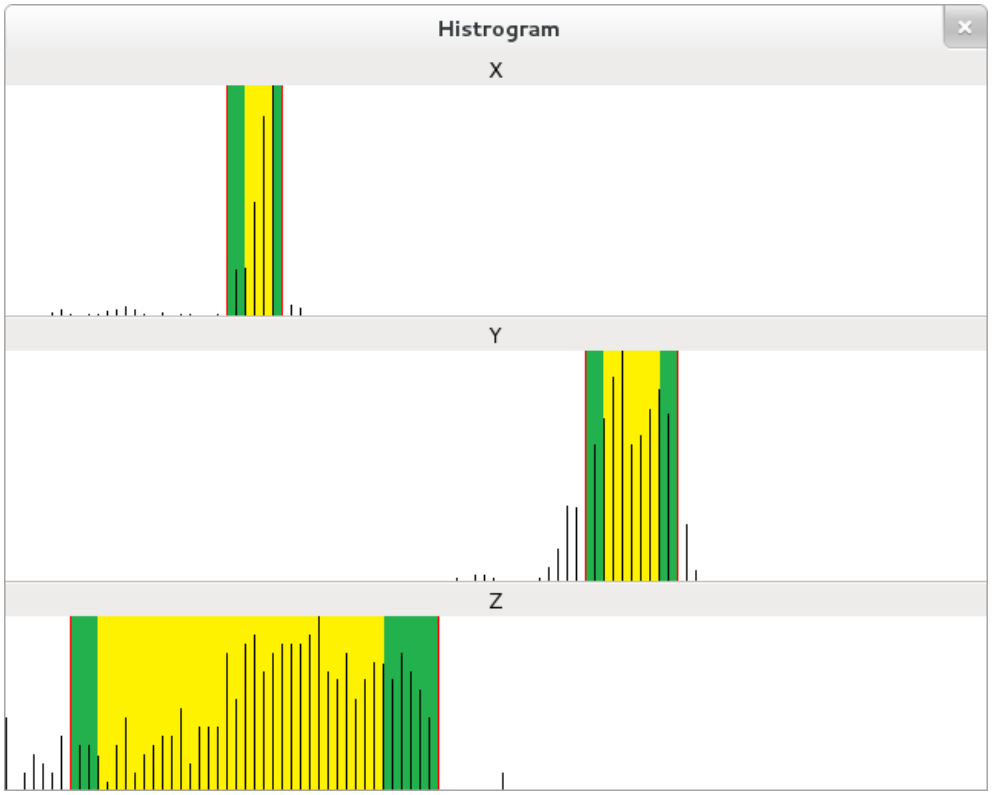


Figura 5.10: Histogramas acotados

Esta técnica depende de una serie de parámetros que son necesarios fijar en función de los diferentes entornos donde se instale el sistema:

- Dimensiones de la zona monitorizada.
- Número de *bins* del histograma.
- Umbral de porcentaje de puntos para incluir el *bin* a la *zona densa*. Este valor ha sido fijado en todas las pruebas realizadas a un 5 %, pero en algunos entornos es posible que sea necesario volver a estudiar el valor idóneo ya que depende mucho del número de divisiones que se estén utilizando y, por supuesto, del tamaño del entorno.

Los prismas obtenidos tras aplicar este proceso de agrupamiento (figura 5.11) es similar al obtenido con RANSAC. En el trabajo presentado por Marta Martínez [Martínez de la Casa, 204 205], también se aplicó con éxito un umbralizado de este tipo, aunque en su caso caso aplicado a la segmentación de objetos basados en color y se utilizaba un umbralizado doble.

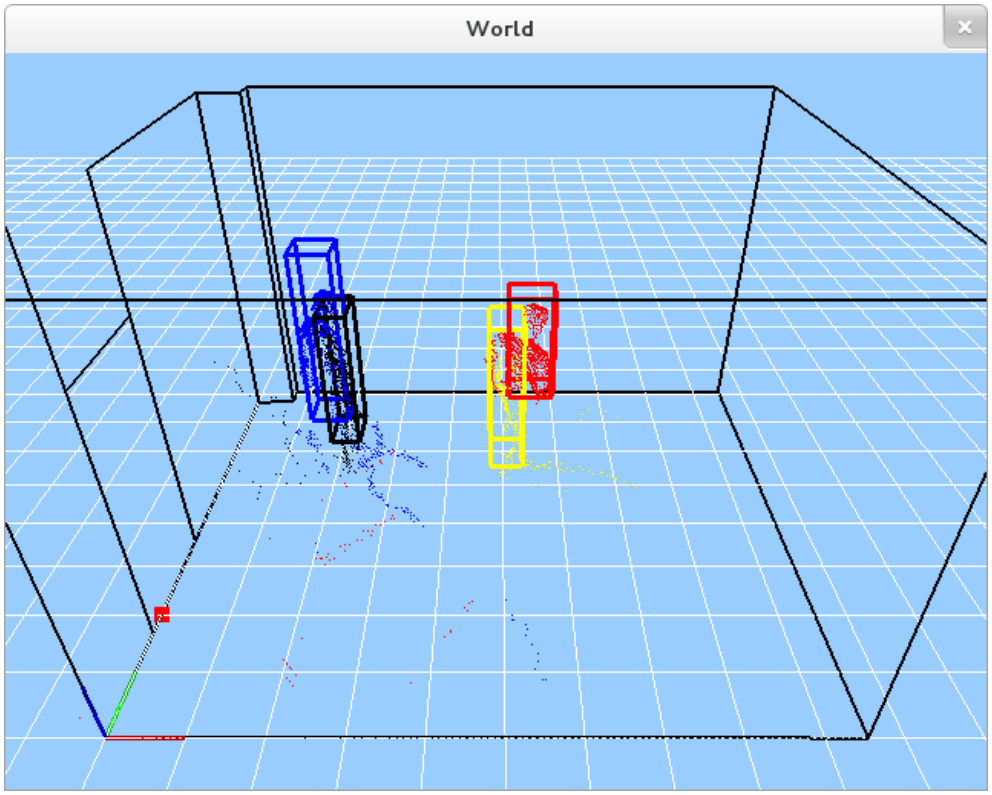


Figura 5.11: Resultado del filtrado con histogramas

5.4. Detección de observaciones no válidas

Una vez filtradas las observaciones independientes es necesario, antes de del proceso de fusionado, determinar si alguna de ellas no son válidas. Algunas premisas añadidas tienen que ver con el tamaño de los prismas:

- Altura máxima de 2,5 metros.
- Volumen máximo de 1 metro cúbico.

Además, tras las numerosas pruebas realizadas de la detección de personas por parte de la librería NITE, se observaron problemas cuando el sensor se colocaba de forma oblicua a alguna pared. En muchos de los casos, si alguien pasa por delante de la pared, ésta es tomada como una persona (figura 5.12).

Para solucionar este tipo de falsos positivos se ha utilizado una detección de planos implementada en la librería PCL. Esta detección, está basada en el algoritmo de RANSAC, utilizando el modelo de un plano, y devuelve la probabilidad de que la observación sea un plano. Las premisas que se deben cumplir para que la observación sea tomada como un plano es que los puntos no sean muy distantes (si la observación es realmente un plano, los puntos serán prácticamente equidistantes entre sí), y que al menos el 80 % estén contenidos en el modelo con mayor puntuación de RANSAC. En

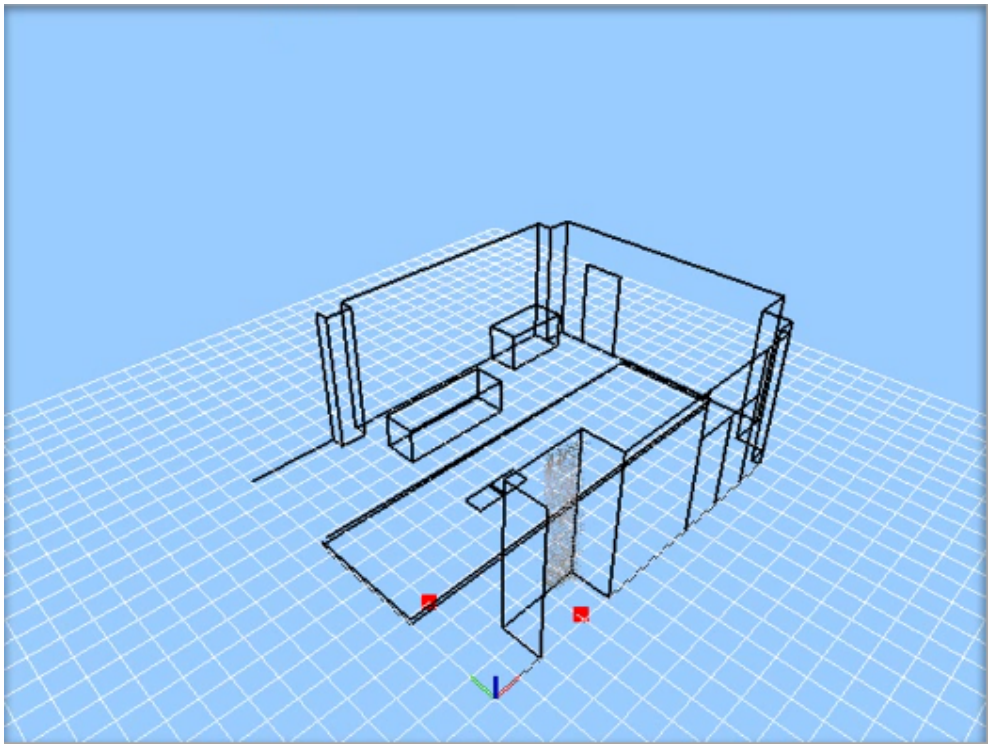


Figura 5.12: Falso positivo en la detección de personas de NITE

caso de que esta detección sea positiva se descarta el prisma y no es tomando como una persona.

5.5. Fusión de observaciones

Una vez filtradas todas las observaciones y descartadas aquellas que no son válidas, es necesario aplicar un proceso para combinar aquellas observaciones que sean susceptibles de pertenecer a la misma persona pero que han sido captadas desde dispositivos distintos.

Dos nubes individuales captadas desde dos dispositivos distintos serán fusionadas cuando el centro de masas de los prismas obtenidos, tras el proceso de filtrado, sea menor que una cierta distancia. El centro de masas es calculado como el punto 3D resultante de promediar todos los puntos de la nube individual una vez agrupada en un prisma. Esta distancia es uno de los parámetros de configuración del sistema y debe mantener un compromiso entre la combinación de observaciones que realmente son de una misma persona, y que mantenga divididas personas que son distintas pero están muy juntas. Es un parámetro que depende en gran medida de la calidad de la calibración de las cámaras. Si la calibración del sistema es buena, el umbral de combinación puede ser bajo y discernir muy bien si las combinaciones se producen por un hecho o por otro. De hecho, si la calibración fuese perfecta, la nube de puntos debería solaparse. Sin embargo, si el error de calibración es alto, la distancia de observaciones tomadas desde dos sensores puede ser incluso mayor que la distancia entre dos personas, por

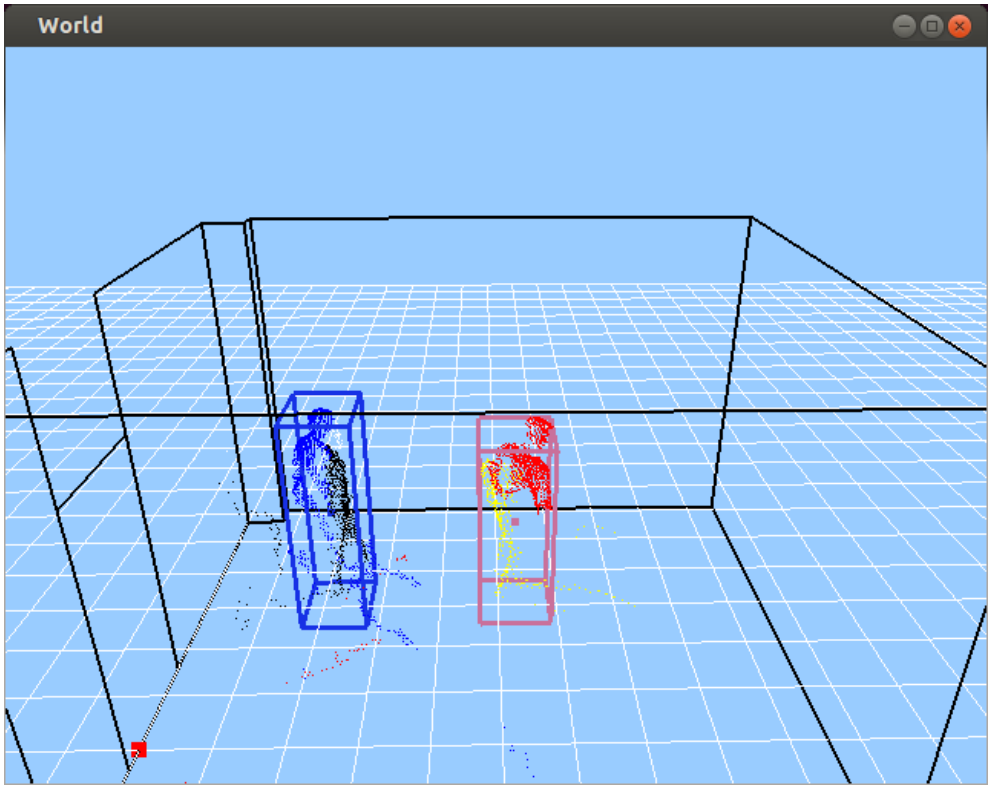


Figura 5.13: Resultado de combinar observaciones coincidentes.

ello este parámetro debe ser fijado una vez se haya instalado el sistema y se haya comprobado la calidad del proceso de calibración.

El resultado obtenido después de aplicar este proceso es una único prisma para cada una de las personas detectadas, y dicho prisma engloba las dos zonas densas de las observaciones iniciales (figura 5.13).

5.6. Módulo de seguimiento

Una vez explicado el proceso por el cual se obtienen las observaciones que van a servir de entrada al sistema de seguimiento, detallaremos los diferentes motores de seguimiento implementados. El primero de ellos, es un motor simple basado en las distancias de las observaciones de una iteración a otra y el segundo de ellos, más complejo, basado en el filtro de Kalman.

5.6.1. Por distancias

El módulo de seguimiento desarrollado asocia observaciones de *t* con las observaciones de *t-1*, en función de la distancia entre ellas. Siguiendo este paradigma, una observación en un instante es asociada con otra observación capturada en el instante anterior si la distancia entre estas observaciones es inferior a un cierto umbral.

Este es un motor muy sencillo de implementar y funciona relativamente bien cuando el seguimiento se realiza sobre pocas personas y éstas están separadas entre sí. El problema de utilizar este motor de seguimiento es la dificultad de fijar el umbral de asociación. Este valor fija la relación entre la distancia máxima que una persona se puede desplazar de un instante al siguiente, y la distancia mínima a la que pueden estar dos personas. Si el umbral de distancia es mayor a la distancia entre dos personas detectadas, la asociación entre estas observaciones y las observaciones del instante anterior serán totalmente aleatorias. Sin embargo, si este umbral es demasiado pequeño, dos observaciones de una misma persona en dos instantes diferentes distan mucho entre sí, porque la persona se esté movimiento rápidamente, serán tomadas como dos personas distintas.

5.6.2. Filtro de Kalman

Como el motor por distancias no es lo suficientemente robusto para aplicarlo en el seguimiento tridimensional de personas, se estudió la aplicabilidad de diferentes técnicas de seguimiento. Una de las más utilizadas en sistemas ruidosos es el filtrado de Kalman [Kalman, 1960]. La técnica de Kalman permite la estimación de una variable de la cual se disponen una serie 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 aplicando las observaciones del estado actual.

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 la siguiente ecuaciones:

Ecuación del proceso a modelar:

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

Ecuación de relación entre la medida a estudiar y la medida a observar:

$$Z_k = H x_k + V_k$$

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 $\mathbf{x}$ corresponde con la posición $(\mathbf{x_x}, \mathbf{x_y}, \mathbf{x_z})$ y la configuración dinámica del estado expresada como un vector de velocidades de la persona en los tres planos $(\mathbf{v_x}, \mathbf{v_y}, \mathbf{v_z})$. Este proceso será alimentado con las observaciones proporcionadas por el sistema que será el centro del prisma expresado como $(\mathbf{z_x}, \mathbf{z_y}, \mathbf{z_z})$. En el modelado del prisma presentado anteriormente se tienen en cuenta otras 3 dimensiones que corresponden al volumen del prisma pero que no han sido incluidas para las estimaciones de Kalman con el fin de simplificar los cálculos. De esta forma, cuando se utilice el seguimiento basado en Kalman, la posición (x,y,z)

será estimado por el filtro mientras que el tamaño del prisma será tomado directamente de la observación de entrada.

$$\begin{cases} X_{x_{k+1}} = X_{x_k} + V_x dt \\ X_{y_{k+1}} = X_{y_k} + V_y dt \\ X_{z_{k+1}} = X_{z_k} + V_z dt \\ V_{x_{k+1}} = V_{x_k} \\ V_{y_{k+1}} = V_{y_k} \\ V_{z_{k+1}} = V_{z_k} \end{cases} \qquad A = \begin{pmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

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:

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

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 medida 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 observacional. Este parámetro determina la proporción del error que será utilizado para actualizar 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 5.14.

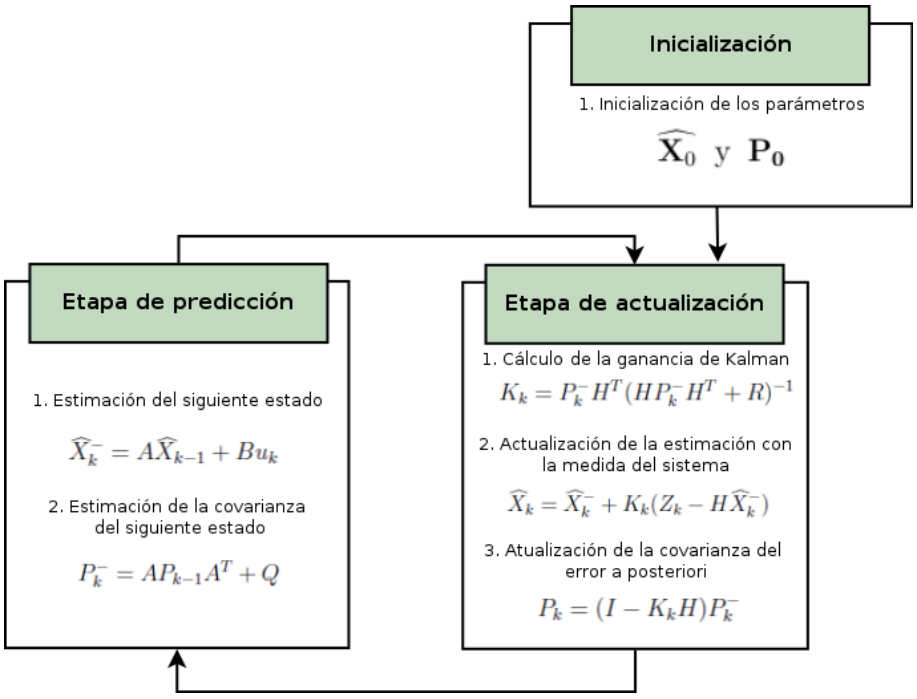


Figura 5.14: Diagrama del filtrado de Kalman.

En este trabajo se ha utilizado el filtro de Kalman para estimar la zona de interés donde se va a buscar una persona detectada en el instante t en siguiente instante de tiempo (t+1). La búsqueda local se realiza por distancia al igual que en el seguimiento por distancias. Lo que aporta Kalman es reducir la zona de búsqueda local, minimizando las variaciones de personas cuando éstas están juntas y conseguir un algoritmo de seguimiento mucho más estable y robusto.

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

5.7. Módulo de alarmas y situaciones de riesgo

Una vez explicados cada uno de los pasos del sistema de seguimiento 3D de personas desarrollado en este trabajo, se detallarán cómo son detectadas las situaciones de riesgo y las diferentes situaciones que es capaz de detectar. Para comprobar el estado del sistema y determinar si se ha producido alguna situación de riesgo se ha desarrollado un módulo de alarmas propio que toma como entrada la salida del módulo de seguimiento, es decir, la posición 3D de cada una de las personas del entorno junto con la nube de puntos original.

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 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 en 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 de riesgo.

- **Estancia prolongada en el baño**
  Análogo a la situación de riesgo anterior, el sistema es capaz de detectar si alguien ha entrado al baño y si no sale de él en un tiempo prudencial, lanzará 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.

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.

5.8. Módulo gráfico

El sistema desarrollado dispone de una interfaz gráfica donde se vuelca toda la información generada (figura 5.15). 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:

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

Este módulo se ejecuta como una hebra independiente y es completamente opcional, no es necesario levantar el interfaz para que el algoritmo detecte las diversas situaciones de riesgo. Toda la información que es representada se recoge del módulo de adquisición estando protegida frente a accesos concurrentes detrás de un *mutex*.

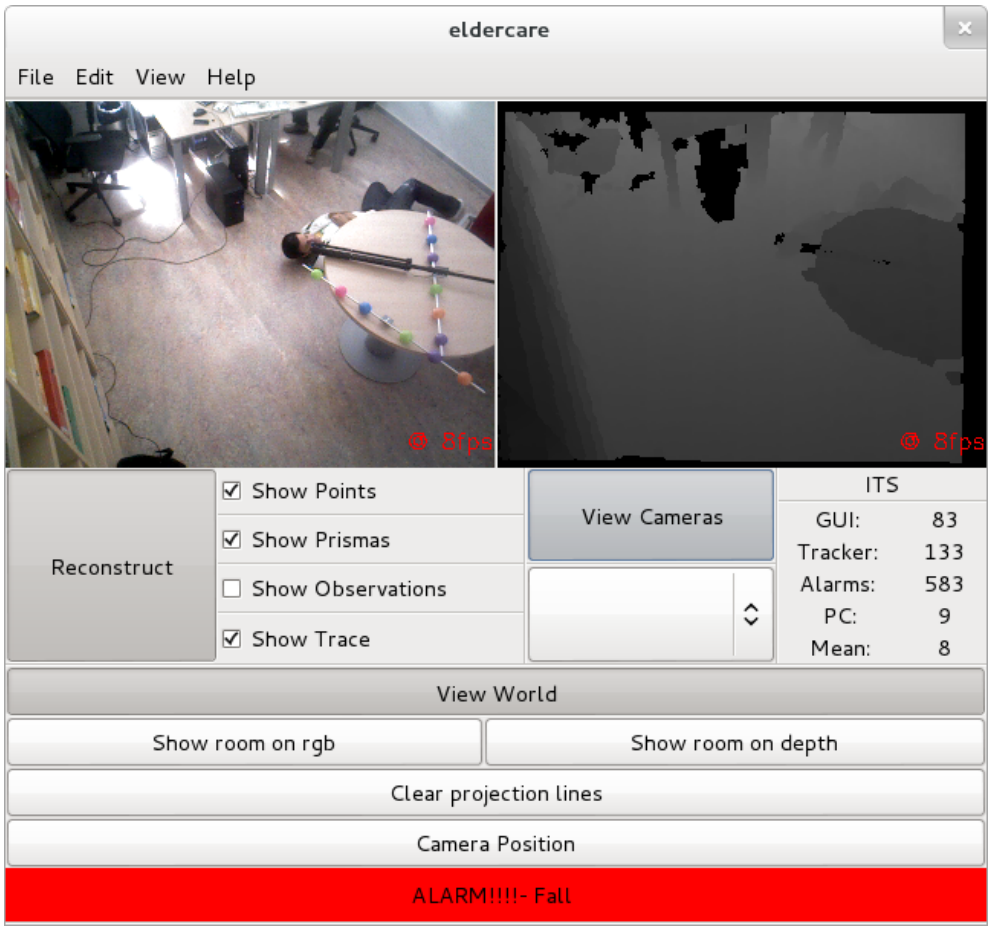


Figura 5.15: Interfaz gráfico de ElderCare.

A través del propio visor, se permite la conexión a los diferentes dispositivos instalados en el sistema y ver directamente tanto la 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.

5.9. 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 deben estar fijados los siguientes parámetros:

- Tipo de filtrado.
  Cada filtrado tiene una serie de parámetros propios, para RANSAC:
  - El número de iteraciones de RANSAC.
  - El número de puntos utilizados por RANSAC para el cálculo de las soluciones parciales.

  Para el filtrado por histogramas es necesario fijar:
  - El tamaño del entorno monitorizado.
  - El número de bins para los histogramas.
- Distancia máxima entre dos observaciones de distintos sensores para ser combinadas
- Motor de seguimiento utilizado
  En función del motor de seguimiento utilizado, es necesario fijar una serie de parámetros propios, para el motor basado en distancias es simplemente la distancia máxima entre dos observaciones de dos instantes de tiempo diferentes, mientras que para el seguimiento basado en Kalman son:
  - Distancia máxima entre la predicción de Kalman y la observación en el instante actual.
  - Tiempo máximo que se utilizarán las predicciones de Kalman sin recibir alguna observación (persistencia temporal).
  - Tiempo mínimo que debe estar una persona detectada por el sistema antes de utilizar predicciones.

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, 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 prima, junto con la franja horaria que debe estar activa.

5.10. Acceso remoto al sistema

Es necesario poder acceder al sistema de forma remota ya que, en un entorno teleasistencial, estará instalado en la propia casa del sujeto monitorizado. Es necesario el acceso al sistema en dos modos principalmente. Primeramente, 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 5.16). Este componte a su vez, muestra en el propio visor un histórico con las últimas alamas 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. 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 dado soporte al componente *remoteConfigurator* el cual permite transmitir por completo el fichero de configuración del sistema pudiéndolo cargarlo en tiempo de ejecución por la aplicación sin necesidad de parar el sistema. La descripción de este componente así como su funcionamiento ha sido descrito en el apartado 4.4.

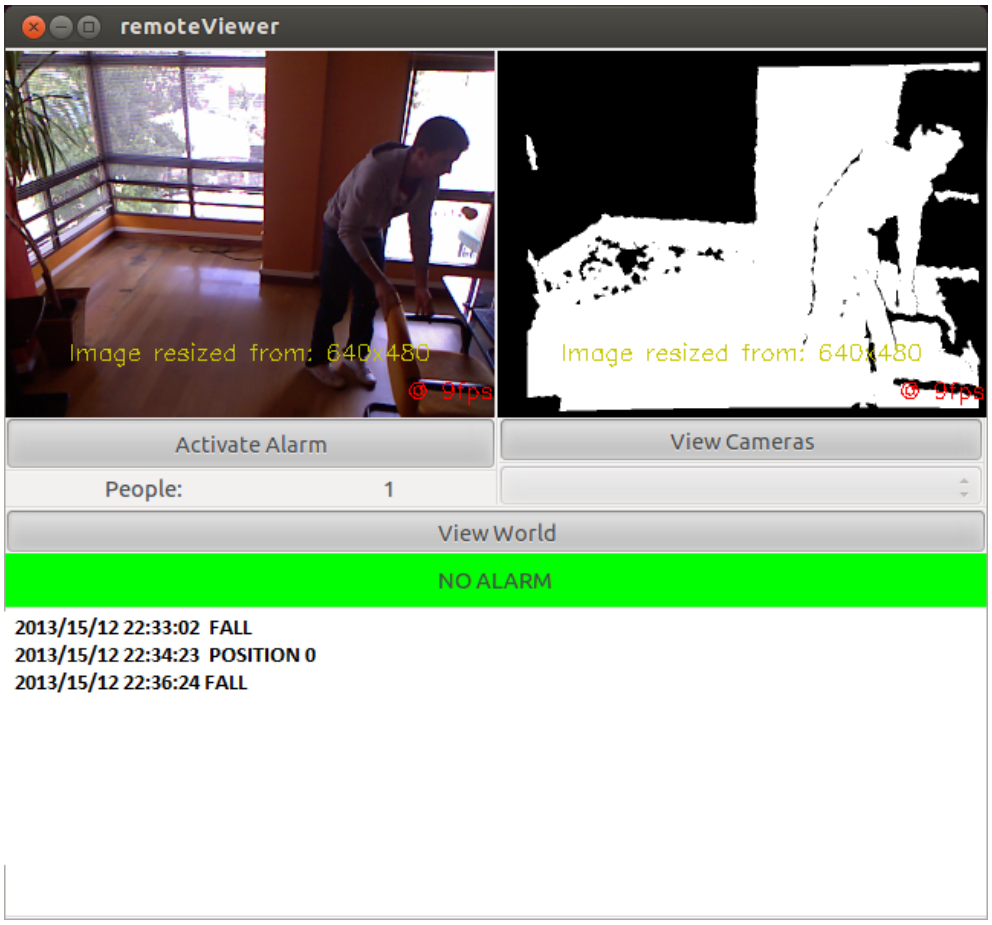


Figura 5.16: RemoteViewer.

Capítulo 6
Experimentación y resultados

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

6.1. Comparativa de los diferentes algoritmos de agrupamiento

El algoritmo de seguimiento tridimensional de personas planteado en este trabajo tiene la posibilidad de funcionar con dos motores de seguimiento: el basado en distancias y el basado en el filtro de Kalman. A su vez, las nubes de puntos que reciben cada uno de estos motores pueden ser agrupadas utilizando dos métodos, uno basado en RANSAC y otro propio basado en histogramas.

Desde el punto de vista del rendimiento computacional, la agrupación basada en histogramas es más eficiente, sólo es necesario recorrer la nube de puntos una vez para hacer el cálculo de los histogramas tanto convencional como de acumulación. Sin embargo, con RANSAC es necesario recorrer la nube de puntos muchas más veces, una vez por cada evaluación de los resultados parciales. En la tabla 6.1 se pueden apreciar los resultados obtenidos de calcular la media y la desviación estándar de 3000 ejecuciones, en ella podemos ver que, la ejecución de la agrupación por histogramas se realiza 10 veces más rápida que por RANSAC. A su vez, la agrupación por histogramas tiene la desventaja de ser muy parametrizable y necesitar fijar una serie de valores de configuración que son necesarios fijar de la forma correcta en función del entorno que se esté monitorizando.

Tipo de filtro	**Promedio (μs)**	**Desviación típica (μs)**
RANSAC	4217	1500
Histogramas	47	26

Tabla 6.1: Rendimiento de los agrupamientos

A pesar de ser la agrupación por histogramas más eficiente computacionalmente, el filtrado por RANSAC sigue siendo muy ligero, necesitando alrededor de 4 milisegundos para ejecutarse. Debido a que no necesita ningún tipo de parametrización para seguir

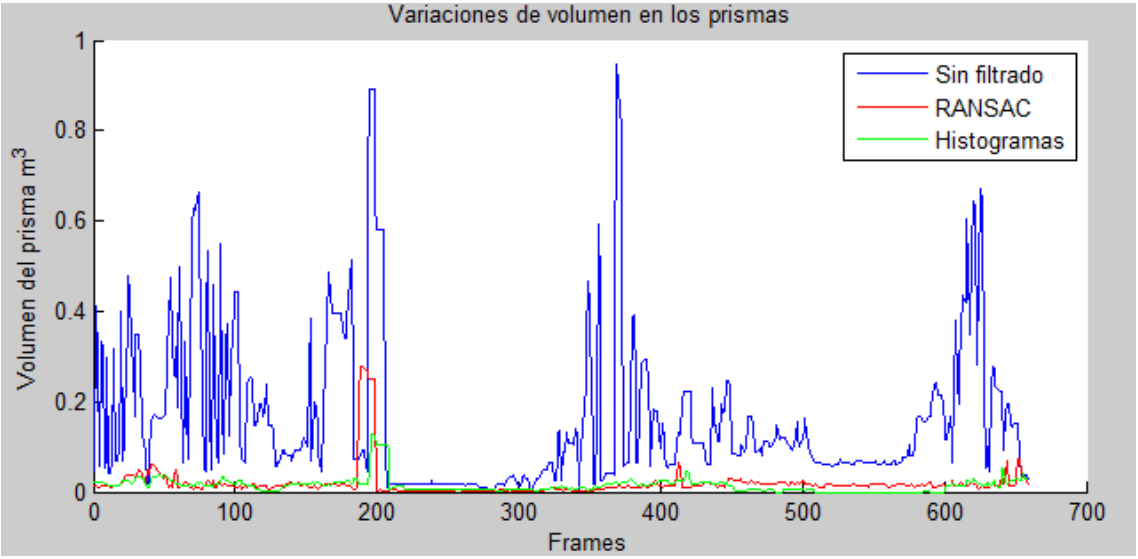


Figura 6.1: Volumen del prisma de una persona con las diferentes técnicas de agrupamiento

funcionando en cualquier entorno de forma correcta, éste ha sido el filtrado utilizado para las pruebas finales del sistema.

Para comprobar el resultado real de cada una de las técnicas implementadas, se han guardado los datos de un sensor funcionando a 10Hz monitorizando una zona en la que se encontraba una persona andando libremente y simulando dos caídas, con una duración aproximada de 60 segundos. Con el fin de evaluar los dos tipos de agrupamiento, se realizó un experimento para comprobar el funcionamiento y evolución temporal de cada filtro. En la figura 6.1, se pueden ver las variaciones de volumen del prismas que realiza el seguimiento de la persona monitorizada con los datos de prueba. En ella, se puede comprobar cómo el volumen del prisma es muy variable al ser calculado con los datos sin filtrar, debido a que es muy sensible al ruido del sensor y a las imprecisiones en la segmentación de personas de NITE. De la misma forma, se puede apreciar cómo estos valores se estabilizan utilizando tanto el agrupamiento por RANSAC como el agrupamiento por Histogramas.

En un segundo experimento realizado, se estudió la influencia de las variaciones de volumen detectadas en el experimento anterior en los datos finales de posición 3D de personas. En la figura 6.2, se pueden ver las posiciones desglosadas en los planos X, Y y Z. En estos resultados, se refleja la estabilización que se produce tanto en el plano X como en el plano Y, independientemente del agrupamiento utilizado, pero a su vez, se puede apreciar que el proceso de agrupación introduce cierta inestabilidad en el plano Z. Esto se debe a la propia naturaleza de las técnicas de agrupamiento utilizadas, ya que toma como premisa la búsqueda de zonas donde la nube de puntos sea densa. Estas fluctuaciones se deben a que, en numerosas ocasiones, tanto la cabeza como los pies no son detectados de forma precisa siendo representadas con muy pocos puntos, los cuales son descartados por RANSAC y por los histogramas. La única repercusión directa de

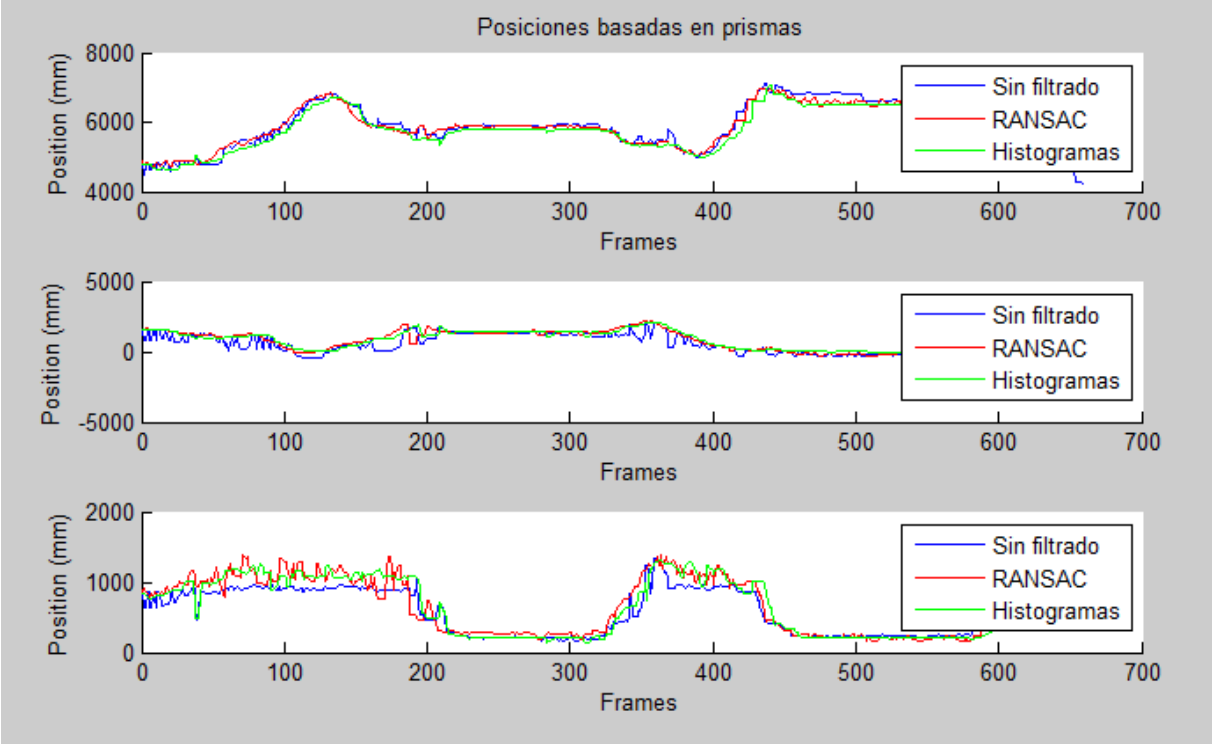


Figura 6.2: Posición 3D de una persona con las diferentes técnicas de agrupamiento

este hecho es que el algoritmo presentado no es capaz de detectar con fiabilidad la altura de las personas, sin embargo, no afecta al seguimiento ni a la detección de las diferentes situaciones de riesgo presentadas ya que se consigue gran precisión en los planos X e Y.

6.1.1. Comparativa de los algoritmos de seguimiento

El siguiente experimento trata de comprobar las diferencias de las trayectorias 3D estimadas de una personas en el mismo recorrido utilizando el motor de seguimiento por distancias y el basado en el filtro de Kalman. Para ello se realizó una ejecución simultánea de ambos motores de seguimiento sobre los datos de prueba y los resultados obtenidos se pueden observar en la figura 6.3. Como era previsible, el filtro de Kalman genera una trayectoria mucho más suavizada y estable que el emparejamiento por distancias.

Para complementar la información generada con el experimento anterior, se han realizado unas pruebas que tienen que ver con la estabilidad del algoritmo de seguimiento. Según los resultados anteriores, el seguimiento basado en el filtro de Kalman produce unas trayectorias mucho más suaves y estables que el seguimiento con distancias. Sin embargo, es necesario comprobar la fiabilidad del seguimiento desde el punto de vista de la estabilidad. Un buen algoritmo de seguimiento 3D debe ser capaz de seguir a cada una de las personas de forma independiente y realizar un buen emparejamiento entre las nuevas observaciones y el estado anterior del sistema. En este

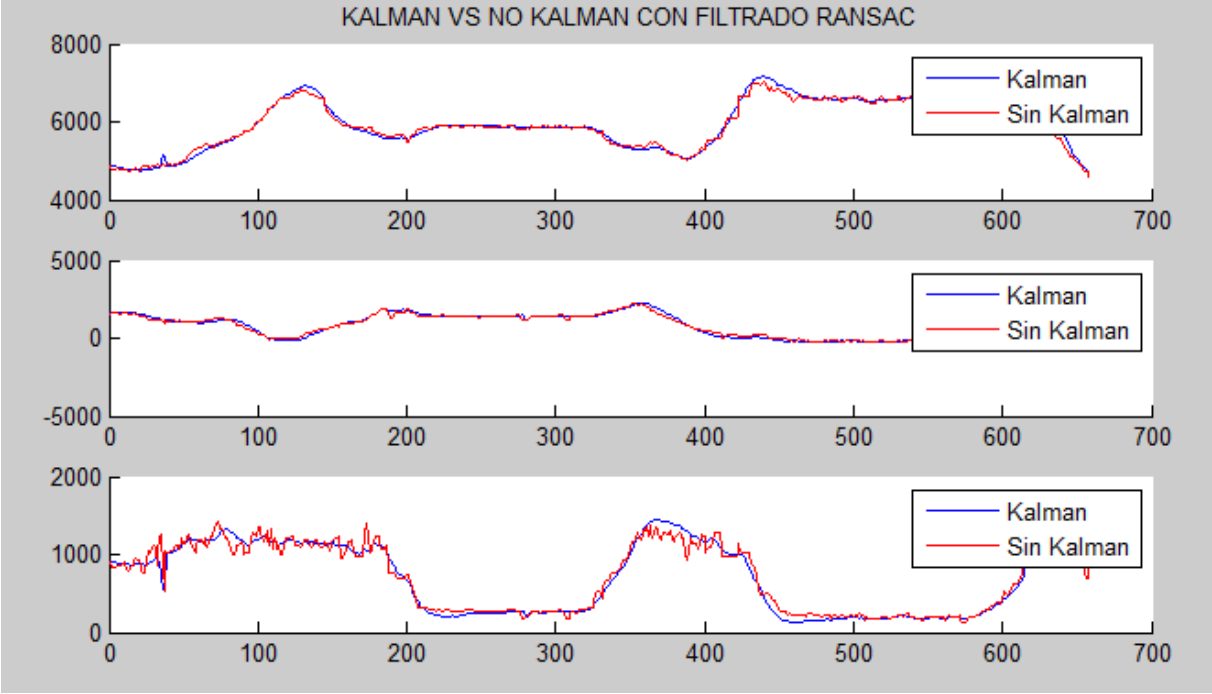


Figura 6.3: Comparación del motor de seguimiento por distancias con el agrupamiento de Kalman

experimento se han comparado, sobre los mismos datos de prueba, el número rupturas en el seguimiento. Es decir, cuántas veces el algoritmo no es capaz de hacer un buen emparejamiento y para resolverlo tiene que descartar los datos anteriores y generar una nueva persona. En la figura 6.4, se ha representado cada una de estas rupturas con una línea vertical. Las búsquedas locales, tanto en el algoritmo basado en distancias como en el filtro de Kalman, se han realizado sobre la misma distancia, en un radio de 30 centímetros. Como se puede observar, los mejores resultados han sido obtenidos con el filtro de Kalman, siendo capaz de realizar el seguimiento de la persona durante toda la grabación sin ninguna variación. Este exitoso resultado se debe a que Kalman utiliza un modelo de predicción basado en trayectoria, cada una de las predicciones del estado siguiente avanza generalmente hacia la posición donde se va a encontrar la persona, permitiendo que la búsqueda local con una distancia pequeña dé buenos resultados.

Por último, se ha realizado una evaluación del peso computacional de cada una de las posibles configuraciones, comprobando el tiempo que tarda en ejecutar cada una de ellas. Dichos cálculos, presentados en la tabla 6.2, se han calculado sobre los mismo datos de los experimentos anteriores. Como se puede comprobar, en todos los casos se cumple con los requisitos de sistemas de tiempo real.

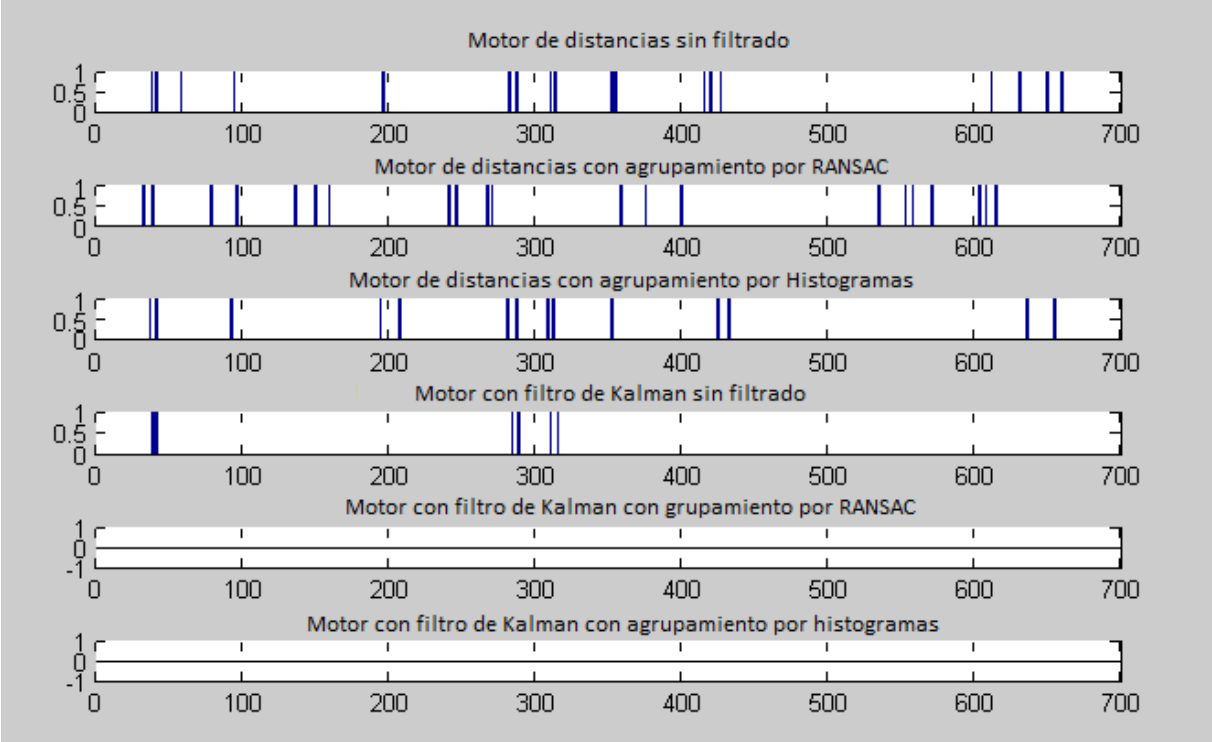


Figura 6.4: Rupturas de seguimiento de personas en las diferentes configuraciones

Controlador	Promedio (ms)	Desviación típica (ms)
Sin filtrado sin kalman	5.3	4.5
Filtrado RANSAC sin kalman	15.4	13.7
Filtrado por histogramas sin kalman	6.8	5.3
Sin filtrado con kalman	5.3	4.5
Filtrado RANSAC con kalman	15.8	13.2
Filtrado por histogramas con kalman	5.7	4.8

Tabla 6.2: Rendimiento de las diferentes configuraciones del motor de seguimiento

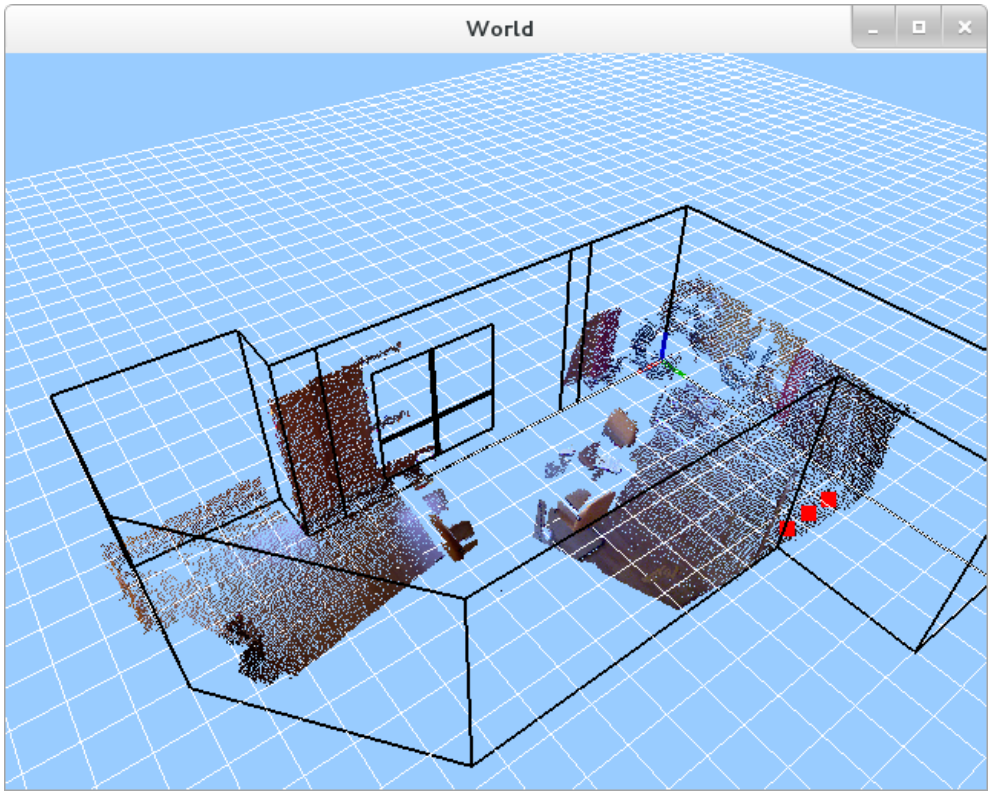


Figura 6.5: Entorno de pruebas 1.

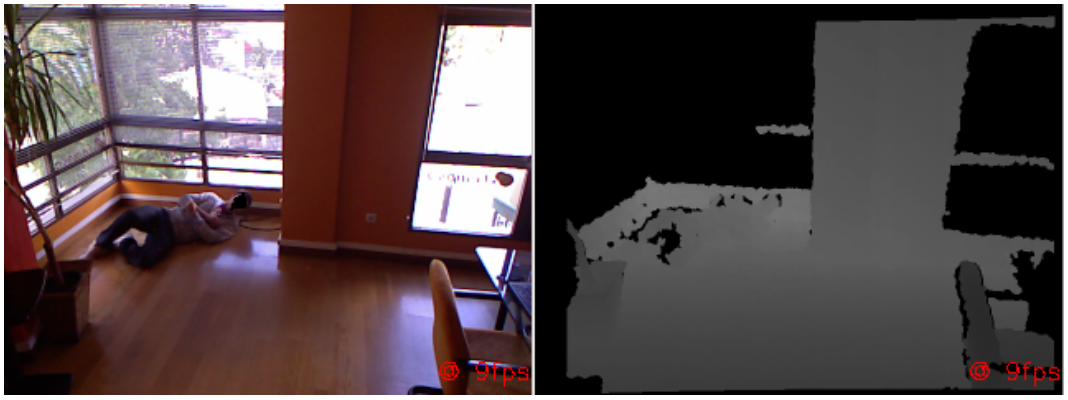
6.2. Validación experimental del sistema

Con el fin de validar el algoritmo de seguimiento desarrollado, así como el módulo de alarmas, se ha comprobado el funcionamiento final del sistema en dos entornos reales.

El primero de ellos, es una sala de reuniones (figura 6.5) monitorizada con dos sensores donde las situaciones de riego definidas son las caídas y las salidas de la estancia utilizando la alarma por posición, fijando la puerta como zona prohibida. Para ello se ha grabado una secuencia de aproximadamente 69 segundos donde se monitoriza a un único sujeto que genera 3 situaciones distintas de caídas: una caída donde el sujeto no es captado de forma completa (figura 6.6(a)), una segunda situación donde el sujeto sufre una caída en una zona con mucho ruido debido a la incidencia directa del sol (figura 6.6(b)) y una tercera en una zona donde no afectan las condiciones solares (figura 6.6(c)). Además de las caídas, se genera una cuarta alarma con una situación de abandono de la estancia (figura 6.6(d)).

Se contrastaron los resultados obtenidos con un total de 2000 repeticiones, obteniendo los resultados expuestos en las tabla 6.3, con una tasa de aciertos del 100 %. En toda la evaluación solamente se ha generado un único caso de falso positivo en caías.

(a)



(b)

(c)

(d)

Figura 6.6: (a) Situación de caída 1 (b) Situación de caída 2(c) Situación de caída 3 (d) Situación de salida

Situación	Aciertos	Falsos negativos
Situación 1	2000	0
Situación 2	2000	0
Situación 3	2000	0
Situación 4	2000	0

Tabla 6.3: Resultados de la detección en las diferentes situaciones de riego

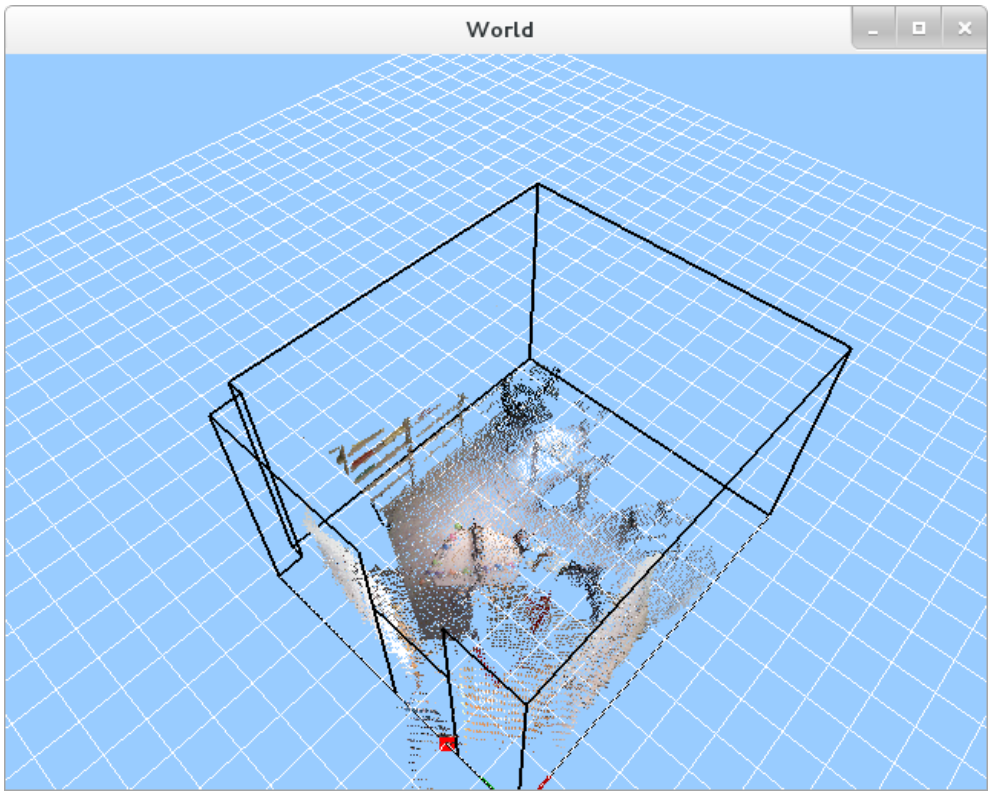


Figura 6.7: Entorno de pruebas 2.

El segundo entorno donde se instaló el sistema es en un despacho del departamental III del campus de Fuelabrada de la Universidad Rey Juan Carlos. En este caso, el entorno también estaba monitorizado con 2 cámaras pero esta vez, se encontraban allí situados dos sujetos que generaban diferentes situaciones de riego (figura 6.7). La primera situación es la salida de una de las personas siendo ocluido en gran medida por la segunda persona. La segunda situación se produce simultáneamente con la tercera, una de las personas abandona la estancia cuando la otra ha sufrido una caída y se encuentra situado tendido en el suelo (figuras 6.8(b) y 6.8(c)).

En este caso, los resultados obtenidos tras evaluar el entorno planteado 2.000 veces están detallados en la tabla 6.4, obteniendo una tasa de aciertos del 100 %. En cuanto a falsos positivos, sólo se han generado dos alarmas que no correspondían a ninguna de las situaciones reales, correpondiendo a 2 falsos positivos en salidas.

Todos los resultados presentados en este capítulo se han realizado utilizando el

(a)

(b)

(c)

Figura 6.8: (a) Situación de salida 1 (b) Situación de salida 2 (c) Situación de caída 1

Situación	Verdaderos Positivos	Falsos negativos
Situación 1	2000	0
Situación 2	2000	0
Situación 3	2000	0

Tabla 6.4: Resultados de la detección en las diferentes situaciones de riesgo

módulo de agrupamiento basado en RANSAC y el módulo de seguimiento con la configuración del filtro de KALMAN. Se ha utilizado el sistema operativo debian stable x64 sobre un ordenador con un procesador i7 quad-core a 2.7 GHz con 4GB de RAM. Así mismo, los datos utilizados para realizar las evaluaciones han sido siempre exactamente los mismos gracias a que *JdeRobot* incorpora herramientas de reproducción enlatada de logs, permitiendo grabar una serie de datos en disco para más tarde reproducirlos como si se tratase del propio sensor.

Capítulo 7
Conclusiones

En los capítulos anteriores se ha realizado una amplia descripción del problema abordado y de las soluciones que se han desarrollado, así como de los resultados obtenidos en los diversos experimentos realizados. En este capítulo se presentarán las conclusiones principales del trabajo y se propondrán una serie de líneas futuras de investigación y desarrollo que se pueden continuar a partir de este trabajo fin de grado.

7.1. Conclusiones

Tras un desarrollo de cerca de 24.500 líneas de código, divididos entre los diferentes componentes realizados, el driver del sensor y la aplicación final de ElderCare, se ha conseguido alcanzar cada uno de los objetivos planteados en el capítulo 2. Con todo ello, se ha conseguido desarrollar un algoritmo de seguimiento tridimensional de personas e integrarlo dentro de un sistema de teleasistencia, capaz de detectar una serie de situaciones de riesgo de forma eficiente y robusta.

A continuación, se repasarán los distintos subobjetivos planteados para recapitular la solución a cada uno de ellos:

1. El primer subobjetivo a alcanzar tenía que ver con la creación de un servidor que permitiese acceder a los sensores tipo Kinect desde *JdeRobot*. Tal y como se ha explicado en el apartado 4.1, se ha desarrollado un componente denominado *OpenNIServer* que incluye toda la funcionalidad necesaria para servir los datos ofrecidos por el sensor. Para validar el funcionamiento de este servidor, se ha desarrollado un visor capaz de mostrar todos los datos proporcionados, denominado *remoteViewer* y detallado en el capítulo 4.2.

   Una funcionalidad vital para este proyecto es la detección de personas. El servidor *OpenNIServer* incorpora la detección de personas de NITE por lo que es capaz de ofrecer los datos, tanto imágenes como nube de puntos, determinando si el dato en cuestión pertenece a una persona u otra o ninguna.

2. El segundo subobjetivo consistía en diseñar e implementar un algoritmo de estimación 3D de personas. El sistema desarrollado está detallado paso a paso en el capítulo 5, en él se describe todo el proceso de agrupación de puntos, la fusión de información cuando existen datos redundantes, el rechazo de observaciones no

válidas y los diferentes motores de seguimiento implementados.

3. El tercer subobjetivo fija la necesidad de detectar una serie de situaciones de riesgo en función de los datos obtenidos por el algoritmo de seguimiento. Para ello, se desarrolló un módulo de alarmas, detallado en el capítulo 5.7, capaz de detectar caídas, proximidad a zonas peligrosas, entrada y salida de estancias, ausencia de paso, estancia prolongada en el baño o intrusiones en zonas volumétricas prohibidas.

4. En el cuarto subobjetivo, se plantea el desarrollo de las aplicaciones que permitan tanto el acceso a los parámetros de configuración del sistema como a la visualización del estado y los datos de cada uno de los sensores. Para ello, se ha desarrollado el componente *remoteConfigurator*, que da soporte a la parte de configuración remota y el componente *remoteViewer* que permite la conexión remota al nodo principal del sistema y representar visualmente el estado de la aplicación junto a los datos de cada sensor. Todo ello se detalla en el apartado 5.10.

5. En el último subobjetivo se plantea la necesidad de realizar una validación experimental del sistema desarrollado en entornos reales (capítulo 7). Para ello, se han realizado dos validaciones independientes, una de ellas en una oficina de una empresa y la segunda en un despacho de la propia universidad. En ambos experimentos los resultados fueron satisfactorios, obteniendo una tasa de aciertos del 100 % con únicamente 2 falsos positivos en toda la evaluación.

Requisitos

Los requisitos que debía cumplir el sistema desarrollado en este trabajo se especifican 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 del sistema es *JdeRobot*, escrito por completo utilizando el lenguaje C++. El hecho de haber utilizado esta plataforma para el desarrollo ha permitido que sea directamente posible su implementación, de forma distribuida, y utilizando la red como vía de comunicación entre componentes.

Los componentes desarrollados para *JdeRobot* con el fin de dar soporte a los dispositivos utilizados en este proyecto han sido *OpenNIServer*, *RgbdViewer*, *rgbdManualCalibrator* y *remoteConfigurator*. Sobre este soporte, se ha desarrollado el algoritmo de seguimiento de personas en 3D así como el detector de situaciones de riego, todo ello encapsulado en la aplicación *ElderCare*.

El sistema desarrollado es robusto y con una alta fiabilidad en la detección de las diferentes situaciones de riesgo. También es fácilmente integrable en cualquier sistema

ya que utiliza interfaces Ice para ofrecer tanto la información de salida del algoritmo de seguimiento como la información sobre las diferentes alarmas generadas.

Descripción software

Este trabajo consta de varios componentes que han sido detallados a lo largo de la presente memoria:

1. *OpenNIServer*: que con 3788 líneas de código ofrece un acceso total a los componentes de los dispositivos utilizados, así como funcionalidad adicional para conversión de los datos directamente a nube de puntos y la funcionalidad de la detección de personas.

2. *RgbdViewer*(2627 líneas): que ofrece un visor con diversa funcionalidad extra para volcar en un interfaz GTK muy intuitivo toda la información ofrecida por *OpenNIServer*.

3. *RgbdViewerCalibrator*(3096 líneas): ofrece diferentes herramientas para la calibración manual de varios sensores en un mismo entorno.

4. *remoteViewer* y *remoteConfigurator* con 4302 y 939 líneas de código respectivamente ofrecen diversa funcionalidad para el acceso y configuración remotas.

5. *ElderCare*: con 9753 líneas de código incorpora todas los algoritmos del seguimiento de personas 3D y la detección de las diferentes situaciones de riesgo.

Aportes en este trabajo

Las novedades presentadas en este trabajo son dos. Por una parte, un soporte completo para los sensores RGBD al entorno *JdeRobot*: un servidor para acceder a los datos, un cliente para visualizar la información y una herramienta para calibrar el sensores en diferentes entornos. Con estos tres componentes se abren las puertas al desarrollo de nuevas aplicaciones en muy diferentes ámbitos donde se puedan utilizar este tipo de sensores, como la navegación con robots o la creación de mapas 3D.

La segunda y principal novedad de este proyecto, es la aplicación de los sensores RGB-D para implementar un sistema de teleasistencia basado en seguimiento 3D de personas, capaz de detectar una serie de situaciones de riesgo vitales en entornos donde se desenvuelven personas con algún tipo de dependencia. Esto contrasta con las versiones anteriores, que utilizaban varias cámaras convencionales para esta labor.

Este trabajo, ha creado un sistema de teleasistencia totalmente autónomo en el que las personas monitorizadas no necesitan llevar ningún atuendo ni dispositivo especial, solamente sobre sensores RGB-D. Esto permite reducir el número de sensores empleados, pudiendo incluso utilizar un sólo sensor para monitorizar una sala de

pequeño tamaño. La información tomada como entrada del algoritmo de seguimiento es únicamente los datos espaciales 3D ofrecidos por el sensor permitiendo, al no utilizar información adicional de color, que el sistema funcione incluso en total oscuridad, siendo mucho más robusto y estable.

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 de 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 remotos. De igual forma, se han reforzado los conocimientos previos en el uso de GTK para el desarrollo de interfaces de usuario y OpenGL para la visualización de escenas tridimensionales.

En lo referente al seguimiento en 3D, se han adquirido conocimientos sobre el tratamiento de información en 3 dimensiones, conversiones geométricas tridimensionales, filtrado de datos obtenidos con sensores ruidosos a partir de primitivas de modelado utilizando RANSAC y modelos estimación de movimiento a través del filtro de Kalman.

También se han adquirido valiosos conocimientos sobre la configuración de sistemas de forma remota utilizando ficheros con formato XML, así como los diferentes analizadores disponibles para trabajar con este lenguaje de documentos.

7.2. Trabajos futuros

Este trabajo fin de grado sienta la bases para futuros proyectos con sensores RGB-D bajo la plataforma *JdeRobot*. Utilizando los componentes desarrollados en este trabajo, que aportan un soporte completo a este tipo de sensores, se pueden desarrollar nuevas aplicaciones en cualquier ámbito de interés.

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 herramienta de calibración automática o semiautomática de sensores RGB-D que permita la estimación de los parámetros de posición y orientación del sensor de forma sencilla y en un tiempo asumible.

Una limitación de este trabajo, es la utilización de NITE como detector de personas. Aunque licenciado como Apache (v.2), es de código cerrado, por lo que cualquier cambio que se haga en el kernel del detector puede afectar al sistema planteado en este trabajo. Una línea de futuro sería trabajar en el desarrollo propio tanto de detección de personas

como de seguimiento.

Otra línea factible es el desarrollo de aplicaciones complementarias al sistema planteado en el presente trabajo. La primera funcionalidad a desarrollar sería una herramienta visual multidispositivo, por ejemplo mediante el uso de *html5*, que muestre en una página web los datos generados por el sistema. Esta puerta queda abierta por la propia arquitectura de *ElderCare* ya que es capaz de servir toda la información generada con interfaces Ice. Otro complemento interesante a desarrollar sería una herramienta visual con la que poder establecer y configurar las diferentes situaciones de riesgo, de forma sencilla e intuitiva, por usuarios 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.

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

[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.

[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, 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.

[Kalman, 1960] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[Khoshelham y Elberink,] 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.

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

[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.

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

[Oliver *et al.*, 2012] Ayrton Oliver, Steven Kang, Burkhard C. Wünsche, y Bruce MacDonald. Using the kinect as a navigation sensor for mobile robotics. In *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, IVCNZ '12, pages 509–514, New York, NY, USA, 2012. ACM.

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

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

[pri,] Primesense. `http://www.primesense.com`. 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.

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

[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, 2009.

[Viager, 2011] Mikkel Viager. Analysis of Kinect for mobile robots. Technical report, Technical report, Technical University of Denmark, 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.