INGENIERÍA TECNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso Académico 2009/2010

Proyecto de Fin de Carrera

Memoria visual 3D para un robot móvil con par estéreo

Autor: Pablo Miangolarra Tejada

Tutor: Prof. Dr. José María Cañas Plaza

For Rebecca,

you have made it possible

Agradecimientos

Deseo expresar mi más profundo agradecimiento a todos los miembros del Grupo de robótica de la Universidad Rey Juan Carlos por su apoyo y colaboración. En especial a mí tutor Jose María Cañas por toda el tiempo, dedicación y ayuda que me ha prestado durante el desarrollo de este trabajo.

Sin olvidarme de los compañeros de laboratorio, Julio, Eduardo, Gonzalo, Darío y Luis Miguel, por haberme ayudado en numerosas ocasiones a lo largo de estos meses.

Tambien me gustaría agradecer a todos mis compañeros de carrera, con los que he compartido sueños y objetivos, los momentos en común de los que hemos disfrutados a lo largo de estos años, a Alain, Daniel, Sergio, Carlos y Gonzalo.

Además, quiero dar las gracias a mis grandes amigos, por su comprensión y apoyo durante todo este tiempo, gracias a Sergio, Jenny, Alejandro, Sergio, Raúl, Isabel, Iván y Sergio, sin olvidar a otros muchos que también merecería ser nombrados.

Finalmente, quiero dar las gracias a mi familia por transmitirme valores como el esfuerzo y la dedicación que han hecho posible este trabajo.

gracias, a todos.

Resumen

La visión artificial aplicada al campo de la robótica es, hoy en día, una constante en el desarrollo de nuevos sistemas robóticos. El abaratamiento en los costes de fabricación de las cámaras ha hecho posible que los nuevos modelos incorporen a la visión como sensor principal. Las cámaras nos ofrecen una fuente potencialmente muy rica de información, pero desgraciadamente la extracción de esta información no es sencilla. La robótica más moderna hace un uso intensivo de las técnicas de visión artificial, con el objetivo de crear robots capaces de interpretar lo que ven por medio de sus cámaras de forma similar a como los hacemos los humanos.

En este proyecto se aborda un problema clásico de la visión artificial, la reconstrucción tridimensional. El sistema en el que se ha probado consta de un robot simulado dotado de un par de cámaras estéreo y del software que realiza la reconstrucción 3D en tiempo real, a partir de la información de las cámaras. Sobre esta representación del entorno, que forma una memoria visual 3D, el robot puede tomar decisiones de navegación.

Este trabajo propone una novedosa técnica para esa reconstrucción de la escena, el componente software desarrollado analiza las imágenes en busca de segmentos rectos en 2D, que posteriormente sitúa en el espacio tridimensional triangulando con la segunda cámara. Mediante el análisis de los segmentos 3D reconstruidos, se formulan también hipótesis perceptivas de objetos cotidianos. El resultado es un reconstrucción de un mundo de oficina con objetos del tipo puerta, pared o pasillo.

Para el desarrollo del proyecto se han utilizado los lenguajes de programación C y C++. La aplicación ha sido implementado en forma de componente de la plataforma JDErobot, que nos ha permitido reutilizar componentes ya desarrollados de una forma sencilla. También nos hemos apoyado en numerosas bibliotecas de software libre cómo OpenGL, GSL o GTK.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se presentará el marco general en el que se sitúa este proyecto fin de carrera. Se explicará qué se entiende por robótica en la actualidad, y las diferentes aplicaciones de la visión artificial en este campo.

1.1. Robótica

Se define Robótica como la rama de la tecnología que se dedica al estudio y diseño de máquinas que deben realizar tareas propias de los humanos o que requieren un uso de la inteligencia. A estás máquinas se las conoce como *robots*. La palabra Robot se deriva de la palabra de origen checoslovaco *robota*, que quiere decir *siervo*, dicha palabra apareció por primera vez en la literatura en la obra R.U.R (1921) (Robot Universalis de Rossum), de Karel Capek.

Los robots se componen esencialmente de tres tipos de dispositivos: sensores, procesadores y actuadores. En un robot los sensores son los encargados de recoger la información del entorno. En este grupo se situarían: el láser, el sonar o las cámaras. Estos dispositivos equivaldrían a nuestros sentidos humanos. Por otro lado se encuentran los procesadores, encargados de analizar los datos que le son suministrados por los sensores, también son los encargados de elaborar una respuesta a estos datos, y enviar la acción que deba llevarse a cabo a los actuadores. Estos últimos se encargan de interactuar con el entorno del mismo modo que los hacen nuestros músculos.

El desarrollo tecnológico ha hecho posible la construcción de diferentes modelos robóticos capaces de ayudarnos o sustituirnos en la realización de tareas peligrosas, repetitivas, difíciles o que requieren precisión. En sectores como la industria del automóvil la utilización de brazos roboticos que sustituyen al hombre en la cadenas de montaje ha sido una constante desde que en 1967 se instalaron los primeros brazos róboticos en una factoría de General Motors. Hoy en dia todas las empresas de la industria del automóvil utilizan esta tecnología en sus procesos de fabricación. En este ámbito los robot se encargan de labores de soldadura, pintado, ensamblaje de piezas, etc. En la figura 1.1 se ve una cadena de montaje de automoviles en los que trabajan multiples brazos robóticos.

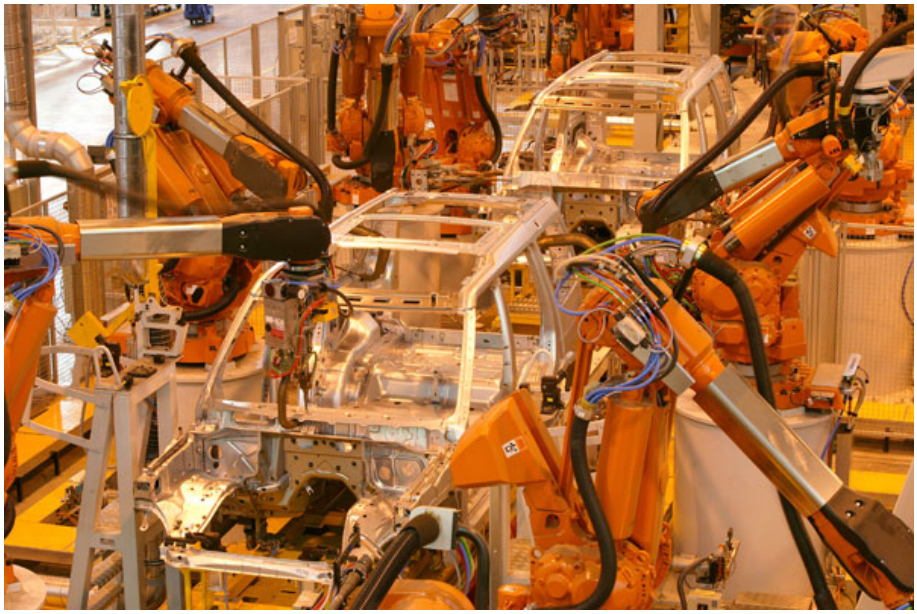

Figura 1.1: Cadena de montaje de automóviles

A finales de la década de 1970, y principios de los 80, surge una nueva rama de la robótica conocida como robótica autónoma. Se define como el área encargada de desarrollar robots capaces de interactuar con el entorno sin intervención humana. Es un campo estrechamente relacionado con la inteligencia artificial.

La combinación de la ingeniería robótica con las distintas técnicas de la visión artificial, genera componentes de gran interés para el ser humano. En la industria alimenticia el sistema desarrollado por ABB para procesos industriales, aumenta la productividad de las empresas. El brazo robótico 1.1 analiza los productos mediante su cámara para ordenarlos por lotes, para su posterior empaquetado.

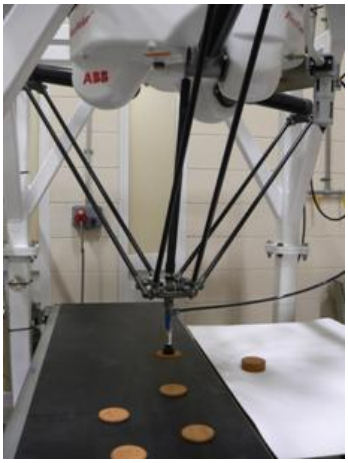

Figura 1.2: Robot de ABB para la industria de la alimentación

Otra de las aplicaciones para las que se han desarrollado robots autónomos es para la exploración espacial. La NASA ha enviado varias sondas espaciales para realizar tareas de exploración de mundos lejanos como precedente a una posterior colonización de los planetas. Las sondas *Spirit* (figura 1.3) y *Opportunity*, enviados por la NASA a Marte hace más de cinco años, han ofrecido gran cantidad de imágenes del planeta rojo y a día de hoy siguen plenamente operativas. La cámaras de vídeo incorporadas en lo alto de su mástil ayudan al robot a tomar decisiones sobre el terreno.

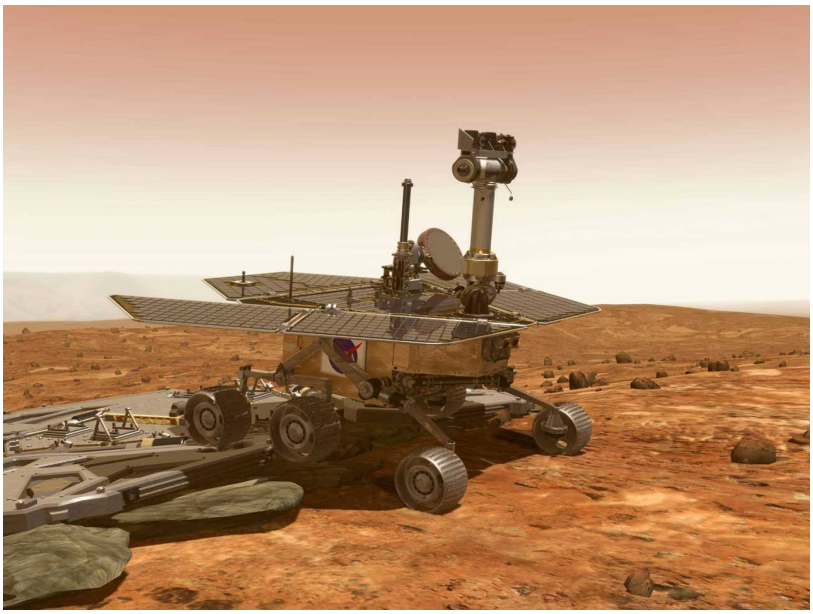

Figura 1.3: La sonda espacial *Spirit*.

En un plano más terrenal, los robot pueden ejercer un papel de mayor relevancia dentro de nuestras vidas cotidianas, desde hace unos años la empresa norteamericana *iRobot*, comercializa el robot *Roomba* (figura 1.1) para la ayuda doméstica. Este robot es capaz de aspirar una habitación de forma autónoma. *Roomba* es el primer robot autónomo para la ayuda doméstica comercializado para el gran público, y sin duda ha cosechado un reconocido éxito de ventas.

Figura 1.4: Robot doméstico *Roomba*

Pero sin duda el mayor éxito comercial en lo que a robótica domestica se refiere se ha conseguido con robots enfocados al entretenimiento. Los kit robóticos *Lego Mindstorm* comercializados por la empresa de juguetes danesa *Lego* cuentan con legiones de seguidores. Estos kits permiten la construcción y programación de robots como el que aparece en la figura 1.1.

Figura 1.5: Robot *Lego Mindstorm*

En la actualidad existe una fuerte tendencia en ir aún más lejos. En este sentido, las grandes corporaciones japonesas como *Toyota* u *Honda* trabajan en el desarrollo de robots humanoides autónomos, con la intención de solventar un problema acuciante en la sociedad japonesa, como es el rápido envejecimiento de la población y su falta de cuidado en el ámbito doméstico. El propio Bill Gates, responsable de la llegada de los ordenadores personales a los hogares de todo el mundo, afirmaba en el artículo *A robot in Every Home*, para la revista *Scientific American* [Gates, 2007], que la situación de la robótica actual es muy similar a la del ordenador personal hace ya varios años, y que él confiá en que a medio-largo plazo empezaremos a tener robots en nuestras casas, y a verlos progresivamente en las calles de nuestras ciudades.

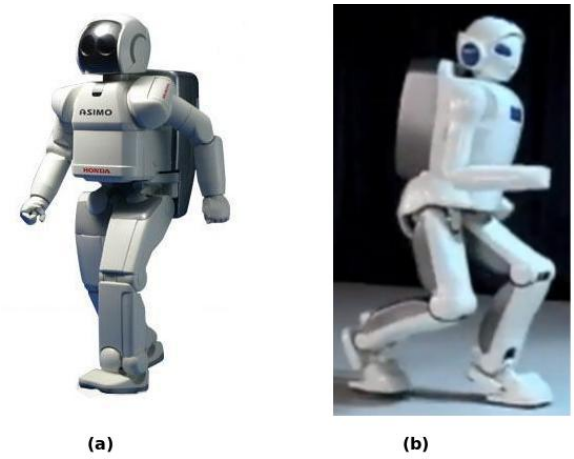

Figura 1.6: (a) Robot *Asimo* desarrollado por Honda y (b) prototipo de Toyota

Con el objetivo de favorecer la investigación en el campo de la robótica autónoma, diferentes empresas han desarrollado sus propios modelos de robot humanoides, el robot *Asimo* (figura 1.6), diseñado por Honda, y su homólogo el prototipo (figura 1.6) diseñado por Toyota, cuentan con una gran agilidad motriz y utilizan sus cámaras como sensores principales, siendo el modelo de Honda capaz de reconocer y aprender nuevos objetos. A pesar de los logros conseguidos estos robot deben servir para madurar las tecnologia de movimiento, de interacción robot-persona, etc.

1.2. Visión artificial en robótica

La visión computacional es una rama de la inteligencia artificial encargada de obtener información de las imágenes captadas por cámaras conectadas al ordenador. La cantidad de información que nos ofrecen las cámaras puede ser enorme, y muy diversa, de aquí la dificultad de desarrollar aplicaciones capaces de entender esta información y desarrollar una respuesta acorde con dicha información.

El inicio de la visión artificial se produjo en 1961, por Larry Roberts, quien tuvo la necesidad de conectar una cámara por primera vez a un computador, y analizar la información que ésta ofrecía. Roberts creó un programa que podía ver una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva. A partir de entonces numerosos científicos han tratado el problema de analizar e interpretar la información dada por una cámara. Esto no resultó ser trivial y supuso la creación de lo que hoy llamamos visión artificial.

La gran ventaja del uso de cámaras para analizar información del entorno es la cantidad de información contenida en las imágenes. Dicha información no es ofrecida por ningún otro tipo de dispositivo. Además, el coste de fabricación de dispositivos ópticos hoy en día es mucho menor que otros tipos de sensores. La dificultad radica en la complejidad para la extracción de dicha información. Algo que puede resultar sencillo de interpretar para los humanos, es de una complejidad abrumadora para una máquina.

A comienzos de los años 90 comenzaron a aparecer procesadores y tarjetas gráficas capaces de procesar esta información en un tiempo razonable. La complejidad de la visión artificial ha llevado a dividir el problema en otros muchos más sencillos.

Ciñéndonos a la robotica, la visión artificial en los robots se utiliza para diversas tareas, tales como: El reconocimiento de objetos de interés, el seguimientos de objetos, la auto-localización del robot en el entorno, la navegación del robot por el entorno, evitar obstáculos, interactuar con los humanos, etc. A pesar de que obtener información mediante visión artificial tiene una gran complejidad, las cámaras se han convertido en el sensor principal de los robots para percibir lo que les rodea.

Un ejemplo de robot que usa visión es el robot *mSecurit* (figura 1.7) desarrollado por la empresa española *Movirobotics*, es capaz de ayudar en tareas de vigilancia en naves industriales o entornos de oficina, esta dotado tanto de cámaras convencionales como de una cámara térmica. siendo capaz de analizar las imágenes y detectar intrusos en el área vigilada.

Figura 1.7: *mSecurit* robot móvil de vigilancia

Otro ejemplo de robots que se apoyan en visión son los participantes en el *DARPA Urban Challenge* [1], competición con vehículos totalmente autónomos capaces de moverse entre el tráfico de una ciudad para alcanzar determinados objetivos marcados. Para realizar estás tareas se utiliza de forma intensiva las cámaras que porta el vehículo, realizando análisis de la lineas de los carriles de la carretera o identificando las diversas señales de tráfico. En la figura 1.8 se muestra a uno de los participantes del último de estos retos organizado en Noviembre de 2007.

Figura 1.8: Vehículo autónomo, ganador del *Urban Challenge* en 2007

[1] http://www.darpa.mil/grandchallenge/index.asp

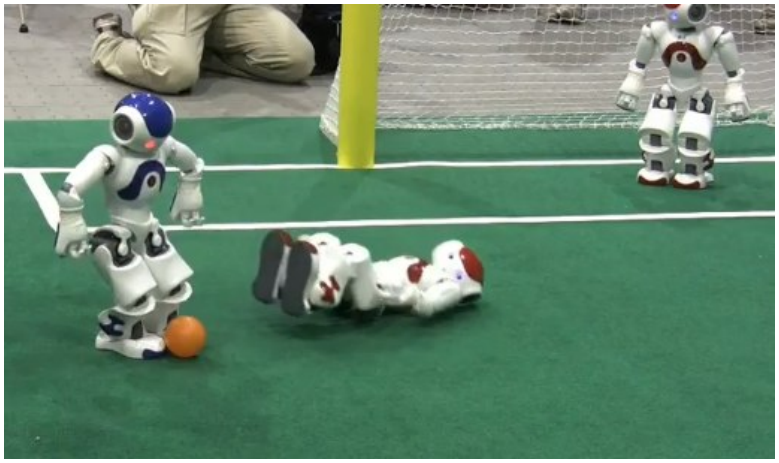

Figura 1.9: Partido de fútbol entre robots *Nao* durante la *Robocup*.

Un ejemplo de robot autónomo humanoide que utiliza técnicas de visión artificial es el robot *Nao* desarrollado por *Aldebaran*. Debido a su relación calidad/precio es ampliamente utilizado en proyectos de investigación. El grupo de Robótica de la URJC, posee varios modelos y participa con ellos en la competición internacional llamada *Robocup* [2]. La competición consta de varias pruebas, donde los diferentes grupos de investigación demuestran sus avances. Una de las pruebas más populares es el partido de fútbol. En esta prueba, la cámara, como sensor principal, ayuda al robot a localizar la pelota [Agüero Durán, 2010], auto-localizarse así mismo sobre el terreno de juego [Perdices García, 2009], y localizar al resto de jugadores, entre otra tareas.

1.3. Reconstrucción 3D

Uno de los usos tradicionales de la visión en robot es la reconstrucción 3D del entorno del robot, para que este pueda navegar de modo seguro y pueda tomar decisiones de movmiento en base a esa reconstrucción.

La motivación de este proyecto es crear un sistema de reconstrucción tridimensional con un par estéreo de cámaras a bordo de un robot. El sistema construirá un mapa tridimensional del mundo a partir de la información captada por sus cámaras. El mundo será un mundo estático creado con el simulador Gazebo.

En el Grupo de Robótica de la Universidad Rey Juan Carlos se han abordado problemas de visión artificial en robots, localización visual y sistemas de atención desde diferentes

[2] http://www.robocup.org/

puntos de vista, todos ellos forman los antecedentes inmediatos de este proyecto y los precursores directos sobre los que nos hemos apoyado. En un principio, se investigó un sistema de atención con una única cámara que mediante movimientos sacádicos reconstruía una imagen general de la escena [Martínez de la Casa Puebla, 2005]. También se estudió la navegación de robots a través de sistemas de atención en tres dimensiones [León Cadahía, 2006].

Más adelante se investigaron algoritmos de percepción en tres dimensiones que usaban un par estéreo simulado y técnicas de triangulación y segmentación [Esteban Pacios, 2007]. En este trabajo se identificaban los bordes de los objetos en la imagen y se realizaba una lenta reconstrucción de estos bordes mediante puntos, tal y como muestra la figura 1.10.

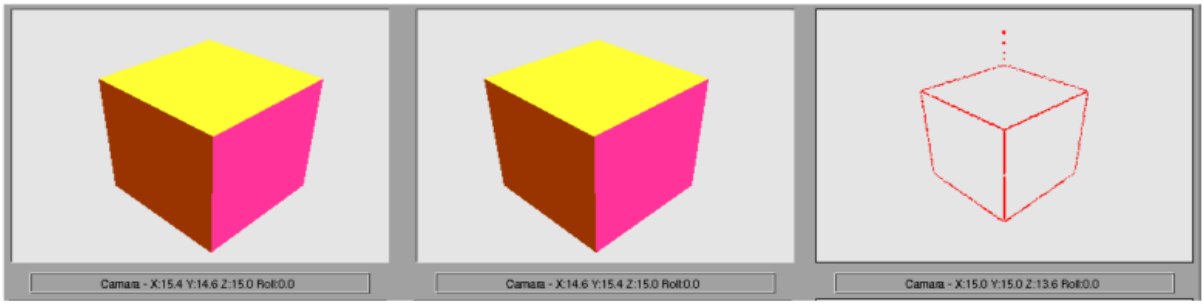

Figura 1.10: Imagen captada por las cámaras y reconstrucción 3D

Una vez reconstruidos los bordes se realizaba un análisis de estos puntos proponiendo tres métodos para segmentar bordes que se muestran en la figura 1.11: mediante segmentación basada en esquinas, mediante reconstrucción incremental y un algoritmo híbrido.

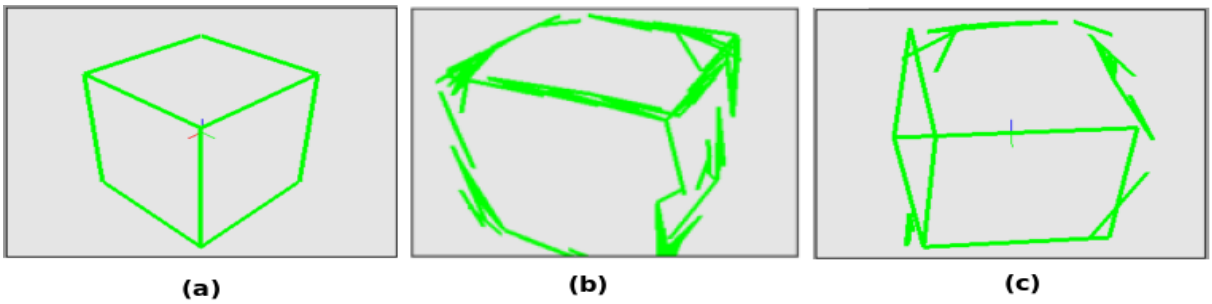

Figura 1.11: Reconstrucción (a) basada en esquinas, (b) incremental e (c) híbrida

En el proyecto fin de carrera de [Mendoza Baños, 2008] también se ha estudiado la reconstrucción 3D de una escena, esta vez con cámaras reales fijas, mediante la

triangulación de los puntos de interés de las imágenes captadas por un par estéreo. En este proyecto se realiza un filtrado de bordes para obtener el conjunto de puntos en el plano imagen que se van a triangular mediante el par estéreo. En la figura 1.12 se aprecia una reconstrucción de una escena sencilla.

Figura 1.12: Reconstrucción de un escenario sencillo

La reconstrucción era muy lenta y poco precisa en escenarios difíciles, como muestra la figura 1.13. Sobre estos algoritmos estudiados se apoyan en gran parte los algoritmos del presente proyecto.

Figura 1.13: Reconstrucción de un escenario dificil

Otra técnica probada en el grupo para la reconstrucción 3D es mediante el uso de algoritmos evolutivos [García Martínez, 2007]. En este proyecto se realizaba una recosntrucción 3D a partir de las imagenes de un par estereo de cámras simuladas. El sistema generaba distintos grupos de poblaciones, para buscar el candidato ideal que reconstruyera objetos en la imagen. Para obtener la reconstruccion 3D se utilizaron tres tipos distintos de algoritmos: algoritmo de moscas, algoritmo de segmentos 3D fijos y algoritmo de segmentos 3D variables, tal y como muestra la figura 1.14.

Posteriormente se investigó ligar la reconstrucción 3D con la atención visual. La atención del sistema se centraba sobre ciertos objetos, atendiendo a su color, bordes o movimiento,

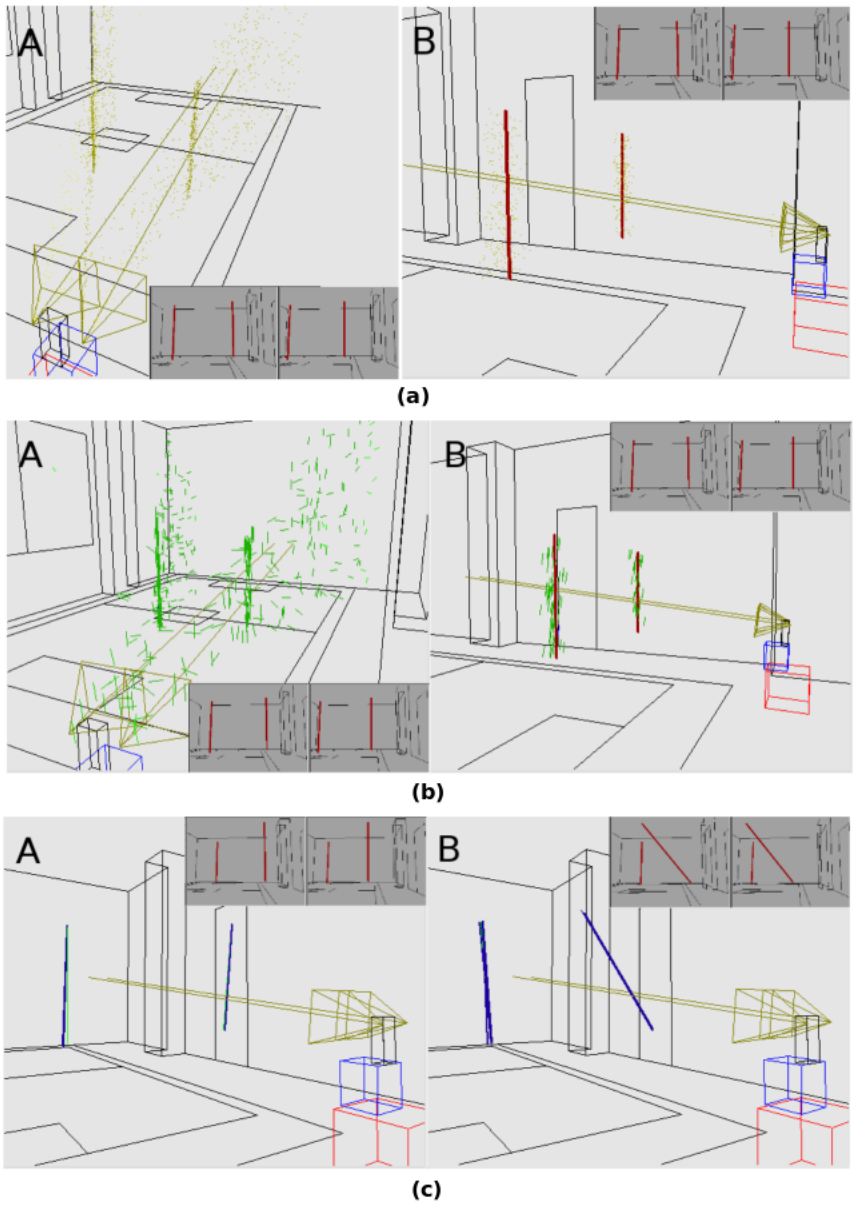

Figura 1.14: (a) Algoritmo de moscas, (b) algoritmo de segmentos 3D fijos y (c) algoritmo de segmentos 3D variables

y reconstruirlos mediante la triangulación de sus puntos de interés [Calvo Palomino, 2008]. El sistema calculaba un mapa de saliencia para las imágenes que captaban sus cámaras y determinaba qué punto debía reconstruir en cada instante. Cada píxel de interés de la imagen tenía una puntuación atendiendo a sus propiedades de color, bordes y movimiento, y estas se podían combinar entre ellas. En la figura se muestra una reconstrucción de una escena aplicando filtro de bordes, la visualización se lleva a cabo con OpenGL.

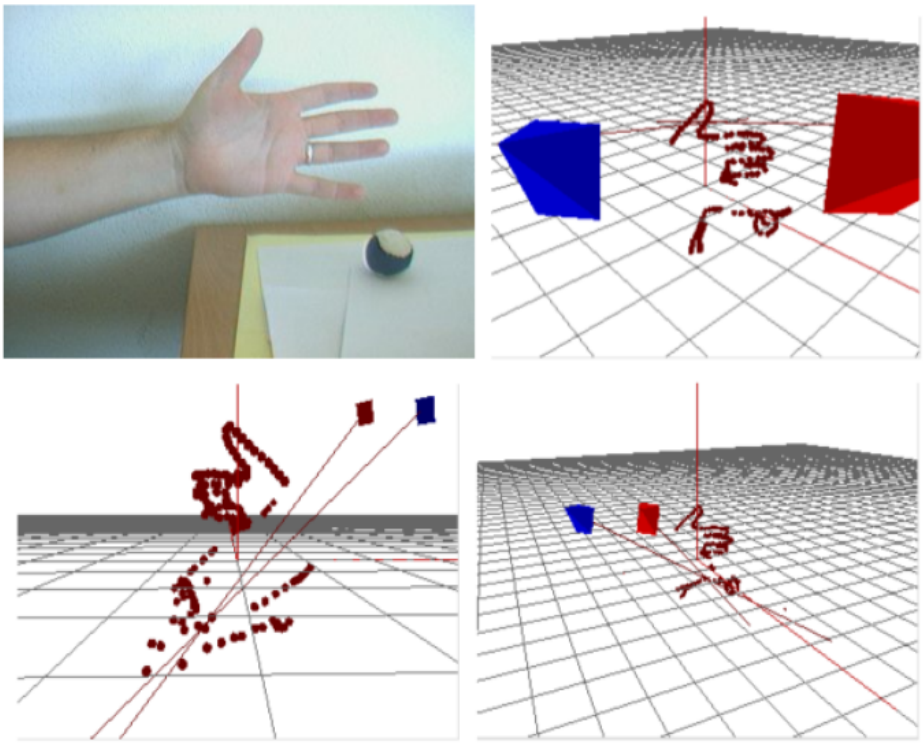

Figura 1.15: Reconstrucción de un escenario y visualización con OpenGL

La reconstrucción se realizaba con puntos 3D, y aplicaba restricciones en la búsqueda del píxel homólogo en la otra imagen consiguiendo un comportamiento más vivaz y rápido de la aplicación. En la figura 1.15 se aprecia una reconstrucción con filtro de bordes, y su visualización. Además, en este proyecto se introdujo el concepto de hipótesis perceptivas. Se manejaba el concepto *cuadrado*, el sistema podía hipotetizar la posición del cuarto vértice a partir de los otros tres restantes tal y como muestra la figura 1.16.

Por último, se planteó la posibilidad de triangulación de estos puntos siguiendo un método novedoso, mediante movimientos de vergencia de las cámaras, incluyendo un modelo geométrico que mantenía caracterizadas las cámaras en cada momento [Abella Dago, 2009]. En la figura 1.17 se muestra el par estéreo de cámaras utilizadas para el proyecto y la reconstrucción de una escena mediante puntos 3D. La calidad de la

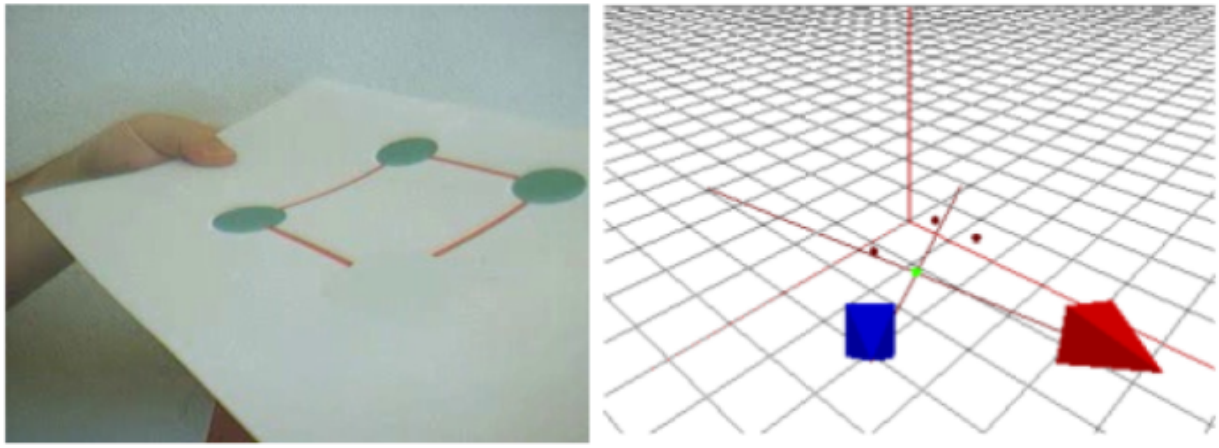

Figura 1.16: Reconstrucción 3D vértices de un cuadrado

reconstrucción 3D depende en gran medida de la calibración de las cámaras y del modelo geométrico implementado.

En este proyecto también se introdujo la primitiva segmento como herramienta para la reconstrucción 3D, tal y como muestra la figura 1.18

Con el objetivo de avanzar en estas líneas de reconstrucción 3D desde visión se plantea el presente proyecto. En concreto, realizamos un análisis de los segmentos rectos de la imagen, y tratamos de situar fielmente estos segmentos en el espacio tridimensional, en lugar de puntos 3D. Realizamos la percepción de objetos complejos por hipotesis. Incluimos una potente memoria visual 3D más extensa, que permite al robot moverse por el entorno realizando la reconstrucción 3D.

La memoria está dividida en seis capítulos. El segundo capítulo trata los objetivos concretos y requisitos planteados para el proyecto. El tercer capítulo ofrece una descripción de la infraestructura utilizada. La implementación y los detalles técnicos de esta investigación se encuentran en el cuarto capítulo. En el quinto capítulo detallamos varios experimentos con los que validar experimentalmente las técnicas de reconstrucción 3D desarrolladas. Por último, en el sexto capítulo se presentan las conclusiones extraídas de este estudio y se proponen posibles líneas futuras de investigación.

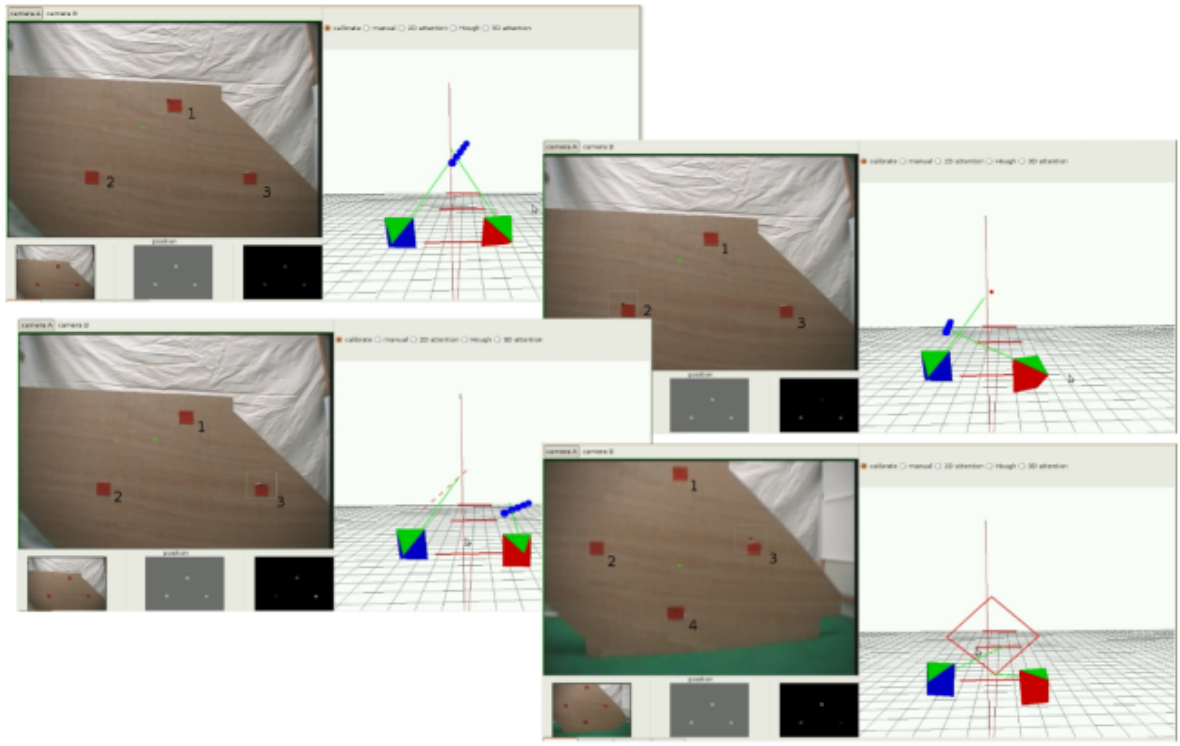

Figura 1.17: Reconstrucción de una escena mediante algoritmo de vergencia

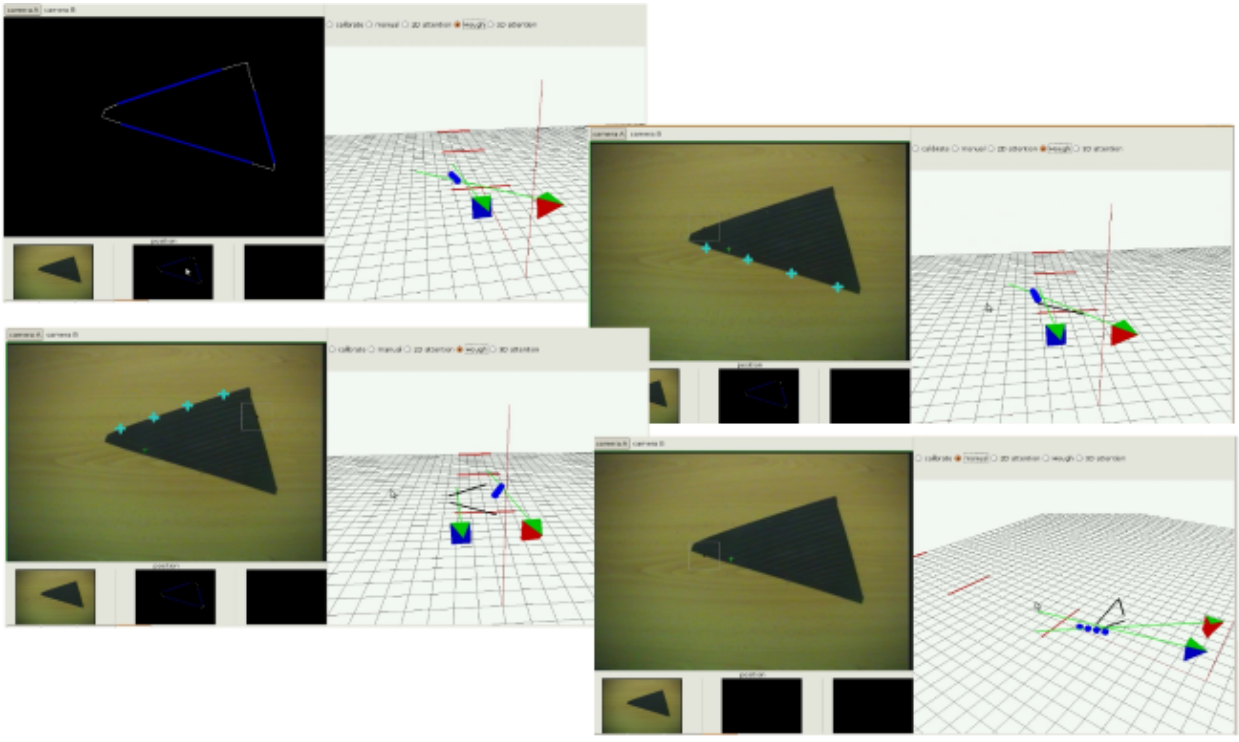

Figura 1.18: Reconstrucción 3D de una figura sencilla mediante segmentos

Capítulo 2

Objetivos

Una vez se ha descrito el contexto general en el que se desarrolla nuestro trabajo, en este capítulo se expondrán los objetivos concretos que se pretenden alcanzar, los requisitos que se desean satisfacer, y la metodología que se ha puesto en práctica para conseguirlos.

2.1. Descripción del problema

El objetivo de este proyecto es crear una aplicación capaz de *reconstruir un mundo tridimensional por el que se mueve el robot, en terminos de segmentos 3D* desde las imágenes captadas por un par estéreo de cámaras a bordo de un robot. Esa reconstrucción forma una suerte de memoria visual 3D muy potente, compuesta por segmentos 3D y objetos.

Ese objetivo general lo hemos articulado en tres subobjetivos que nos permitirán completar el objetivo principal, y que se describen a continuación:

1. El sistema analizará las imágenes ofrecidas por el par estéreo, y obtendrá los segmentos rectos que aparezcan en ellas.

2. Una vez identificados estos segmentos 2D en las imágenes, el sistema calculará la posición en el espacio tridimensional de los mismos, obteniendo un conjunto de segmentos 3D instantáneos.

3. Se implementará una memoria visual 3D de objetos que almacenará los segmentos 3D a largo plazo en coordenadas absolutas y un conjunto de estímulos complejos que ayudarán a la representación del mundo reconstruido. El sistema debe soportar el uso sobre un robot móvil que se desplaza grandes distancias por su entorno.

Los algoritmos desarrollados deberán ser validados experimentalmente. Se probarán en diferentes mundos virtuales, donde se situará un robot *Pioneer DX* provisto de un par de cámaras estéreo, para poder mejorar los algoritmos sin necesidad de usar el robot real. Las imágenes captadas por las cámaras serán la entrada para nuestra aplicación. Como resultado de salida obtendremos un mapa tridimensional del mundo simulado.

El sistema tendrá una interfaz gráfica que permitirá visualizar lo reconstruido. La representación gráfica deberá ser fidedigna y ayudar en la detección de errores.

Con este sistema se pretende explotar el grado de libertad que concede la plataforma *Pioneer* y su fiable odometría, siendo capaz de *salir fuera del laboratorio* y representar objetos dispersos en el espacio.

Queda fuera de este proyecto los algoritmos que deciden hacia donde debe mirar el robot en cada momento o como debe moverse. Nuestro sistema será pasivo, irá recosntruyendo lo que vaya apareciendo delante de las cámaras del robot.

El robot será dirigido a través de la misma aplicación de forma teleoperada y el sistema soportará el uso de cámaras móviles, con idea de abrir camino a visión atentiva.

2.2. Requisitos

Teniendo en cuenta los objetivos de la sección anterior, el proyecto deberá satisfacer una serie de requisitos descritos a continuación:

- El sistema se ejecutará sobre un entorno *GNU/Linux* con libertad para elegir la distribución. No obstante, se recomienda utilizar la distribución *Debian Ubuntu*[1].
- Este proyecto hará uso de la arquitectura de desarrollo JDERobot, con la que trabaja el grupo de robótica de la URJC. Esta arquitectura está desarrollada sobre el lenguaje de programación C, C++ y Python. Nuestro proyecto será desarrollado íntegramente en los lenguajes C y C++.
- El sistema implementado trabajará sobre el simulador robótico Gazebo en cualquiera de sus versiones. No obstante se recomienda la versión 0.7.0 .
- Los algoritmos utilizados deberán ser lo suficientemente eficientes como para ser ejecutados de forma iterativa en el Robot Pioneer simulado, favoreciendo que el robot aún en movimiento, consiga reconstrucciones precisas.

[1] http://www.ubuntu.com/

- La precisión conseguida en el posicionamiento de los segmentos deberá ser del orden de centímetros, siendo aceptable un error en la triangulación en 3D de menos de 10 centímetros, a una distancia menor de 2 metros.

- Licencia: El código del proyecto es software libre, por lo que será liberado bajo la licencia **GPLv3**[2].

2.3. Metodología de desarrollo

El ciclo de vida elegido para desarrollar este proyecto es el *modelo de desarrollo en espiral basado en prototipos*(figura 2.1). El modelo se basa en una serie de ciclos repetitivos, con cada uno de los cuales el proyecto final va ganando madurez. Con cada ciclo se consigue un prototipo. La ventaja que nos ofrece este modelo frente a otros, es el concepto de *evaluación de alternativas*, que concede flexibilidad a la hora de poder variar las funcionalidades del sistema. Por esta razón es un modelo muy apto para desarrollos en investigación.

Figura 2.1: Modelo en espiral.

Este modelo se caracteriza por la relación de las subtareas existentes en cada ciclo. En cada uno de estos ciclos, existen cuatro etapas:

- Análisis de requisitos: Se determinan los objetivos que debe conseguir el prototipo, en esta iteración. Con cada nueva iteración la funcionalidad va en aumento, al igual que la complejidad del producto, acercándonos al componente final.

[2]http://www.gnu.org/licenses/gpl-3.0-standalone.html

- Evaluar alternativas: Determina las distintas alternativas a seguir para conseguir los objetivos marcados en la etapa anterior. Además permite modificar los planteamiento realizados en iteraciones anteriores, concediendo flexibilidad al modelo. Así mismo, es prioritario minimizar los riesgos.

- Desarrollar y verificar: En esta etapa se crea el prototipo, según lo discutido en las etapas anteriores, y se comprueba que su funcionamiento es correcto.

- Planificar: Atendiendo a los resultados de la verificación al final de la etapa anterior, se planifica la siguiente iteración, teniendo en cuenta cualquier posible modificación fruto de los errores detectados durante el ciclo.

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 programado reuniones semanales con el tutor del proyecto para determinar los objetivos que se pretendían alcanzar y para evaluar las alternativas de desarrollo.

2.3.1. Plan de trabajo

El desarrollo de este proyecto fin de carrera se ha dividido en diferentes fases. Durante el transcurso del desarrollo se ha ido completando una bitácora [3], con abundante material multimedia, que refleja de manera detallada las distintas etapas seguidas en el desarrollo:

1. **Familiarización con la plataforma** En esta etapa crearemos una primera aplicación para familiarizarnos con la plataforma de desarrollo software JDERobot, ya que iba a ser la utilizada en el proyecto. Este primer componente implementará un comportamiento autónomo sencillo para una plataforma robótica Pioneer simulada en *Gazebo*. Para obtener soltura en el manejo de cámaras simuladas en *Gazebo*, implementaremos un comportamiento autónomo sencillo con un control basado en visión. En la programación de estos componente utilizaremos tecnologías que nos resultarán de utilidad en la aplicación final, como la creación de Interfaces de usuario con GTK/Glade2, o la creación de mundos simulados en Gazebo.

2. **Técnicas de visión computacional** Para tener un primer contacto con las técnicas de visión computacional necesarias para el desarrollo de nuestro proyecto realizaremos varios nuevos componentes para JDErobot. Todos los componentes utilizarán la

[3] http://jderobot.org/index.php/Miangolarra

librería gráfica de Intel, *OpenCV*, para aprender a manejar las distintas funciones de filtros de bordes, obtención de segmentos, filtros de color y flujo óptico.

3. **Análisis de imagen 2D** En esta etapa se estudiarán los distintos métodos para obtener segmentos 2D rectos en el plano imagen.

4. **Triangulación de segmentos 3D** En esta etapa comenzamos a construir las primeras utilidades de nuestro proyecto. Aplicando lo aprendido y buscando información en otro proyectos [Calvo Palomino, 2008], [Abella Dago, 2009], desarrollaremos varios algoritmos para la triangulación de puntos de interés en la escena con el robot estático. Implementaremos un algoritmo para la triangulación de segmentos en el espacio. Crearemos un modelo geométrico que nos permita expresar los segmento 3D según diferentes sistemas de referencia. En esta etapa, comenzaremos a crear escenas en 3D con OpenGL.

5. **Memoria de objetos 3D.** La etapa anterior facilita la implementación de una memoria de segmentos 3D a largo plazo que permitirá al robot moverse por el mundo al mismo tiempo que lo reconstruye. Además se incluirán métodos para obtener una memoria depurada y fiable de segmentos. El componente incluirá una memoria de hipótesis perceptivas para llevar a cabo una reconstrucción de la escena con objetos definidos *a priori* como puertas, paredes y pasillos, dichas hipótesis habrán de verificarse previamente a su representación.

2.3.2. Esfuerzo temporal

Durante el desarrollo he ido rellenando un diario de trabajo que muestra el esfuerzo temporal que ha exigido este proyecto. Las primeras fases del proyecto en las que me centré en la familiarización con la plataforma JDErobot, así como con diferentes conceptos como la visión artificial, no quedan reflejadas en la figura 2.2, aunque esto supuso aproximadamente el curso universitario de 2008/2009.

A partir de septiembre de 2009 se comenzó a desarrollar el componente final. Como se puede observar en la figura 2.2 el desarrollo de este proyecto ha requerido de un esfuerzo constante de aproximadamente unas 40 horas de trabajo semanales, desde la fecha de inicio, hasta su presentación en la convocatoria de Enero, exceptuando el periodo de la vacaciones navideñas.

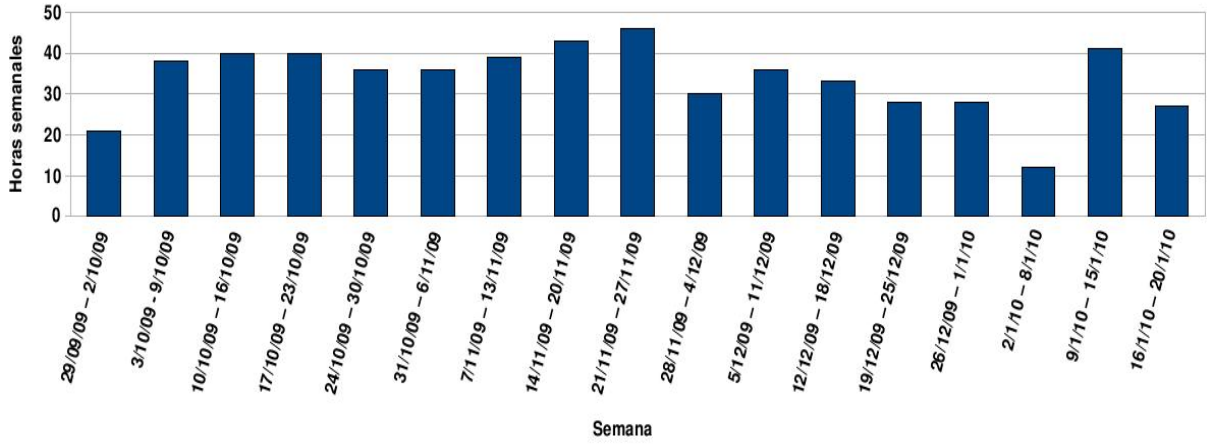

Figura 2.2: Esfuerzo temporal.

Capítulo 3

Entorno y plataforma de desarrollo

En este capítulo se describirán los recursos software empleados en el desarrollo del proyecto. Se hablará de las bibliotecas de las que hace uso el sistema, desarrolladas por la comunidad de software libre y por el propio grupo de robótica de la Universidad Rey Juan Carlos.

El sistema operativo escogido es GNU/Linux, en concreto la distribución Ubuntu 8.04 LTS (Hardy Heron). Este proyecto se ha probado en la versión Ubuntu 9.10 (Karmic Koala) con óptimos resultados igualmente. Ubuntu es una distribución enfocada a computadores personales y se ha convertido en los últimos años en una de las distribuciones Linux más populares a nivel mundial. Su interfaz sencillo, su fluida instalación en prácticamente cualquier ordenador, así como su libertad de uso, ofrecen grandes ventajas tanto a usuarios especializados como a los que no lo son.

La aplicación principal de este proyecto fin de carrera se ha programado como un componente de la plataforma JDErobot de programación de robots. El lenguaje utilizado para el desarrollo de nuestro componente es C y C++. Se ha hecho está elección puesto que la plataforma JDERobot está escrita en este lenguaje. Además, al ser el lenguaje más utilizado por el grupo de investigación, la reutilización de código de otros componentes era mucho más sencilla.

3.1. Gazebo

Para probar los desarrollos realizados sin necesidad de utilizar un robot real se ha utilizado el simulador Gazebo [1]. Se ha usado el simulador para poder avanzar más rápidamente en el desarrollo de los algoritmos, que en el futuro se probarán en el robot real.

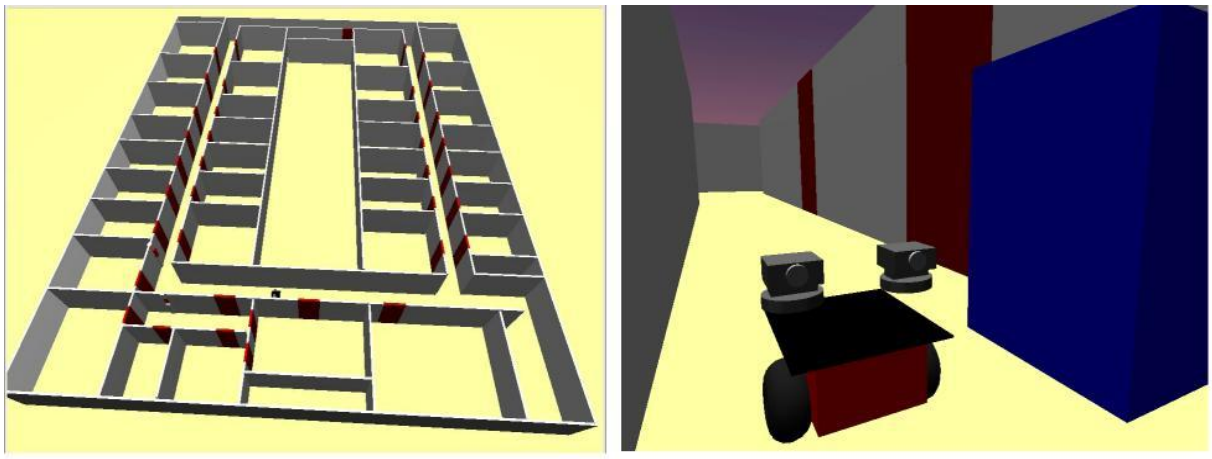

Figura 3.1: Mundo departamental y robot *Pioneer* simulado provisto de par estéreo

Gazebo es un simulador de múltiples robots en 3D (Fig. 3.1), que permite simular también sensores y objetos con los que obtener realimentación e interactuar. Gazebo es software libre bajo licencia GPL y está muy relacionado con los proyectos Player y Stage. Proporciona varios modelos de robots, como el Pioneer2DX, Pioneer2AT y SegwayRMP, y diversos mundos en los que probar estos robots. Además, permite configurar nuestros propios mundos, robots y objetos, en los que se pueden establecer las propiedades que creamos necesarias. A este respecto ha sido de gran utilidad el mundo *departamental* que podemos ver en la figura 3.1, creado por [Manuel Domínguez, 2009], que simula el ala Oeste de la primera planta del edificio departamental II del campus de Móstoles, donde están los despachos y el laboratorio del grupo de robótica. Adicionamente se han desarrollado otros mundos que nos ha sido de gran utilidad para la realización de experimentos.

El simulador Gazebo nos permite situar un robot con diversas configuraciones dentro del mundo simulado. En nuestro caso dotamos a un modelo Pioneer2AT de un par estéreo de cámaras móviles tal y como muestra la figura 3.1. Esta configuración permite tener una base móvil para el sistema y un par de cámaras para extraer información del entorno.

[1] http://playerstage.sourceforge.net/gazebo/gazebo.html

Para comunicarnos con el robot en el simulador Gazebo se ha utilizado el driver Gazebo de JDErobot, explicado más abajo, y que nos ha permitido mover el robot, conocer su posición actual dentro del mundo y obtener las imágenes necesarias.

La versión utilizada de Gazebo ha sido la 0.7.0 [2].

3.2. JDERobot

El proyecto se ha desarrollado sobre la plataforma JDERobot [3]. Esta plataforma software ha sido creada por el grupo de robótica de la URJC, se trata de una plataforma para aplicaciones con robots y visión artificial. Se ha elegido JDERobot por la facilidad con la que se pueden utilizar otras aplicaciones y componentes ya programados.

JDERobot proporciona un entorno de programación donde la aplicación se compone de distintos hilos de ejecución asíncronos llamados *esquemas*, que poseen su propio interfaz gráfico. Cada esquema es un *plugin* que se carga automáticamente en la aplicación y que realiza una funcionalidad específica que podrá ser reutilizada. Nuestro componente principal es uno de estos *plugins*.

Para conectar los esquemas de la aplicación con los sensores y actuadores del robot existe una conjunto de drivers, que también actúan como *plugins*, y son los encargados de dialogar con los dispositivos hardware concretos.

JDERobot simplifica el acceso a los dispositivos hardware desde el programa por medio de interfaces, de modo que leer la medida de un sensor es simplemente leer una variable local, y lanzar órdenes a los motores es tan simple como escribir en otra.

Para el análisis de imágenes se ha utilizado las variables *varcolorA* y *varcolorB* del interfaz *varcolor*. JDERobot permite leer de aquí toda la información captada por las cámaras, haciendo posible al esquema trabajar tanto con imágenes reales como con las simuladas. Igualmente, el interfaz *robot* proporcionan la información sobre los *encoders* del robot.

Existen dos tipos de esquemas en JDERobot, los perceptivos, encargados de realizar algún tipo de procesamiento de datos para proporcionar información sobre el robot, o sobre el mundo en el que opera, y los esquemas de actuación, que se encargan de dilucidar entre las distintas opciones para realizar una tarea.

[2] http://sourceforge.net/projects/playerstage/files/
[3] http://jderobot.org/

La versión utilizada de JDERobot ha sido la 4.3.0, lanzada en abril de 2009, y que se compone de 17 esquemas y 12 drivers. A continuación, explicaremos los diferentes drivers y bibliotecas contenidas en la suite de JDERobot que hemos utilizado en nuestra aplicación.

3.2.1. Driver Gazebo

El driver Gazebo permite la conexión una aplicación de JDERobot con el simulador Gazebo. El simulador Gazebo ofrece gran cantidad de sensores y actuadores. El driver permite la recepción de imágenes generadas por cámaras simuladas, dar órdenes de movimiento al robot, o controlar el cuello mecánico de la cámara. El driver ofrece de forma transparente, mediante interfaces, la información que se transmite desde el simulador, y traduce la información generada por nuestra aplicación para que podamos dar órdenes al robot.

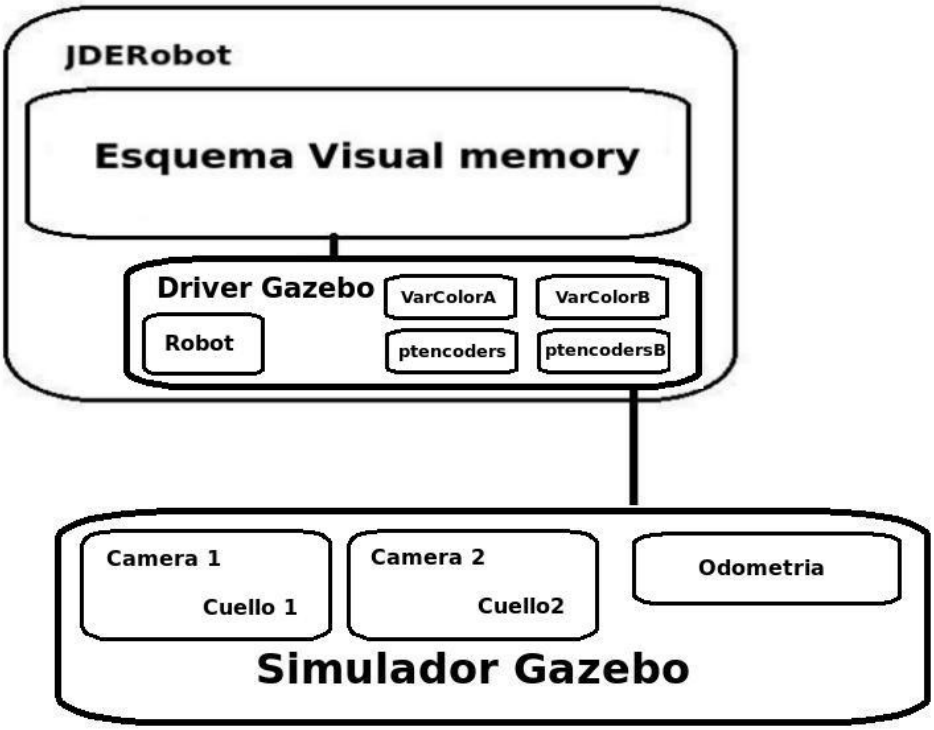

Figura 3.2: Jerarquía de comunicación entre el driver y el simulador Gazebo

El driver nos ofrece a través de *varcolorA* y *varcolorB* las imágenes captadas por las cámaras, a través de *robot* la posición y orientación del robot, y a través de *ptencoders* y *ptencodersB* el estado de los cuellos mecánicos. La aplicación puede mandar órdenes a los

motores del robot por medio de *motors*, o a los cuellos mecánicos por medio de *ptmotors* y *ptmotorsB*.

El driver incluido en la versión 4.3 de JDERobot no soporta el control de más de un cuello mecánico, por lo que fue necesario desarrollar en este proyecto un nuevo driver para ser capaces de controlar los dos cuellos mecánicos de las cámaras estéreo sobre un mismo robot. El nuevo driver Gazebo desarrollado se ha incorporado a la versión oficial de JDErobot y constituye un aporte lateral de este proyecto a la infraestructura de la plataforma. En la figura 3.2 se aprecia cómo se realiza la comunicación entre los diferentes componentes y su jerarquía.

3.2.2. Progeo

La biblioteca Progeo, proporcionada también por JDERobot, es una librería sobre geometría proyectiva que nos proporciona algunas funciones útiles para relacionar información visual de una cámara en 2D con información espacial en 3D. Esta relación se consigue principalmente mediante dos funciones:

- *Proyectar*: Función que permite obtener el píxel en dos dimensiones del plano imagen de la cámara en el que proyecta un punto del espacio en tres dimensiones.
- *Retro-proyectar*: Esta función realiza la operación inversa: A partir de un píxel en dos dimensiones en el plano imagen permite obtener la recta 3D del conjunto en tres dimensiones de puntos que proyectan en ese píxel de la imagen.

  Una tercera función contenida en la biblioteca también ha sido de gran utilidad:

- *Visualizar linea*: Esta función permite saber si una línea en el espacio 3D es visible por el campo de visión de las cámaras, y en caso afirmativo, determinar si la linea es vista en su totalidad, o tan solo parte de la misma.

Antes de utilizar estas tres funciones, es necesario calibrar la cámara que estemos utilizando para que los cálculos realizados por las funciones sean los correctos. Para ello, es necesario saber que Progeo se basa en un modelo de cámara Pinhole (Figura 3.3), que se define por un conjunto de parámetros intrínsecos y extrínsecos. Los parámetros extrínsecos consisten en la posición de la cámara en tres dimensiones y en su orientación (el foco de atención de la cámara y el *roll*), mientras que los intrínsecos son la distancia focal y la posición del píxel central.

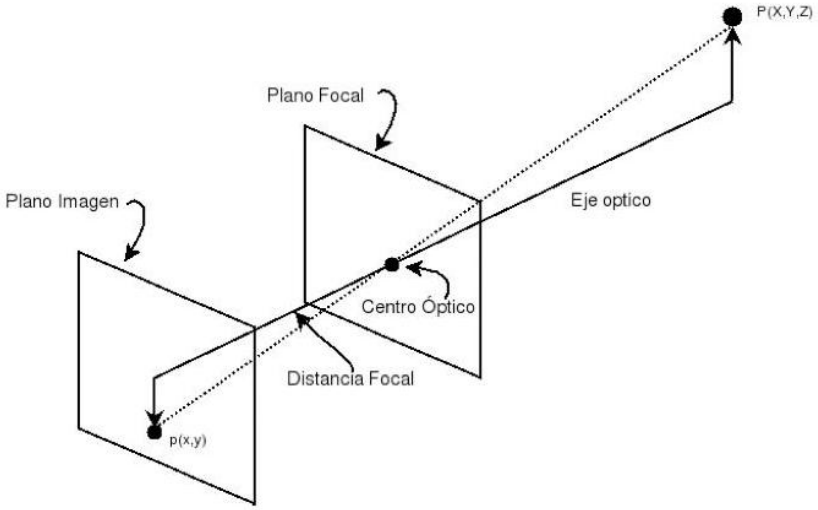

Figura 3.3: Modelo de cámara Pinhole.

Progeo ha sido muy útil en el desarrollo del proyecto, mediante la funciones *project* y *backproject* realizamos la triangulación de puntos 3D en la escena. La función *diplayline* es utilizada para la proyección de los segmentos 3D memorizados sobre el plano imagen 2D.

3.3. GTK+/Glade

GTK+ es un conjunto de bibliotecas multi-plataforma para crear interfaces gráficas de usuario en múltiples lenguajes de programación como C, C++, Java, Python, etc. GTK+ es software libre bajo licencia LGPL y es parte del proyecto GNU. Entre las bibliotecas que componen a GTK+, destaca GTK, que es la que realmente contiene los objetos y funciones para la creación de la interfaz de usuario. Ejemplos de aplicaciones desarrolladas con esta librería son el navegador web Firefox o el editor gráfico GIMP.

GTK dispone de editores tipo WYSIWYG [4], como *Glade* [5], que simplifican en alto grado la creación de interfaces gráficos. Con *Glade* podemos generar el interfaz completo mediante la adición de *widgets* que posteriormente han de ser conectados a nuestro código.

JDERobot interactúa con esta biblioteca a través del servicio *graphics_gtk*, que centraliza el acceso a esta biblioteca desde los interfaces gráficos de los distintos esquemas activos. A través de las funciones *show*, *guidisplay* y *hide*, el componente inicializa el interfaz, lo refresca de manera iterativa, o lo desactiva.

[4] What You See Is What You Get - Lo que ves es lo que obtienes

[5] http://glade.gnome.org

En nuestro proyecto hemos utilizado GTK+ para la construcción de la interfaz gráfica, ayudándonos del diseñador de interfaces *Glade* para simplificar el desarrollo. En la figura 3.4 se puede ver el interfaz gráfico completo, hecho con *Glade*, para el esquema *Colortuner*, que se ha implementado como práctica introductoria al uso de visión artificial en este proyecto.

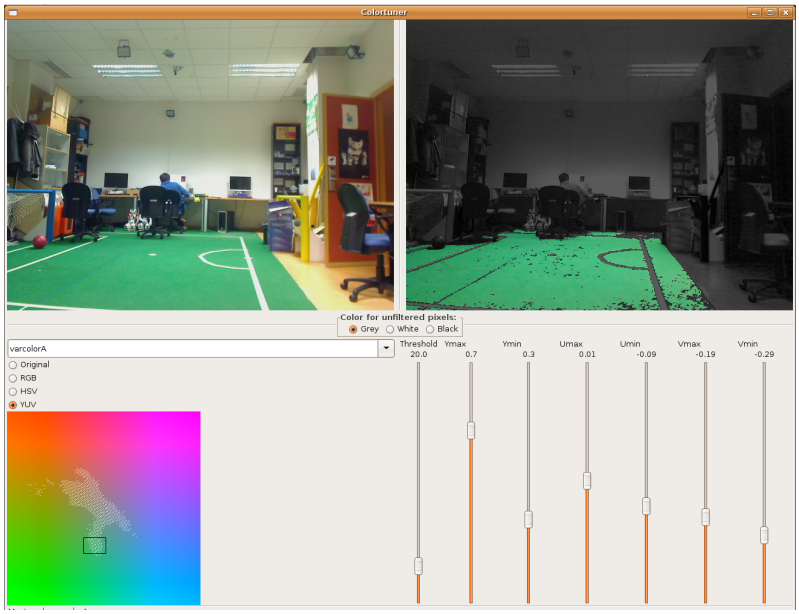

Figura 3.4: Interfaz gráfica del esquema *Colortuner*, para la creación de filtros de color

En concreto hemos utilizado la versión 2.0 de la biblioteca GTK. Este conjunto de bibliotecas se puede instalar desde los repositorios oficiales de *ubuntu*. Los paquetes esenciales son: libgtk2.0-0, libgtk2.0-dev, libgtk2.0-0-dbg, libgtk2.0-doc, libgtkextra-x11-2.0-1, libgtkextra-x11-2.0-dbg, libgtkextra-x11-2.0-dev, libgnomecanvas2-dev, libgtkglext1, libgtkglext1-dev, libgtkglext1-doc.

3.4. OpenGL

OpenGL [6] es una especificación estándar que define una API multi-plataforma e independiente del lenguaje de programación para desarrollar aplicaciones con gráficos en 2D y 3D. A partir de primitivas geométricas simples, como puntos o rectas, permite generar escenas tridimensionales complejas. Actualmente es ampliamente utilizado en realidad virtual, desarrollo de videojuegos y en multitud de representaciones científicas.

[6] http://www.opengl.org

Las operaciones necesarias para la materialización de estos gráficos se delegan en la GPU (procesador de la tarjeta gráfica). De este modo se libera a la CPU de las tareas de generación de gráficos, quedando disponible para cualquier otra tarea. En la figura 3.5 se aprecia la versatilidad de esta biblioteca utilizada en la creación de videojuegos .

Hemos utilizado esta API en nuestras aplicaciones para la representación en 3D de la escena reconstruida por el sistema, visualizando la posición de los segmentos triangulados.

Figura 3.5: Captura del famoso videojuego Quake III

En este proyecto hemos utilizado la versión 7.0.3 de la biblioteca Mesa 3D. Este conjunto de bibliotecas se puede instalar desde los repositorios oficiales de *ubuntu*. Los paquetes esenciales son: libgl1-mesa-dev, libgl1-mesa-glx, mesa-common-dev, gle-doc, libglu1-mesa, libglu1-mesa-dev, freeglut3, freeglut3-dev, freeglut3-dbg, libglut3, libglut3-dev, glut-doc, glutg3-dev.

3.5. GSL

GSL, GNU Scientific Library, es una biblioteca numérica para la programación en C y C++. Es software libre bajo una licencia GNU pública. La biblioteca ofrece una amplia

variedad de utilidades matemáticas, como generadores de números aleatorios o cálculo de matrices. En concreto, nuestro proyecto hace uso de esta última.

Hemos utilizado la funciones del cálculo de matrices para implementar nuestro modelo geométrico. A través de la multiplicación de matrices realizamos los cambios de base necesarios para representar puntos en el espacio, con un sistema de referencia situado en las cámaras del robot, en el robot o en coordenadas absolutas del mundo. En concreto, se han implementado un conjunto de matrices homogéneas de Rotación y Translación, que tienen en cuenta la posición del robot en el mundo, su orientación, o la posición actual (los valores de PAN y de TILT) de los cuellos mecánicos de las cámaras.

En este proyecto hemos utilizado la versión 1.13 disponible en los repositorios oficiales de *ubuntu*. Los paquetes esenciales son: gsl-bin, libgsl0-dev, libgsl0ldbl.

3.6. OpenCV

OpenCV[7] es una librería de visión artificial desarrollada inicialmente por Intel. Está publicada sobre la licencia BSD, lo que permite su libre utilización en propósitos comerciales o de investigación.

Esta librería multi-plataforma, que funciona bajo sistemas operativos como Windows, Mac OS X, Linux y otros sistemas embebidos, nos proporciona un extenso conjunto de funciones para procesamiento de imágenes. El desarrollo de estas funciones se ha realizado primando la eficiencia, para lo que se ha programado utilizando C y C++ optimizados, pudiendo además hacer uso del sistema de primitivas de rendimiento integradas en los procesadores Intel (IPP). Esta primitivas son un conjunto de rutinas de bajo nivel específicas en estos procesadores y que cuentan con una gran eficiencia.

Dentro del gran número de funcionalidades que ofrece la librería, en este proyecto ha sido de gran utilidad el *filtro de Canny* para la obtención de bordes sobre la imagen. También se han utilizado las funciones de la transformada de *Hough*, en su versión *probabilística*, para la detección de segmentos rectos. Ambas funciones ha facilitado mucho la tarea de la detección de segmentos en la imagen para su posterior triangulación. En la figura 3.6 se puede ver el resultado de la transformada de *Hough* sobre una imagen real.

En la actualidad las librerías de OpenCV dependen del proyecto *Robot Operating System* [8] (ROS), un sistema operativo de código abierto pensado para utilizarse sobre robots.

[7] http://sourceforge.net/projects/opencvlibrary/

[8] http://www.ros.org/wiki/

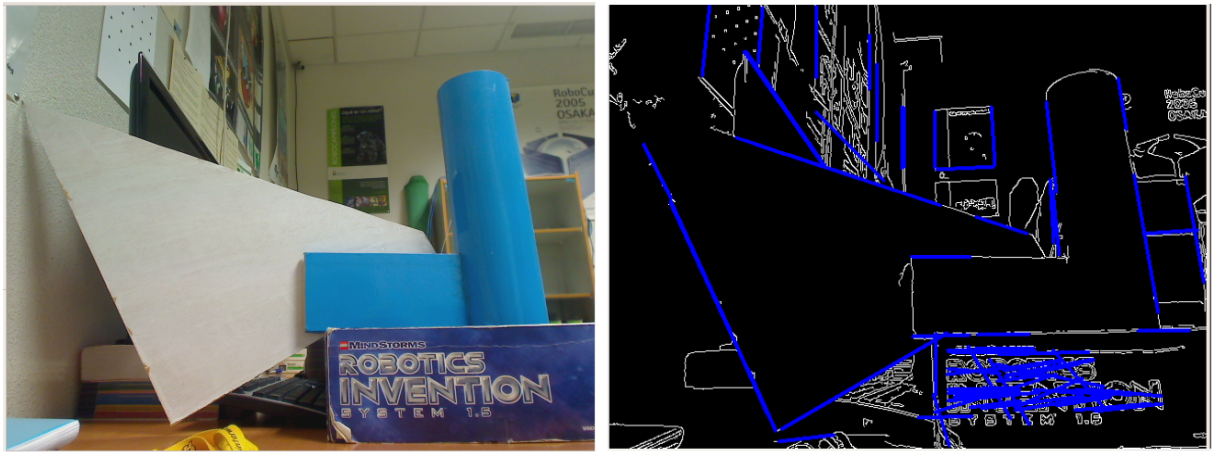

Figura 3.6: Transformada de Hough sobre una imagen real

En este proyecto hemos utilizado la versión 1.0 disponible en los repositorios oficiales de *ubuntu*. Los paquetes esenciales son: libcv-dev, libcv1, python-opencv, libhighgui-dev, libhighgui1, libcvaux-dev, libcvaux1.

Capítulo 4

Descripción Informática

Una vez explicados los objetivos y las herramientas necesarias para el desarrollo del proyecto, se pasará a explicar en profundidad el componente diseñado y programado. Se dará un vistazo general al diseño global del sistema y posteriormente, se explicarán con más detalle las funciones concretas de cada uno de los subcomponentes.

4.1. Diseño general

El propósito del proyecto, como ya hemos explicado, es implementar una aplicación capaz de reconstruir escenas tridimensionales a partir de la información recibida por un par estéreo de cámaras, valiéndose para ello de la primitiva segmento. La aplicación debe tener en cuenta el movimiento del robot y de sus cámaras móviles para situar estos segmentos correctamente en el espacio 3D. A partir de dichos segmentos el programa será capaz de generar hipótesis perceptivas, proponiendo el lugar de objetos en el mundo, como puertas ó pasillos. Estas hipótesis deberán ser correctamente confirmadas en las imágenes. Finalmente llevará a cabo una visualización de la escena con la librería gráfica OpenGL.

La aplicación final engloba tres partes bien diferenciadas que se ven en la figura 4.1. En primer lugar, el sistema de análisis 2D proporciona aquellos segmentos 2D que se perciben en el momento actual en el plano imagen. En segundo lugar, el algoritmo de reconstrucción 3D con el que se sitúan en el espacio los segmentos 2D instantáneos. Por último, la memoria tridimensional que guarda la posición de los segmentos almacenados, genera hipótesis perceptivas y calcula las predicciones de aquellos segmentos en las imagenes actuales para el sistema de análisis 2D.

La programación de la solución se ha llevado a cabo con un único esquema de JDErobot.

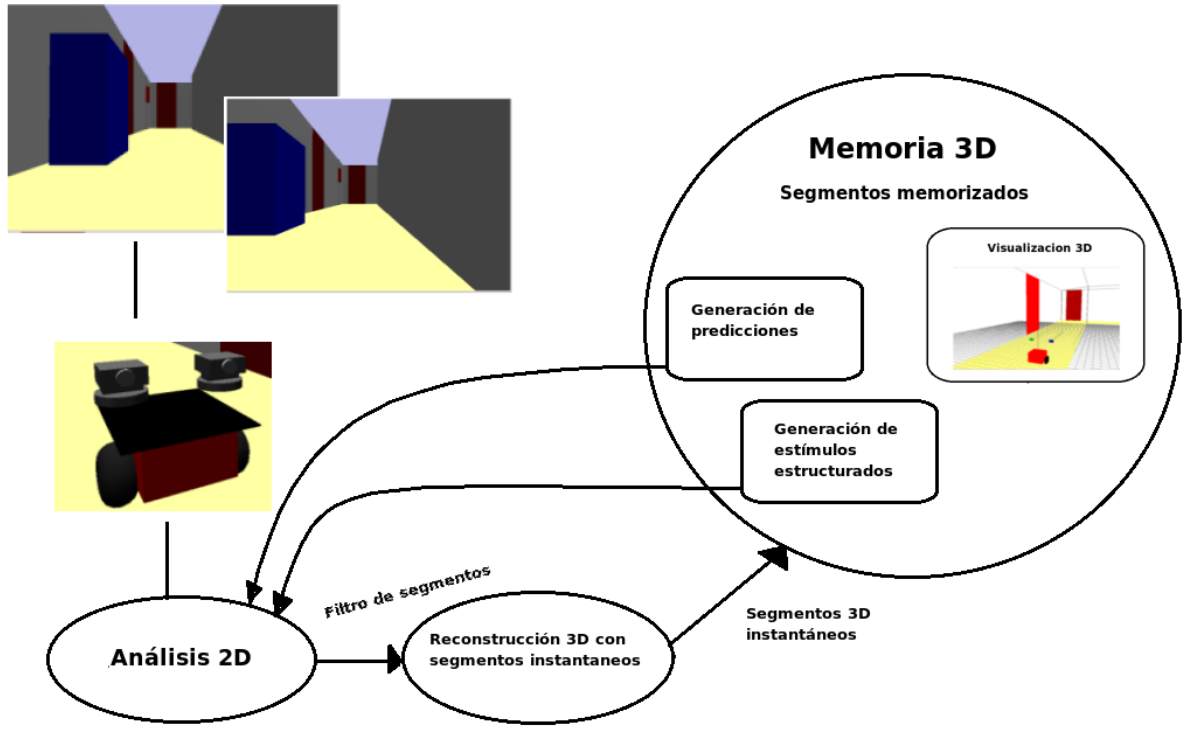

Figura 4.1: Diagrama de bloques del sistema

El esquema recibe las imágenes de las cámaras y las procesa obteniendo segmentos en tres dimensiones. A partir de estos segmentos se obtendrán los diferentes objetos. El esquema *visual memory* es el componente principal del sistema, en él se aglutina toda la lógica del programa. El sistema trabaja sobre un mundo estático simulado en Gazebo.

El funcionamiento final de nuestro proyecto queda reflejado en la figura 4.2. El flujo natural de trabajo comienza con el análisis de las imágenes captadas por las cámaras, a estas imágenes se le aplica un *filtro de Canny* para la obtención de bordes. A partir de los bordes de la imagen aplicaremos la transformada de *Hough*, para la obtención de segmentos rectos en la imagen.

A continuación, el algoritmo de reconstrucción de segmentos 3D instantáneos situará éstos en el espacio tridimensional, mediante la búsqueda de homólogos. Estos segmentos 3D instantáneos se compararán con los almacenados en la memoria 3D, incorporándose a la misma como novedades o corrigiendo a los ya almacenados.

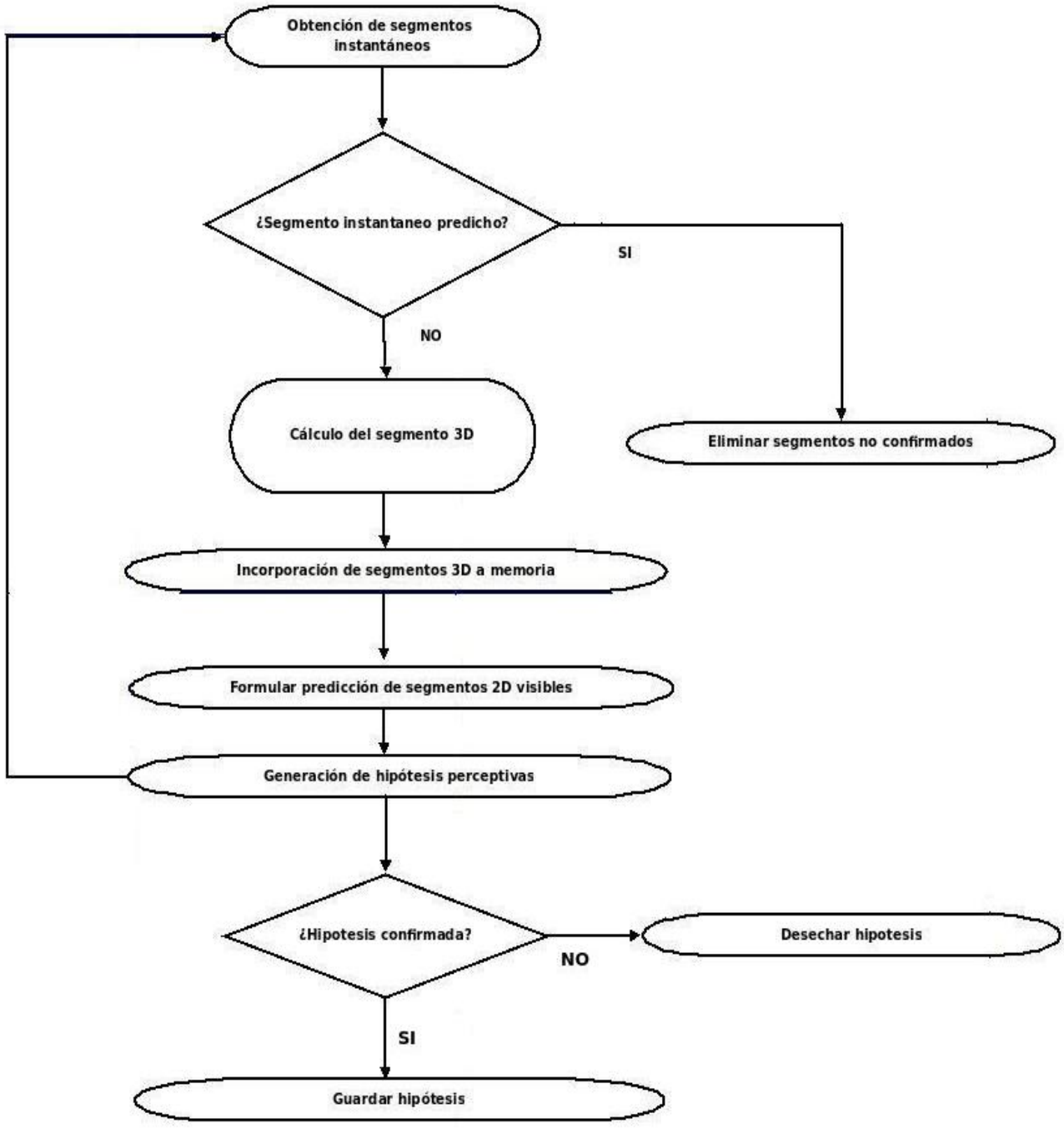

Figura 4.2: Flujo de control del comportamiento del sistema

A partir de los segmentos 3D almacenados en memoria el sistema genera en cada iteración una predicción de aquellos segmentos que debería encontrar en el plano imagen, esta predicción se le pasa al sistema de análisis 2D, provocando que algunos segmentos sean eliminados de memoria o que no lleguen a ser tratados por el sistema.

Además, con los segmentos 3D almacenados el sistema genera estímulos estructurados, considerando, por ejemplo, que dos segmentos verticales de determinada altura y equidistantes entre sí cierta distancia forman una puerta. El sistema confirmará su hipótesis mediante un filtro de color sobre la imagen donde debiera estar dicho objeto. Sobre los estímulos estructurados se pueden realizar nuevas hipótesis como los objetos pared o pasillo.

Finalmente, el sistema mostrará la representación en tres dimensiones de estos objetos. Al ser un esquema perceptivo pasivo, reconstruirá aquello que se ponga delante del robot cuando este se mueve. Esta reconstrucción se realiza en tiempo real, no es *offline*. El robot puede mantener en memoria segmentos reconstruidos anteriormente que han quedado atrás a medida que le robot se mueve y que no son visibles para el robot.

Una vez hemos ofrecida una visión general del funcionamiento del sistema, pasaremos a describir con minuciosidad las diferentes partes de la aplicación.

4.2. Sistema de procesamiento 2D

El sistema de procesamiento 2D nos devuelve los segmentos rectos en la imagen que deben ser tratados por el algoritmo de reconstrucción 3D. En esta sección se explica cómo se seleccionan los segmentos que debemos triangular. Para la búsqueda de segmentos en la imagen hemos reutilizado parte del código de Gonzalo Abella [Abella Dago, 2009].

Las cámaras son una fuente de información muy rica y densa. Las imágenes del robot tienen una tamaño de 640x480 píxeles, y a lo largo de este espacio podemos encontrar una gran cantidad de información. Por ello es imprescindible un algoritmo que seleccione aquellos segmentos rectos bien definidos en cada momento. La primitiva segmento es más compacta que la primitiva píxel.

El componente aporta una característica novedosa en la selección de aquellos segmentos que deben ser triangulados. El componente se ayudará de la memoria de segmentos 3D, explicada más adelante, para tres propósitos fundamentales: Primero, para el ahorro en el cálculo de aquellos segmentos ya memorizados. Segundo, para la confirmación de segmentos memorizados, que en caso de refutarse serían eliminados de la memoria. Por

último la imágenes captadas por las cámaras se utilizan para la confirmación de estímulos estructurados.

A continuación explicaremos los algoritmos utilizados para la obtención de segmentos rectos del plano imagen. Para ello distinguimos entre cámara maestra y esclava.

4.2.1. Cálculo de segmentos 2D

El primer paso del algoritmo consiste en realizar un filtro de Canny para la detección de bordes tanto en la cámara maestra como en la esclava. Como resultado se obtienen aquellos píxeles que son de interés en ambas imágenes. Dicho filtro es considerado como uno de los más robustos para la obtención de bordes. Con el uso de matrices de convolución y derivadas detecta los puntos donde se produce un cambio brusco en el nivel de gris. La librería gráfica *OpenCV* incluye una función que realiza dicho algoritmo:

```
void cvCanny( const CvArr* image, CvArr* edges, double threshold1,
double threshold2, int aperture_size=3 );
```

donde *image* es la imagen de entrada, *edges* la imagen destino. La imagen de salida será una máscara con aquellos píxeles que han pasado el filtro de Canny a verdadero y el resto a falso.

Una vez obtenida dicha máscara se empleará la transformada de Hough para la obtención de segmentos rectos. La transformada de Hough es un algoritmo empleado para encontrar ciertas formas dentro de una imagen, como líneas o círculos. El proceso que realiza el sistema para encontrar cada línea consiste en recorrer cada punto que se desea averiguar si es parte de una línea, es decir, cada punto que se ha identificado anteriormente como borde, calculando las posibles líneas que puedan formar parte de ese punto. Al calcular esto para todos los puntos, se determina qué líneas fueron las que más puntos posibles obtuvieron y se toman éstas como líneas en la imagen.

Hemos utilizado este algoritmo para el reconocimiento de segmentos mediante una de las funciones de *OpenCV*:

```
CvSeq* cvHoughLines2( CvArr* image, void* line_storage, int method,
                      double rho, double theta, int threshold,
                      double param1=0, double param2=0 );
```

donde *image* es la imagen de entrada, *line_storage* es un almacenamiento de memoria para los cálculos propios del algoritmo, *method* es el tipo de algoritmo que se quiere utilizar, *rho* es la distancia entre píxeles que están relacionados, *theta* el ángulo de resolución en radianes, *threshold* es un umbral utilizado por el algoritmo y *param1* y *param2* son parámetros que tienen un significado distinto dependiendo del método que se haya seleccionado. Dichos parámetros se pueden modificar en la interfaz gráfica de forma manual, para obtener mejores resultados dependiendo del entorno del robot.

Existen tres métodos distintos que se pueden utilizar, 'CV_HOUGH_STANDARD', 'CV_HOUGH_PROBABILISTIC' y 'CV_HOUGH_MULTI_SCALE', en nuestro esquema sólo hemos utilizado el método *probabilístico*.

Utilizando Hough probabilístico, *param1* es la distancia mínima que debe tener una línea para considerarla válida y *param2* es el hueco que puede haber entre 2 píxeles para considerar que pertenecen a la misma recta. En este caso obtendremos un conjunto de líneas en las que se nos marca su píxel de inicio y su píxel de final de la línea.

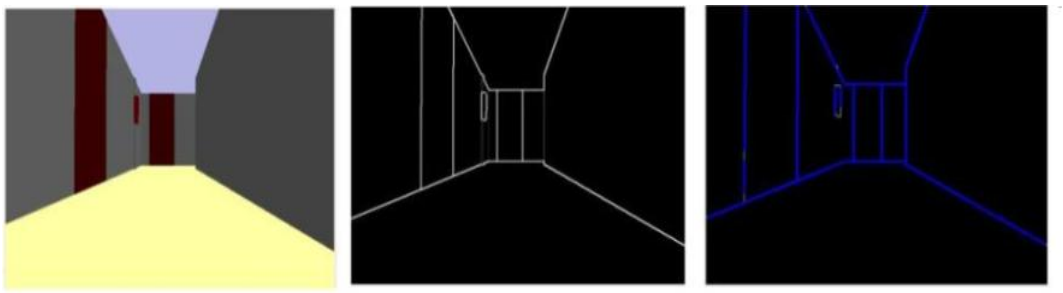

Figura 4.3: Imagen original, tras el filtrado de Canny y resultado de la transformada de Hough

El hecho de que el método probabilístico nos permitiera saber el principio y el final de un segmento nos ofrecía mayores ventajas que utilizar cualquiera de los otros dos métodos. El resultado de aplicar la transformada de Hough es una lista de segmentos en la imagen con sus píxeles de inicio y final.

Por último, en ocasiones la transformada de Hough puede considerar que existen varios segmentos alineados donde sólo existe un único segmento real. Para reducir el número de segmentos calculados se realizará un post-tratamiento de los segmentos obtenidos, que los une teniendo en cuenta si estos son adyacentes y poseen la misma inclinación. Este post-tratamiento repercute en una mejora de la triangulación, puesto que un posible error de posicionamiento del segmento es menos apreciable cuanto mayor es la longitud del mismo.

En la figura 4.4 se aprecia la parte central de los segmentos 2D obteninos por la transformada de Hough, posteriormente se realiza el post-tratamiento de estos segmentos, obteniendose un único segmento 2D para definir cada uno de los marcos laterales de las puertas.

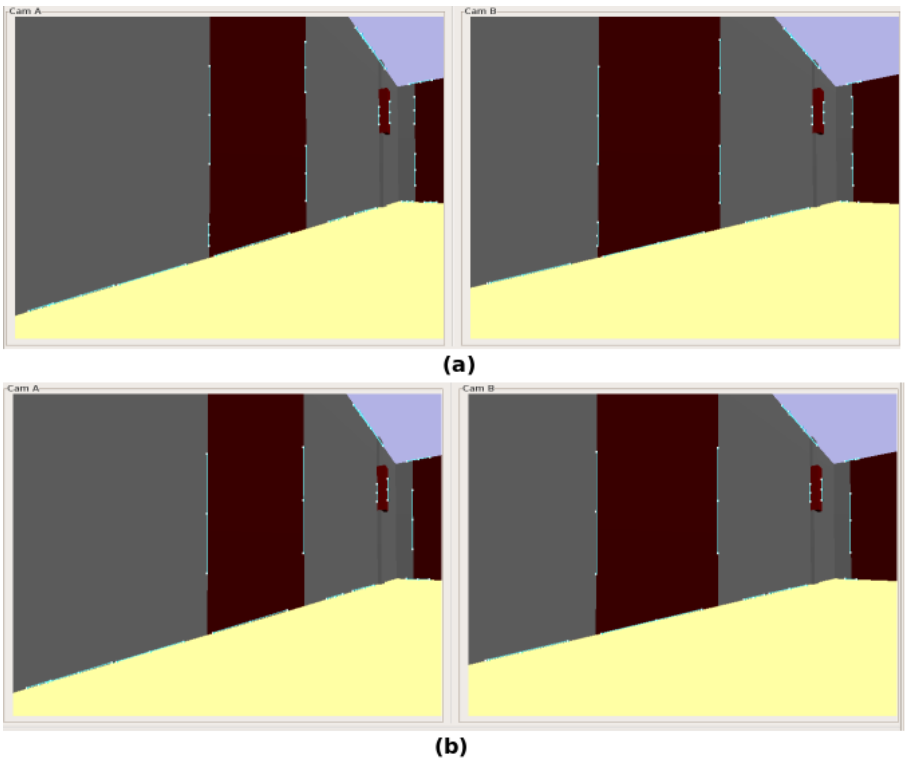

Figura 4.4: Segmentos 2D representados por varios segmentos (a), la misma escena tras el post-tratamiento (b)

4.2.2. Predicciones Instantáneas

El sistema de procesamiento 2D se apoya en la memoria 3D de segmentos para mejorar el análisis de la imagen. El sistema de procesamiento 2D es la fuente de los segmentos a triangular y por lo tanto debe devolver el mejor conjunto de segmentos rectos posible. Mediante las predicciones sobre el plano imagen el sistema corregirá, ignorará o eliminará de memoria 3D los segmentos almacenados.

Previamente al análisis de los segmentos en el plano imagen, se realiza una predicción de aquellos segmentos que deberían ser visibles en la imagen de entrada. Para ello seleccionamos aquellos segmentos almacenados en nuestra memoria 3D que se encuentran delante del robot, y que estoy viendo en cada instante. Para calcular si estamos viendo un

segmento 3D concreto, hemos utilizado la librería de geometría proyectiva Progeo expuesta en el capitulo anterior, y en concreto las funciones *project* y *displayline*. Estas funciones requieren de cámaras calibradas.

Para cada segmento susceptible de ser visto se halla su proyección en el plano imagen, mediante la función *project*:

```
int project(HPoint3D in, HPoint2D *out, TPinHoleCamera camera);
```

donde *in* es el punto 3D de entrada, *out* es la proyección 2D del punto en el plano imagen, *camera* contiene los parámetros de la cámara en la que se realiza la proyección. En caso de que la proyección se encuentre fuera del plano imagen la función devolverá -1.

A continuación se verifica mediante la función *displayline*:

```
int displayline(HPoint2D p1, HPoint2D p2, HPoint2D *a, HPoint2D *b,
TPinHoleCamera camera);
```

donde *p1* y *p2* son los puntos 2D de entrada, y principio y final de la recta, *a* y *b* son los puntos 2D de salida. *a* y *b* serán los extremos si veo la línea en su totalidad, pero varían si la veo parcialmente, y *camera* la cámara concreta. En caso, de que la proyección se encuentre fuera del plano imagen la función devolverá -1.

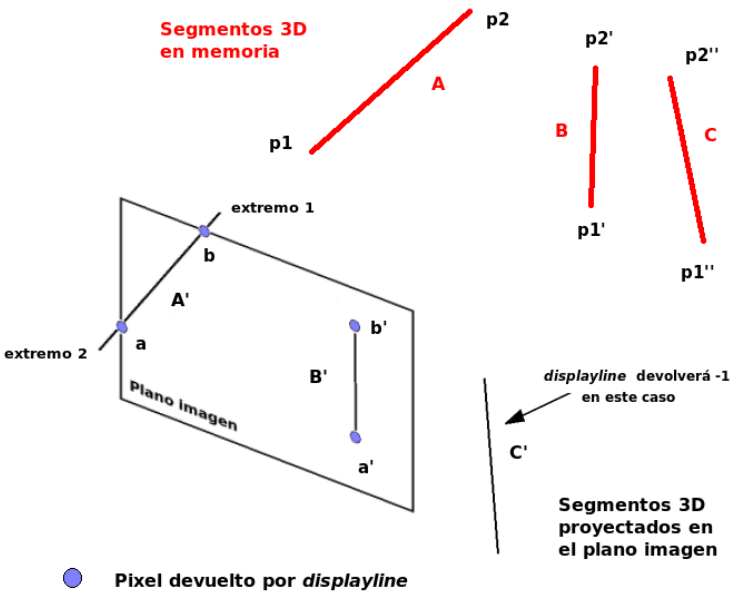

Figura 4.5: Proyección de tres segmentos 3D en el plano imagen

En la figura 4.5 se puede ver tres segmentos 3D A , B y C que se proyectan mediante la función *project* en el plano imagen de la cámara. cuando se calculan los píxeles que proyectan los extremos del segmento C la función *project* devolverá -1, puesto que no es visible. En el caso, de A y B, ambos son segmentos visibles, por lo que *project* nos devolverá la proyección en el plano imagen, pero en el caso de A sus extremos caen fuera de este plano, por lo que utilizaremos *displayline*, para asegurarnos que los píxeles devueltos por *project* son correctos.

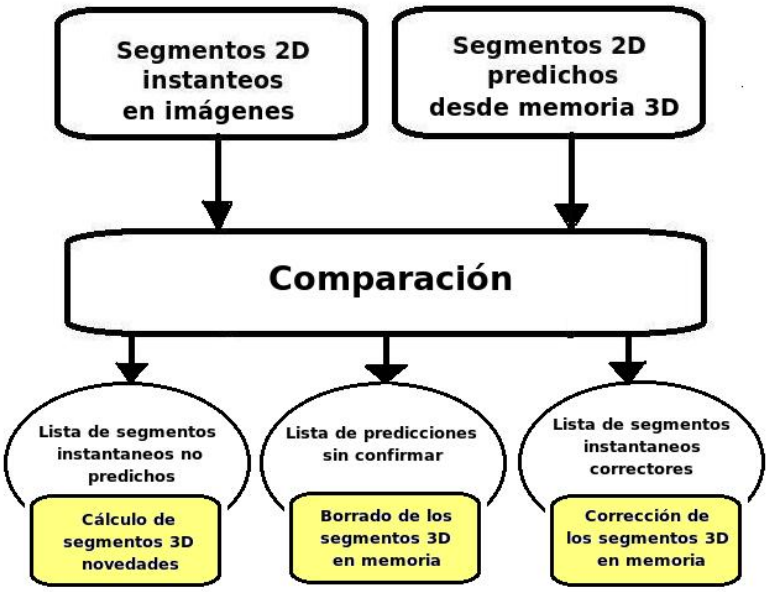

Figura 4.6: Comparación entre segmentos instantáneas y predicciones

El conjunto de segmentos predichos por el sistema se compara con aquellos segmentos instantáneos obtenidos por la transformada de Hough, como resultado de esta comparación obtenemos varios grupos de segmentos, tal y como muestra la figura 4.6. Por un lado, las novedades no predichas que deberán ser trianguladas, por otro lado aquellas predicciones que se han refutado y que provocan que se eliminen de memoria los segmentos 3D que las generaron. Por último, están las predicciones confirmadas pero que no coinciden con el segmento real con exactitud, este conjunto de segmentos corregirán a los segmentos 3D memorizados. Aquellas predicciones que coincidieron de forma exacta con los segmentos reales ahorran tiempo de cómputo en el cálculo de la posición de segmentos ya memorizados.

4.3. Reconstrucción instantánea con segmentos 3D

El sistema de reconstrucción 3D realiza una reconstrucción de aquello que el robot tiene ahora delante de sus cámaras. En esta sección se detallará el funcionamiento del algoritmo de búsqueda de segmentos homólogos y la triangulación de las novedades obtenidas en la etapa anterior.

El sistema de procesamiento 2D nos devuelve un conjunto de segmentos que deben ser situados en el espacio 3D. A partir de este momento, se pone en marcha el algoritmo de reconstrucción 3D para calcular la posición de estos segmentos en el espacio tridimensional basado en la triangulación entre las dos cámaras del par.

El mecanismo seguido se basa en lo propuesto por Gonzalo Abella [Abella Dago, 2009], con mejoras añadidas para obtener un algoritmo más robusto. Gonzalo propone la triangulación de cuatro puntos del segmento, sus dos puntos extremos, y otros dos puntos intermedios, para su verificación.

El algoritmo clásico de reconstrucción de puntos 3D consiste en, una vez tenemos el píxel perteneciente a la proyección en el plano imagen del punto 3D en la cámara maestra, obtener el píxel homólogo en la otra cámara. Roberto Calvo [Calvo Palomino, 2008] en su proyecto usó una serie de restricciones, acotando la búsqueda a la línea epipolar. Esta recta se calcula proyectando en la cámara esclava la recta de retro-proyección del píxel que deseamos triangular de la cámara maestra. Para acotar aún más la búsqueda, sólo se comprueban aquellos píxeles de la imagen auxiliar que sean borde. Por último, una vez se tienen los dos píxeles homólogos, hay que intersectar las rectas de retro-proyección de ambos píxeles. Haciendo esta triangulación obtenemos un punto en el espacio tridimensional.

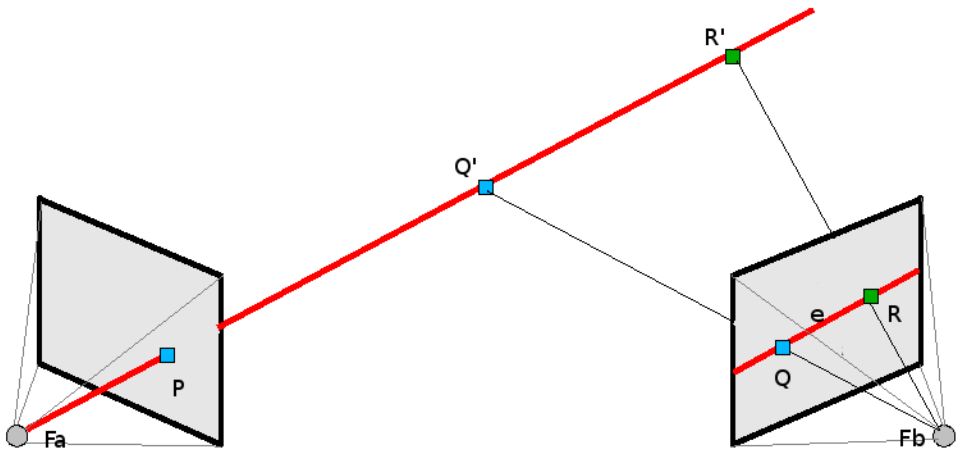

Figura 4.7: Proyección de la linea epipolar, donde se buscará el píxel homólogo

En nuestro algoritmo optamos por seleccionar tan solo tres puntos intermedios del segmento. Esto reduce las posibilidades de que en la búsqueda del píxel homólogo en la otra imagen dicho punto no tenga una proyección en el plano imagen de la cámara auxiliar, haciendo imposible su triangulación. Una vez obtenidos los tres píxeles homólogos en la imagen auxiliar comprobamos que todos pertenecen al mismo segmento 2D en la imagen, comparando estos píxeles con el conjunto de segmentos obtenidos mediante la transformada de Hough en la imagen de la cámara esclava.

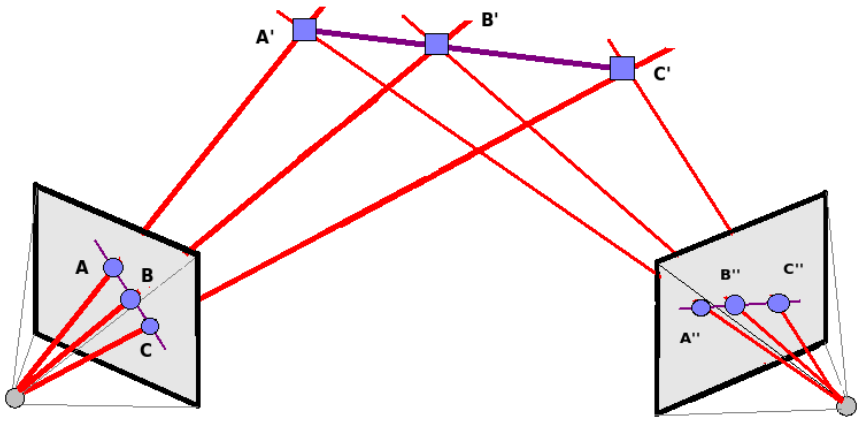

Figura 4.8: Triangulación del mini-segmento

Mediante la obtención de estos tres puntos en el espacio tridimensional tan solo obtendríamos la posición de un *mini-segmento*, pero no el segmento en su totalidad. Para obtener el segmento que estamos viendo en la cámara maestra en su totalidad debemos hallar la intersección entre la recta definida por el *mini-segmento* y las rectas de retro-proyección de los extremos del segmento en el plano imagen. En la figura 4.9 se aprecia cómo las rectas de retro-proyección *r1* y *r2*, se intersectan con la recta definida por el *mini-segmento*, *r3*, obteniéndose los puntos que definen el segmento en el espacio tridimensional, P y Q.

Una vez hemos obtenido el conjunto de segmentos en el espacio tridimensional es necesario hacer un post-tratamiento, previo a su incorporación a memoria permanente. Este post-tratamiento 3D similar al post-tratamiento de los segmentos 2D, consiste en comparar la posición relativa entre los segmentos, su orientación y su proximidad, para detectar aquellos segmentos reales que hayan podido ser expresados mediante dos segmentos 3D. Como resultado obtenemos un conjunto de segmentos en 3D definidos por un sistema de coordenadas solidario con la base del robot, puesto que los parámetros extrínsecos de las cámaras están definidos respecto de la base del robot.

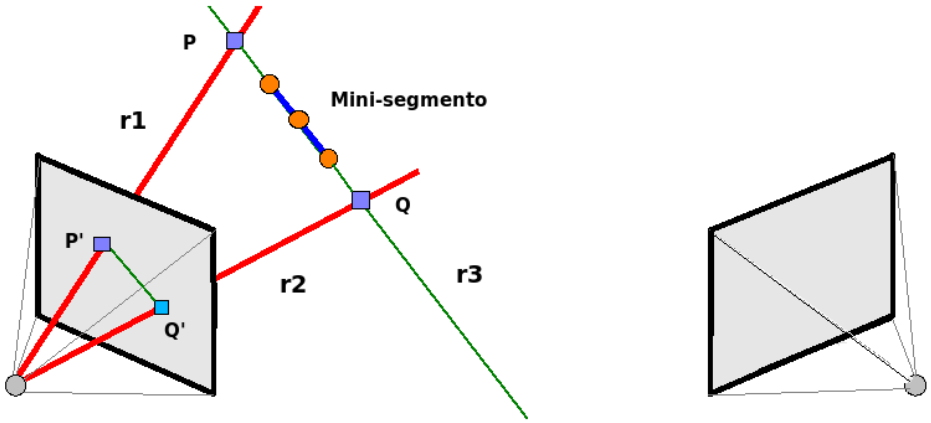

Figura 4.9: Obtención del segmento completo, a partir del mini-segmento

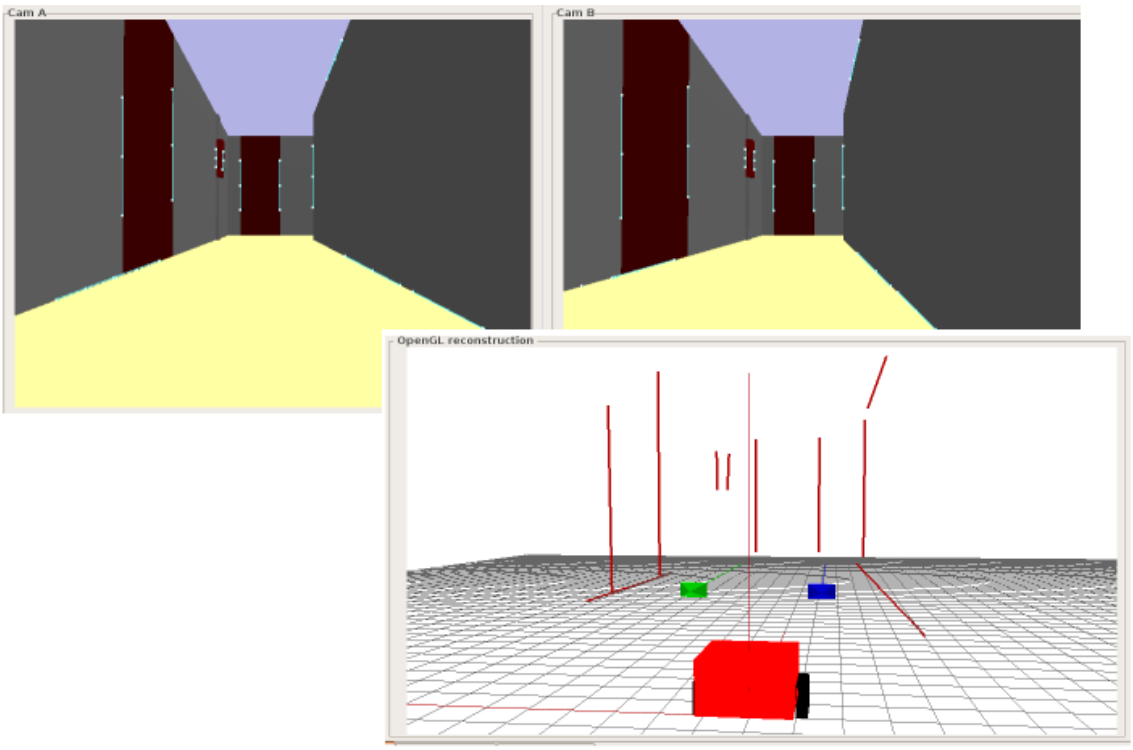

Figura 4.10: Visualización de los segmentos 3D instantaneos

El resultado de la obtención de los segmentos instantáneos de una escena se puede ver en la figura 4.10. En ella se aprecian los mini-segmentos calculados en la imagen de la cámara maestra y sus homólogos en la cámara auxiliar, además se puede ver la representación de estos segmentos con OpenGL en 3D.

4.3.1. Modelo geométrico

El modelo geométrico descrito a continuación permite al sistema mantener caracterizadas el par estereo de cámaras del robot y almacenar los objetos 3D en coordenadas absolutas de forma que el movimiento de la base del robot no afecte a su posición.

Para realizar los cambios de base entre el sistema de referencia relativo al robot y las coordenadas absolutas del mundo hemos desarrollado un modelo geométrico que digiere la posición de los dos cuellos mecánicos y la posición del robot en el mundo.

El modelo geométrico utiliza cuatro sistemas de referencia. El primero sitúa el origen de coordenadas a ras de suelo en un punto intermedio entre las dos cámaras, su función principal es la de devolver los parámetros extrínsecos de las cámaras cuando el cuello mecánico cambia su posición. El segundo sitúa el origen de coordenadas en el mundo simulado. Éste nos expresa la posición del robot y su orientación en cada momento y nos permite situar los objetos identificados en el mundo sin perder la referencia por el movimiento del robot.

Los dos sistemas de referencia restantes sitúan su origen en cada una de las cámaras. Estos son utilizados para cálculos intermedios, permiten tener en cuenta el movimiento de los cuellos mecánicos para la modificación de los parámetros de las cámaras, y realizar una correcta triangulación de los segmentos. La figura 4.11 muestra dónde tiene su origen los diferentes sistemas de referencia.

El cambio de base entre los diferentes sistemas de referencia se ha implementado mediante la construcción de un conjunto de matrices homogéneas de *rotación* y *translación*. Mediante la sucesiva multiplicación de estas matrices RT podemos obtener las coordenadas espaciales según los diferentes sistemas de referencia. Para las operaciones de matrices se ha utilizado la librería matemática *GSL*.

Cuando deseamos hacer un cambio de base, tenemos un punto *P0(X,Y,Z)* expresado en un sistema de referencia determinado, y debemos realizar una multiplicación de matrices para expresar este mismo punto mediante el nuevo sistema de coordenadas, obteniéndose un nuevo punto *P0'(X',Y',Z')*.

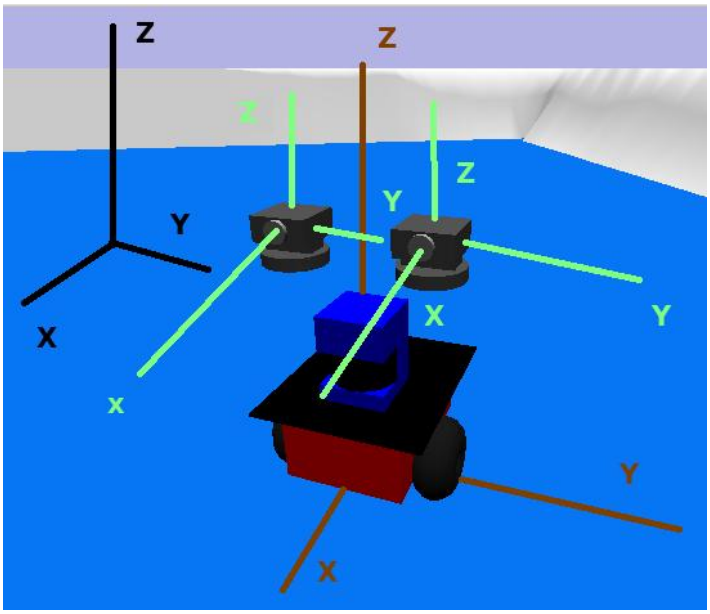

Figura 4.11: Sistemas de referencia utilizados por el sistema

Cambio de base entre coordenadas relativas y absolutas

La siguiente expresión tiene en cuenta la posición del robot en el mundo expresada por los valores de $Position_x$ y $Position_y$, y la orientación del robot que se relaciona con el ángulo θ. Estos valores son ofrecidos por los *encoders* del robot a través de las variables de JDErobot Robot[0] y Robot[1]. Con ellos se obtiene el valor de las coordenadas X e Y, respectivamente. Del mismo modo, el valor de Robot[2] nos devuelve el ángulo que forma el frontal del robot con el eje X absoluto.

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ H' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & Position_x \\ \sin\theta & \cos\theta & 0 & Position_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} \tag{4.1}$$

La reconstrucción 3D con segmentos instantáneos se lleva a cabo con el sistema de referencia solidario con el robot, pero para evitar que el movimiento del robot afecte a la posición de los segmentos es necesario memorizarlos según unas coordenadas absolutas.

Para la obtención de la matriz RT de coordenadas absolutas a relativas basta con el cálculo de las matrices inversas explicadas anteriormente. La ecuación 4.2, muestra las matrices utilizadas para expresar un punto en coordenadas absolutas, según el sistema de referencia solidario con el robot:

$$\begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & -sin\theta * Position_y - \cos\theta * Position_x \\ -\sin\theta & \cos\theta & 0 & sin\theta * Position_x - \cos\theta * Position_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \\ H' \end{bmatrix} \quad (4.2)$$

Cambio de base de coordenadas relativas a la cámara a relativas al robot

Por último, las siguientes matrices tienen en cuenta la posición de la cámara respecto al sistema de referencia situado en el robot, y los valores de *pan* y de *tilt*, de los cuellos mecánicos de las cámaras. Los valores de *pan* y de *tilt* son ofrecidos por los interfaces *ptenconders* y *ptencodersB*, que nos dan la información referente a los ángulos de inclinación de ambas cámaras. En nuestra expresión la relación es directa, α se relaciona con *pan* y β con *tilt*. En este caso, los valores de $Camera_x$,$Camera_y$ y $Camera_z$, corresponden con las coordenadas donde se sitúa la cámara respecto del sistema de referencia centrado en el robot.

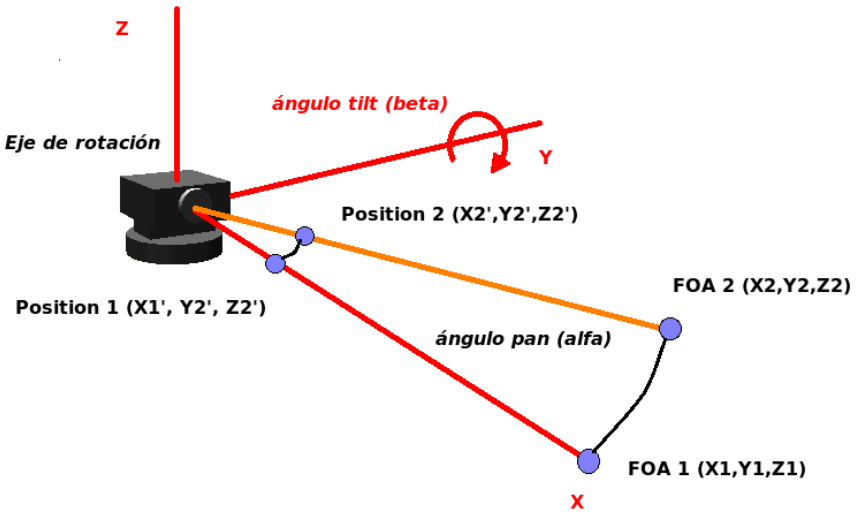

Figura 4.12: Esquema del sistema de referencia situado en la cámara

La figura 4.12 muestra la obtención de las variables de entrada para la matriz de cambio de base.

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ H' \end{bmatrix} = \begin{bmatrix} \cos\alpha * \cos\beta & -\sin\beta & \sin\alpha * \cos\beta & Camera_x \\ -\cos\alpha * \sin\beta & -\cos\beta & -\sin\alpha * \sin\beta & Camera_y \\ \sin\alpha & 0 & -\cos\alpha & Camera_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} \tag{4.3}$$

La construcción de este modelo geométrico ha sido vital para el correcto funcionamiento de múltiples partes del sistema. El movimiento de los cuellos mecánicos afecta a la reconstrucción de objetos si los parámetros extrínsecos de las cámaras no fueran modificados correctamente.

Con cada modificación de *pan* o de *tilt* debemos calcular la nueva posición del foco de atención para tener caracterizadas en todo momento las cámaras del par.

4.4. Memoria visual 3D

A continuación explicaremos las funcionalidades de la memoria 3D y de como ésta se acopla con las demás partes del sistema.

La memoria tridimensional de la que goza este sistema es quizá el componente más complejo y de mayor relevancia de todo el conjunto. Esta memoria de objetos ha sido pensada para proporcionar la capacidad de reconstruir la realidad que rodea al robot de forma fidedigna, sacando el mayor provecho de la autonomía de movimientos que nos ofrece un robot móvil y sus cámaras.

La memoria 3D no es más que un conjunto de listas dinámicas, en las que se almacenan los diferentes objetos para su posterior representación. En ellas se pueden guardar segmentos 3D o estímulos estructurados, guardando todos ellos relación entre sí. Por ejemplo, la eliminación de un segmento podría llevar a la eliminación de una puerta.

La memoria de segmentos 3D cuenta con funciones para la incorporación de los segmentos 3D instantáneos obtenidos en la etapa anterior a memoria. Para ello se ayuda de una memoria caché de segmentos 3D cercanos.

Otra de las funcionalidades que se comentarán será la capacidad del sistema para generar predicciones de segmentos guardados en memoria 3D en el plano imagen de las cámaras. De esta manera que será necesario realizar los cálculos para hallar la posición de aquellos segmentos ya triangulados.

Además, la memoria 3D cuenta con la capacidad de almacenar estímulos estructurados como puerta, pasillo, pared o cuadrado. Estos objetos son guardados en la memoria después

de confirmar su existencia mediante la información captada por las cámaras. El sistema permite visualizar todos estos objetos mediante la librería OpenGL.

4.4.1. Inserción de segmentos instantáneos

Una vez el sistema de reconstrucción de segmentos 3D nos ofrece la lista de novedades entra en juego la memoria 3D. La memoria de segmentos guarda la estructura de datos *Segmento3D*, compuesta por un punto de inicio y otro de final del segmento, un valor de *incertidumbre* que refleja la distancia euclídea entre el punto medio del segmento triangulado y el punto donde se encontraba el robot cuando fue triangulado el segmento, una variable *Booleana* que indica si el segmento se encuentra en memoria caché, una variable *Booleana* que indica si el segmento define alguna puerta y una lista de punteros a objetos puerta, que apuntará a aquellos objetos puerta que son definidos por el segmento.

```
struct Segment3D{
  HPoint3D  start;
  HPoint3D  end;
  float     uncertainty;
  bool      incache;
  bool      doorassign;
  list<list<Door>::iterator> doors;
};
```

El proceso de incorporación a memoria de los nuevos segmentos consiste en comparar uno a uno los segmentos instantáneos calculados con los segmentos ya almacenados, de modo que si dos segmentos están próximos en el espacio y tienen la misma orientación se fusionarán en un nuevo segmento que tenga en cuenta las bondades de sus predecesores, en caso contrario, se incorporarán como auténticas novedades a la memoria de segmentos 3D.

El proceso de fusión de segmentos 3D es el siguiente. Tomamos el segmento con menor incertidumbre como segmento principal. Dicho segmento, por haber sido triangulado a menor distancia que su pareja, definirá la posición y la orientación del nuevo segmento segmento fusionado. Por otro lado, tomaremos los dos vértices más alejados del conjunto de cuatro vértices pertenecientes a los segmentos que se van a fusionar, se calcularán los planos que pasan por estos puntos y son perpendiculares al segmento que contiene los puntos. El segmento final será una prolongación de los extremos del segmento principal

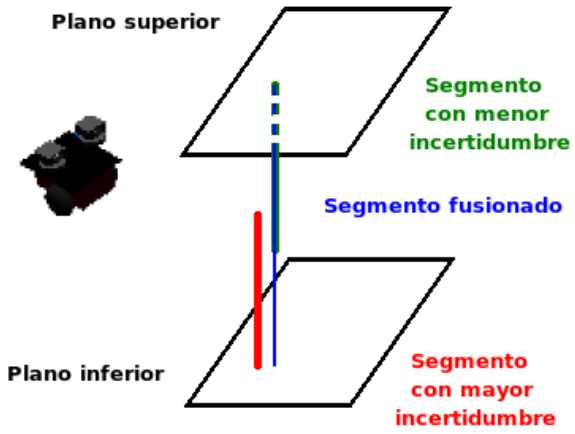

Figura 4.13: Proceso de fusión de segmentos 3D

hasta los puntos de corte con los planos descritos anteriormente. El proceso de fusión puede verse en la figura 4.13 donde el segmento resultante es la prolongación del segmento con menor incertidumbre. Hemos podido ver un segmento de lejos con una determinada longitud y una vez nos acercamos a él, vemos tan solo una fracción del mismo, pero podemos realizar una triangulación más precisa sobre su posición. Por tanto debemos conservar la longitud estimada en el vistazo lejano y la posición estimada en el vistazo cercano.

Un caso bastante común cuando se ejecuta el sistema en el mundo departamental es la detección de segmentos pertenecientes a las puertas. Dicho mundo contiene multitud de puertas, el robot detecta con facilidad los segmentos verticales que componen el marco de la puerta. Cuando se sitúa lejos de la puerta la observa en su totalidad, y aunque no es posible precisar la posición exacta de estos segmentos, si se puede calcular la longitud de estos segmentos, según el robot se vaya acercando empezará a ver tan solo parte de la puerta. Esto permite una reconstrucción más fiable de los segmentos por estar más próximos, pero debemos respetar su longitud.

Para que el proceso de comparación entre los segmentos instantáneos y los segmentos memorizados se lleve a cabo de una manera eficiente, se ha implementado una memoria caché de segmentos. La memoria caché la componen aquellos segmentos cercanos al robot, y se ha implementado como una lista dinámica que guarda punteros dirigidos a ciertos segmentos guardados en la memoria de segmentos 3D. Para cada segmento que se incorpora a memoria, se comprueba si es cercano al robot y se añade a la caché en caso afirmativo. Además, cada cierto número de ejecuciones del algoritmo, se recalcula la memoria caché al completo, comprobando todos los segmentos almacenados en la memoria 3D y comprobando que estén a una determinada distancia cercana a la base del robot.

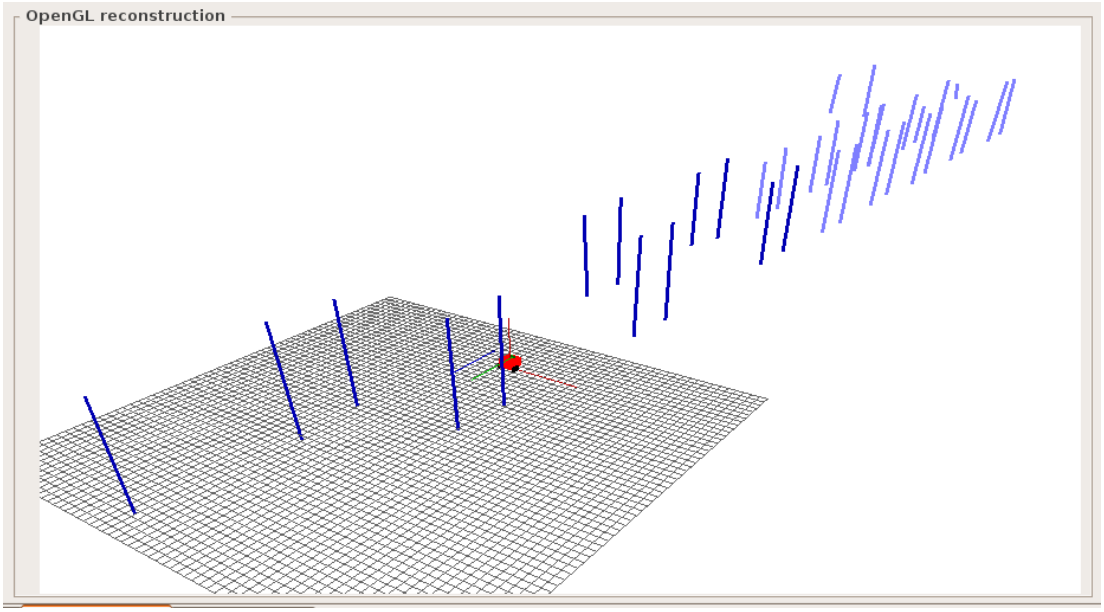

Figura 4.14: Segmentos almacenados en memoria, en tono oscuro aquellos guardados en la caché

De este modo se incorporan aquellos segmentos que en su momento fueron triangulados lejos del robot, pero con el paso del tiempo y el movimiento del robot, es posible que se encuentren cerca de él.

La memoria caché evita comparaciones de todos los elementos de la memoria con todos los segmentos instantáneos, reduciendo con ello la cantidad de cómputo y acelerando el funcionamiento de la memoria visual 3D.

4.4.2. Predicciones, eliminación y corrección de segmentos

La triangulación de un segmento mediante un par estéreo puede resultar complicada, sobre todo sin una calibración perfecta de las cámaras. En esta sección se explican los procesos de depuración y eliminación de aquellos segmentos que pudieron ser mal situados en el espacio.

Tal y como se explicó en secciones anteriores, el sistema de procesamiento 2D devuelve varios conjuntos de segmentos. (a) aquellos segmentos instantáneos que no han sido predichos por la memoria de segmentos 3D, y que deben incorporarse a memoria como novedades; (b) el conjunto de predicciones de segmentos 3D que deben encontrarse en la imagen pero que quedan insatisfechas; y (c) aquellas predicciones que se cumplen, corrigiendo el segmento almacenado en memoria o ahorrando tiempo de cómputo al sistema.

El conjunto de segmentos predichos en la imagen se genera a partir de los segmentos almacenados en memoria 3D, estos segmentos se proyectarán en el plano imagen si son visibles.

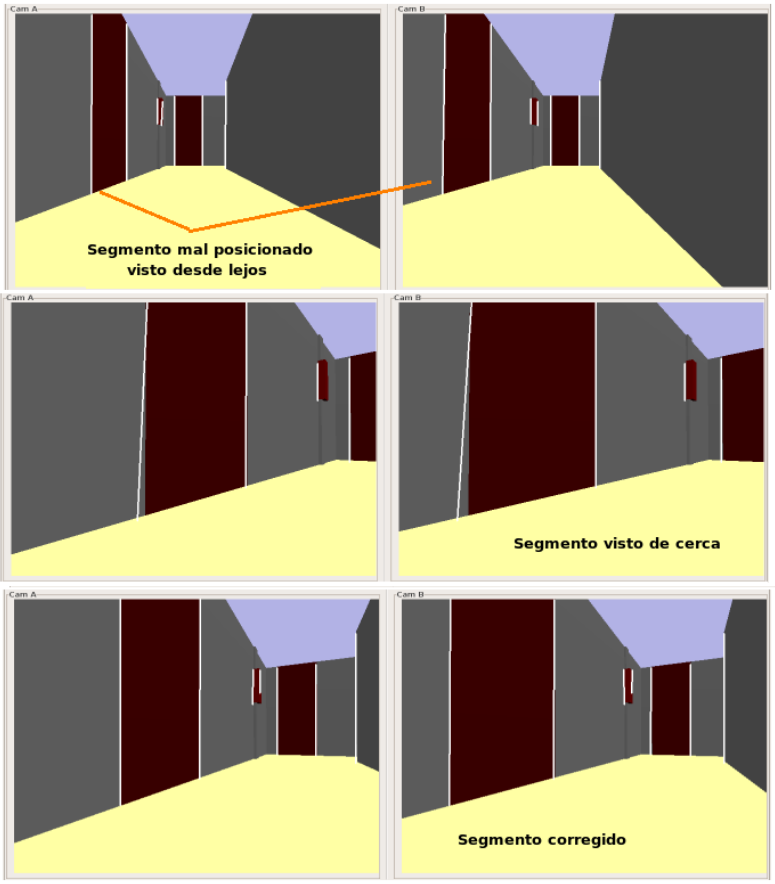

Figura 4.15: La predicción se proyecta sobre la imagen, posteriormente se corrigen los segmentos

Para que un segmento predicho sea 'igual' a otro presente en el plano imagen se lleva a cabo una comparación muy estricta, de modo que si el segmento no se identifica con la predicción, se calculará un nuevo segmento próximo al que generó la predicción. El nuevo segmento 3D instantáneo corregirá al segmento memorizado, obteniéndose una mejor triangulación del segmento real. En la figura 4.15 se aprecia cómo se proyecta la memoria sobre la imagen (líneas blancas), la secuencia de imágenes a medida que el robot se acerca a la puerta, ilustra la corrección de segmentos cuando se está más próximo a ellos.

El proceso de eliminación comienza con una nueva comparación de los segmentos que están en memoria caché con el conjunto de segmentos que nos devolvió el sistema de procesamiento 2D. Esta comparación es más tolerante, es decir, debe ser un segmento que

no sea confirmado por ningún segmento próximo en la imagen captada por las cámaras, así se evita eliminar segmentos susceptibles de ser corregidos. Aquellos segmentos que no tienen confirmación se eliminan de la memoria de forma permanente. Estos segmentos espúreos pueden ser el resultado de una mala triangulación de un segmento lejano, que cuando nos acercamos a él comprobamos que no se corresponde con ningún segmento del escenario.

El proceso de eliminación hace que el sistema sea sensible a las oclusiones. En escenarios estáticos cómo el que nos ofrece el simulador no surge esta problemática, que es más propia de escenarios dinámicos, como el mundo real. Queda para extensiones futuras de este proyecto.

En la figura 4.16 se aprecian las predicciones realizadas como líneas blancas, y cómo estas pueden encajar de forma precisa con un segmento real en la imagen, estar próximo a un segmento o por el contrario no tener ningún segmento que lo confirme, lo que lleva a la eliminación del segmento.

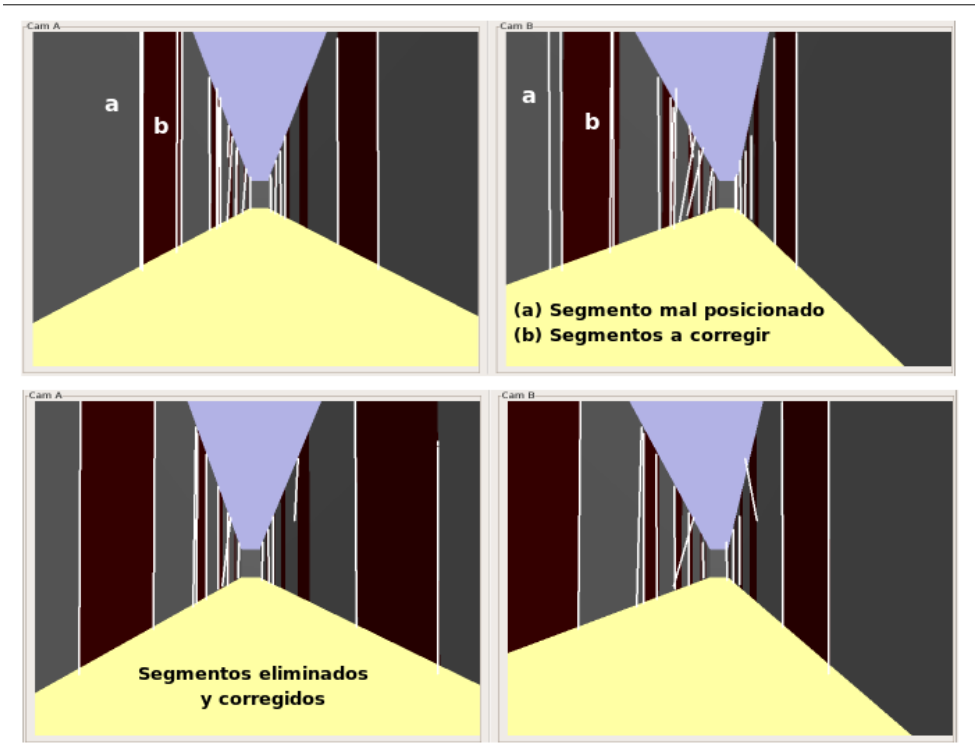

Figura 4.16: Secuencia de eliminación de segmentos en memoria

Adicionalmente a estos mecanismos, la triangulación de segmentos lejanos puede provocar que se generen segmentos erróneos y mal situados en el espacio. Una posterior triangulación de la misma zona podría generar un nuevo segmento para el mismo segmento

real sin llevar a cabo una fusión entre el segmento mal triangulado y el nuevo segmento. Para solucionar este problema se ha implementado una función de mantenimiento de la memoria que compara periodicamente entre sí los segmentos almacenados buscando la posibilidad de hacer nuevas fusiones.

4.4.3. Estímulos estructurados e hipótesis perceptivas

Hasta ahora teníamos una memoria de segmentos 3D. A partir de estos segmentos deseamos crear una memoria de objetos 3D que sea más compacta, con primitivas más abstractas y que expliquen mejor el mundo reconstruido.

Mediante técnicas abductivas, realizamos hipótesis perceptivas desde los segmentos 3D almacenados, proponemos la existencia de ciertos objetos 3D en el espacio que se confirman o refutan comprobando si las predicciones que esa hipótesis conlleva se satisfacen en las imágenes instantáneas.

De este modo primero predecimos que un objeto está en el espacio, esta hipótesis debe ser verificada a continuación, en caso contrario se elimina el objeto. Este método es especialmente útil para estímulo complejos que no caben completamente en una imagen.

Los estímulos estructurados tienen sus características definidas *a priori*, y pueden ser puertas, paredes, pasillos o cuadrados.

En ocasiones, realizaremos hipótesis analizando los segmentos almacenados, como el caso del objeto puerta. Otras veces se generarán nuevas hipótesis bajo el análisis de otros estímulos estructurados, como la generación de paredes y pasillos a partir de las puertas.

Puertas

Una puerta se define como dos segmentos 3D verticales equidistantes entre sí una distancia determinada, y a su vez que tengan una determinada altura, entre estos dos segmentos se hipotetiza que existe una puerta. Cada vez que un par de segmentos incluidos en memoria caché generan una puerta, ésta es incluida dentro de una lista de puertas sin confirmar. Los estímulos estructurados han de ser validados por medio de las imágenes captadas por las cámaras. Tan pronto se añade una puerta en esta lista de elementos sin confirmar el sistema calcula, ayudándose de la función *project* de Progeo, un rectángulo en la imagen donde debería encontrarse la puerta. Dentro de este rectángulo se comprueban varios píxeles de la parte central, si dichos píxeles pasan el filtro de color se confirma la

existencia de una puerta. En caso contrario la puerta será eliminada de forma permanente de la memoria 3D.

La memoria de puertas es una lista dinámica que almacena el tipo de dato *door*, que guarda la posición de la puerta, su orientación (el ángulo que forma el plano longitudinal de la puerta con el eje X según coordenadas absolutas), el tamaño de la puerta, un valor *booleano* que nos indicará si la puerta ha sido validada, y dos punteros dirigidos a los dos segmentos que definen la puerta. De esta manera, las memorias de segmentos y puertas quedan acopladas y las modificaciones en una de ellas afectan a la otra.

```
struct Door{
   HPoint3D position;
   float    size;
   float    orientation;
   bool     validated;
   list<memory3D::Segment3D>::iterator sleft;
   list<memory3D::Segment3D>::iterator sright;
};
```

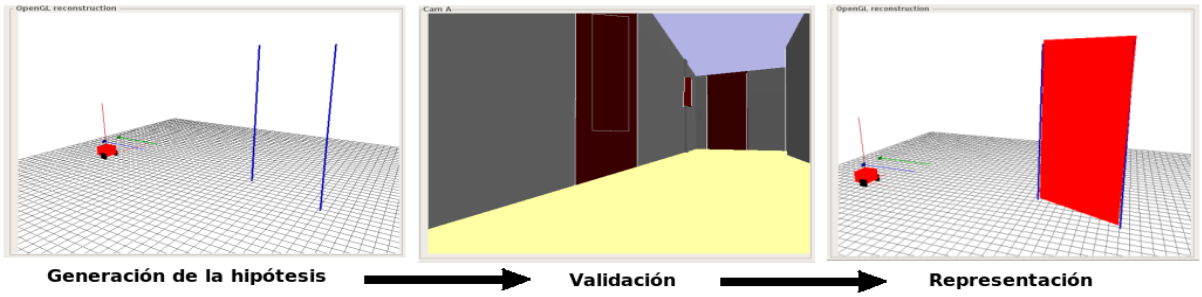

Figura 4.17: Proceso de generación del estimulo estructurado *puerta*

En la figura 4.17 se aprecia cómo a partir de dos segmentos se hipotetiza una puerta que a continuación debe ser validada.

Pared

La hipótesis pared surge de la idea de que cada puerta almacenada en memoria debe estar integrada en una pared. En principio no sabemos el tamaño de esta pared, pero si mediante el análisis de la memoria de puertas encontramos que un conjunto de ellas poseen

la misma orientación y están alineadas, podremos concluir que todas ellas pertenecen a la misma pared.

Este estímulo estructurado no se confirma. La existencia de una puerta permite inferir el concepto pared.

Pasillo

Un pasillo es el espacio entre dos paredes. La hipótesis pasillo se genera a partir de dos filas de puertas paralelas y separadas cierta distancia como muestra la figura 4.18. La búsqueda de pasillos se lleva a cabo cada cierto numero de iteraciones para no sobrecargar el sistema con cálculos reiterativos.

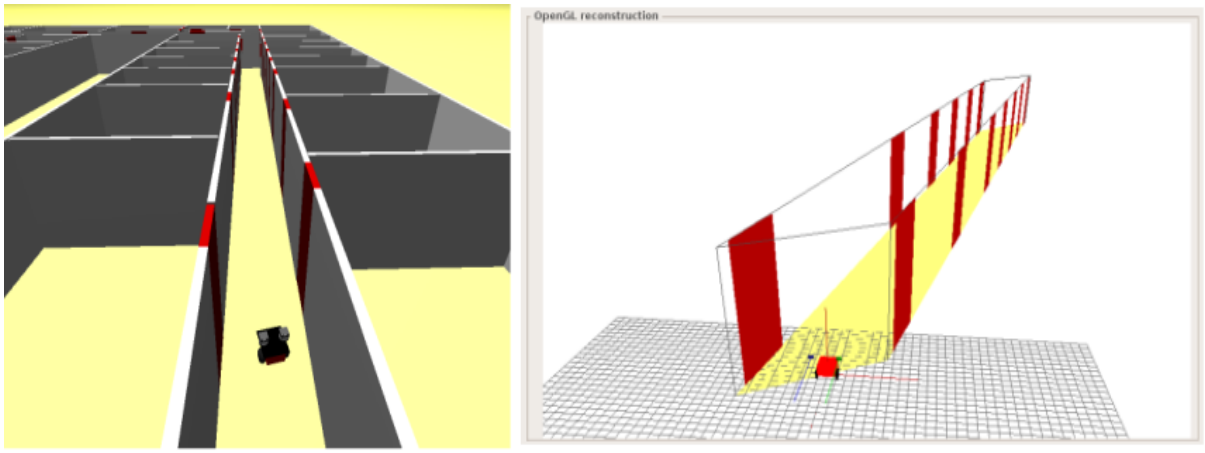

Figura 4.18: Hipótesis pasillo a partir de dos paredes paralelas

En ocasiones un pasillo puede tener tan solo una puerta en una de sus paredes, en estos casos consideramos que la puerta debe pertenecer a una pared y delante suya debe haber un pasillo, de este modo creamos la hipótesis que se puede ver en la figura 4.19.

Una vez tenemos el grupo de pasillos generados por las puertas proponemos que estos pasillos deben intersectarse en algún punto. Hemos prestado mucha atención a la intersección de los pasillos atendiendo a su orientación, con el objetivo de realizar una buena representación de las paredes pertenecientes a estos pasillos. En la figura 4.20 se muestra cuáles deben ser los puntos de unión entre las paredes.

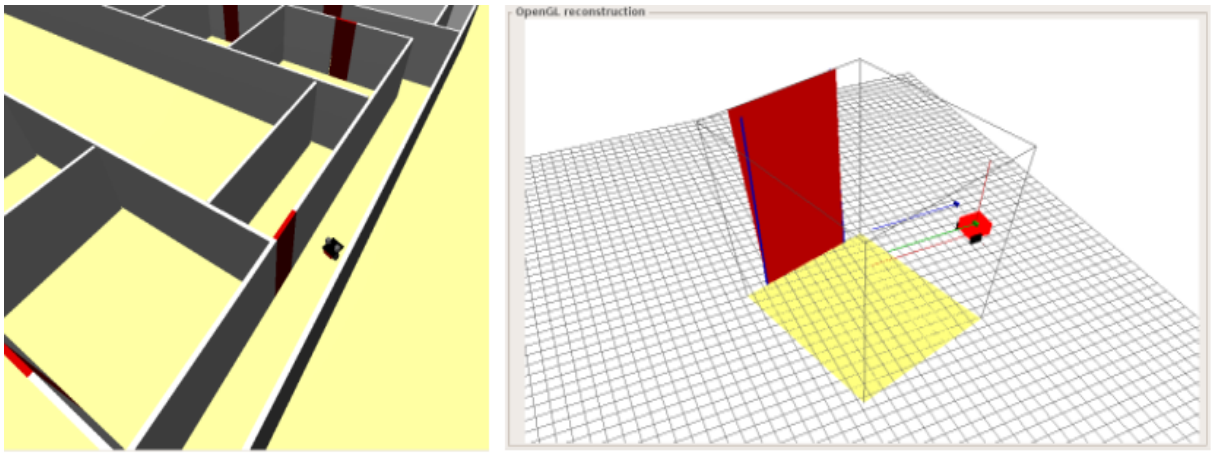

Figura 4.19: Hipótesis pasillo a partir de una única pared

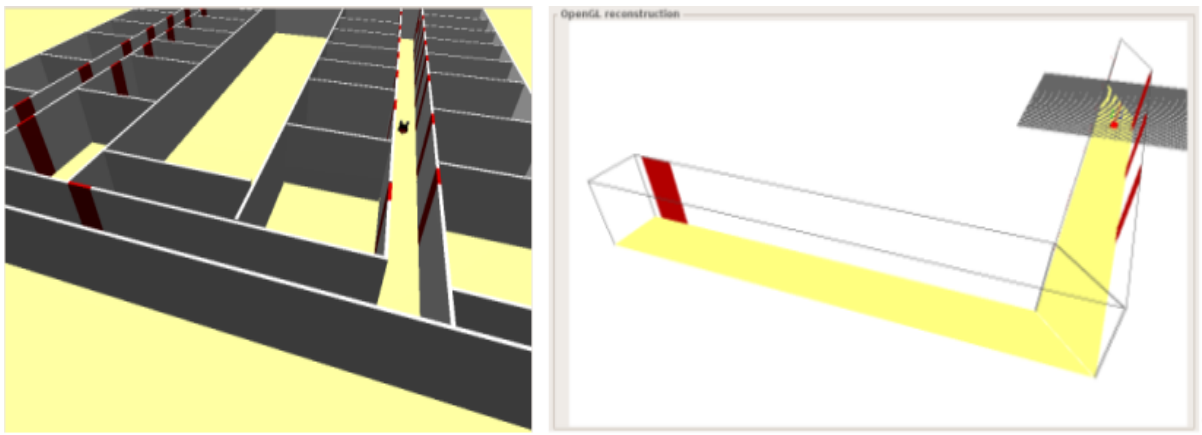

Figura 4.20: Intersección de dos pasillos

Cuadrado

Otro estímulo estructurado que hemos manejado es el cuadrado. Para crear el estímulo cuadrado partimos de dos segmentos rectos 3D guardados en la memoria, con dimensiones similares y que forman entre si un ángulo de 90º. Una vez identificados estos dos segmentos, se calculan las dos aristas restantes de un hipotético cuadrado y su cuarto vértice. El sistema tendrá una lista de hipótesis cuadrado sin confirmar. Con cada nueva ejecución del algoritmo de reconstrucción 3D se analizan los segmentos que contiene la memoria y se intentan confirmar todos los cuadrados posibles. Con aquellos estímulos estructurados cuadrado se creará una nueva lista.

En la figura 4.21 se puede ver un cubo que oculta parte del cuadrado negro. El sistema después de reconstruir la escena con segmentos 3D rectos, analiza estos segmentos en busca de indicios de un posible cuadrado, se generan multitud de hipótesis. Para que el conjunto de cuadrados obtenido se confirme debemos desplazar al robot a un nuevo punto desde donde sus cámaras puedan reconstruir aquellos segmentos que nos permitan obtener el conjunto de estímulos estructurados. Las hipótesis sin confirmar están representadas en tono grisáceo, mientras que aquellas con sus cuatro aristas confirmadas se representan de un tono amarillo intenso.

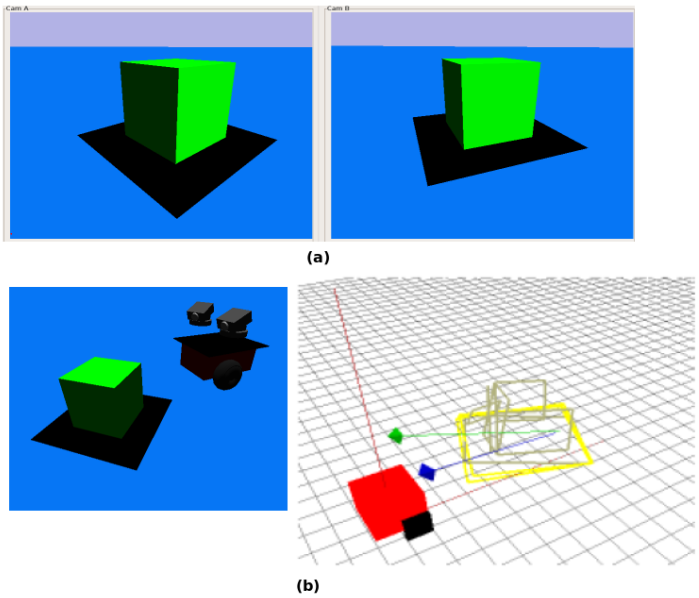

Figura 4.21: (a) Visión desde el par estéreo, (b) escenario y reconstrucción con cuadrados

Cuando el sistema realiza la reconstrucción valiéndose del estímulo cuadrado no genera predicciones sobre el plano imagen ni tampoco elimina segmentos, puesto que de lo contrario

no se podría realizar una confirmación del cuadrado completo debido a las oclusiones.

Capítulo 5

Experimentos

Una vez descrito el diseño y la implementación del sistema de reconstrucción 3D, se expondrá en este cápitulo una serie de experimentos especialmente seleccionados para validar la solución desarrollada y sus diferentes partes. También permiten analizar con cierto detalle el funcionamiento del sistema o de alguno de sus subcomponentes. Además hemos realizado un estudio valorando ciertas alternativas en el diseño, para ayudar a ajustar lo mejor posible el algoritmo final, como por ejemplo ignorar los segmentos horizontales.

5.1. Reconstrucción de mundo departamental

Con este experimento queremos validar la totalidad del sistema, la detección de segmentos en la imagen, su triangulación en el espacio, su incorporación a memoria, y la reconstrucción de los pasillos del mundo. Para este experimento utilizamos el mundo departamental, donde situamos un robot Pioneer con un par estéreo. En este experimento teleoperamos al robot por los pasillos, mientras el sistema perceptivo creado realiza la reconstrucción tridimensional de aquello que ve en cada instante.

El sistema maneja mucho conocimiento ad hoc sobre el mundo para la reconstrucción con conceptos abstractos como puerta, pared, pasillo, etc. Tiene en cuenta, por ejemplo, que todos los pasillos se intersectan y que lo hacen en un ángulo de 90º.

En la figura 5.1 puede verse el mundo de partida en el simulador Gazebo. El resultado de una vuelta completa por los pasillos del mundo, es un mapa tridimensional de los pasillos como se muestra en la figura 5.2. Cualitativamente si comparamos las figuras 5.1 y 5.2 se ve el parecido entre la realidad y la estimación construida.

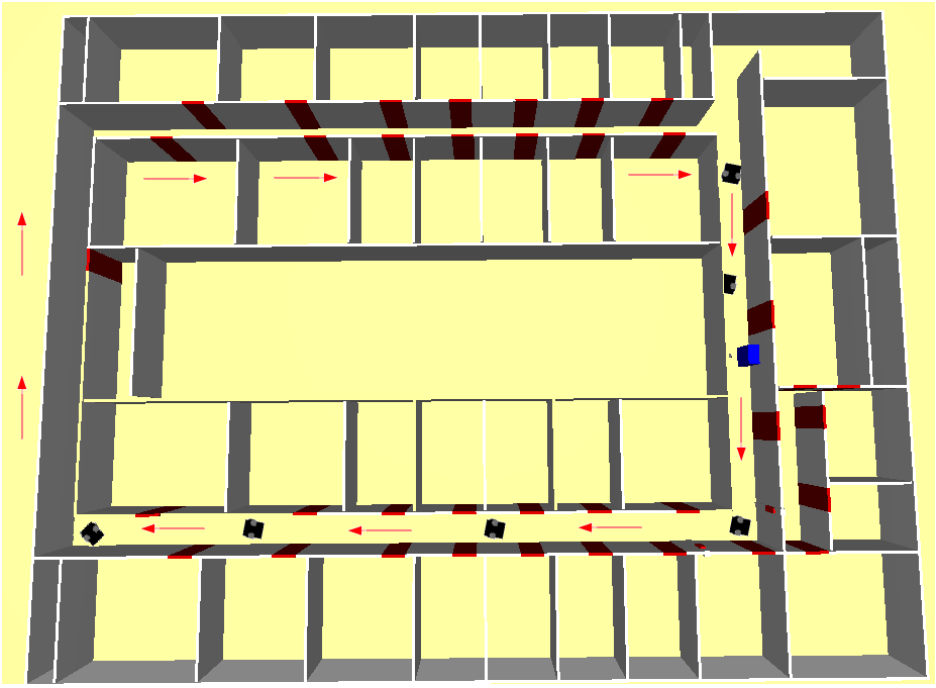

Figura 5.1: Mundo departamental y recorrido del robot por los pasillos

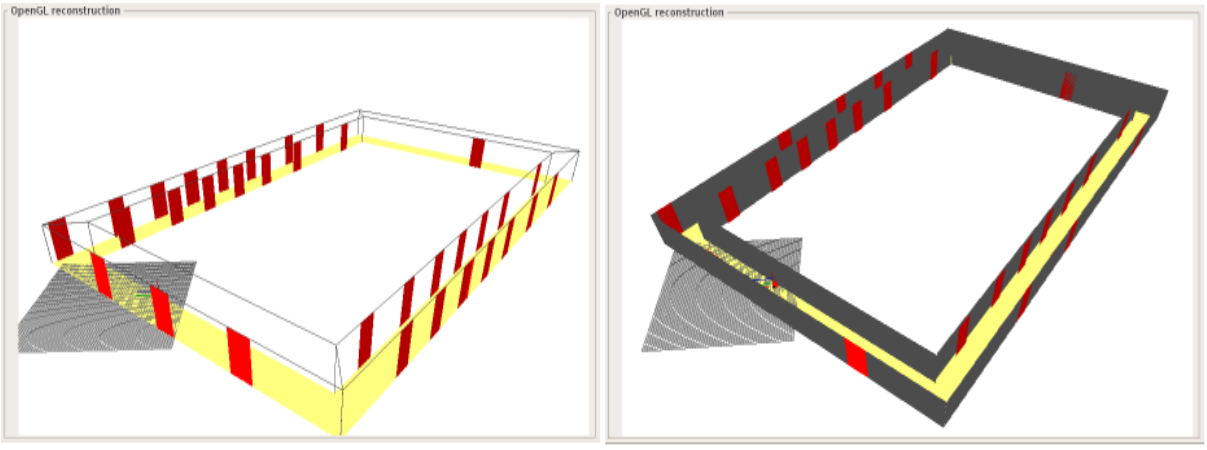

Figura 5.2: Reconstrucción completa de los pasillos del mundo departamental

Análisis de tiempos

El sistema de reconstrucción 3D montado sobre un robot está pensado para reproducir el entorno en tiempo real. Por ello el tiempo de ejecución de cada iteración es crucial. En cada ejecución el sistema predice y extrae los segmentos 2D del plano imagen, posiciona los segmentos 2D en el espacio tridimensional e incorpora los segmentos instantaneos a memoria y genera los estímulos estructurados. De todos estos pasos, el punto donde el algoritmo consume mayor tiempo de cómputo es en la triangulación de los segmentos, aproximadamente 100 milisegundos por cada 10 segmentos reconstruidos, y en concreto, en la búsqueda de puntos homólogos en la imagen de la cámara esclava. En la figura 5.3 se ve la relación entre el número de segmentos 2D en el plano imagen y el tiempo que consume el sistema en calcular la posición de todos ellos en el espacio 3D.

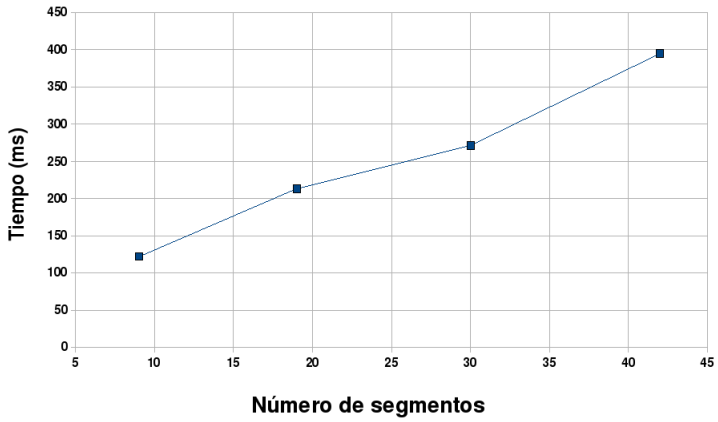

Figura 5.3: Relación entre el número de segmentos 3D instantaneos y el tiempo de ejecución

De aquí se deriva la importancia de las predicciones y de que estas ahorren el cálculo de segmentos 3D ya explicados.

La predicción y la generación de estímulos estructurados depende en gran medida del número de segmentos almacenados en la memoria visual, pero en nuestro experimento nunca tarda más de 5 milisegundos, gracias a la memoria caché implementada.

A raíz de estos resultados y teniendo en cuenta el número de segmentos que suele calcular el algoritmo en cada iteración, en los experimentos le hemos ajustado una frecuencia nominal de 500 milisegundos para que le diera tiempo a realizar todos los cálculos y realizar las tareas de visualización en el interfaz de usuario.

Otro experimento realizado consiste en comprobar cual es la velocidad máxima de navegación del robot por el mundo departamental sin que el sistema deje de representar

los elementos de los pasillos. Debemos tener en cuenta que el algoritmo de reconstrucción se ejecuta cada medio segundo, por lo que a medida que aumentamos la velocidad de navegación tenemos menos observaciones de una misma zona del mundo y por tanto perdemos precisión en la reconstrucción.

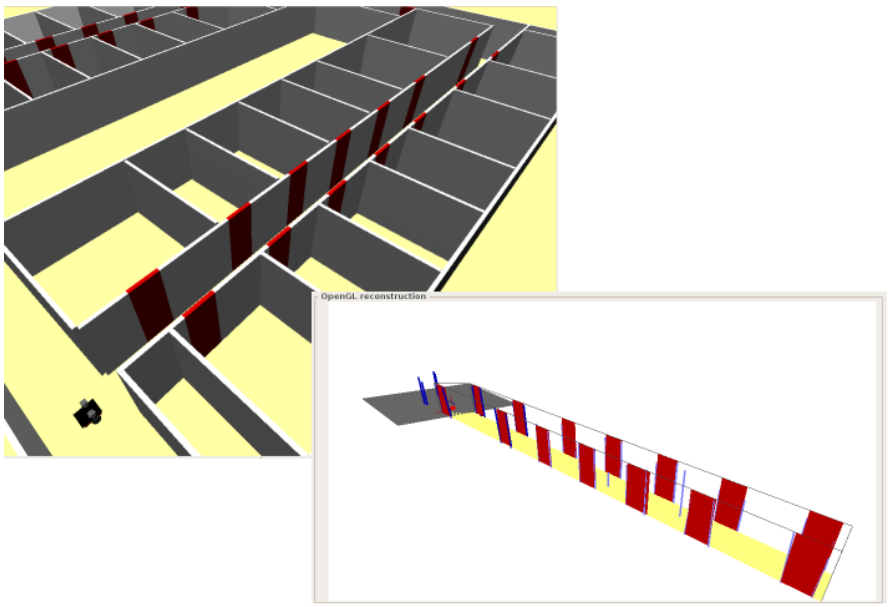

Figura 5.4: Pasillo escenario del experimento y resultado optimo de la reconstrucción 3D

Para esta prueba situamos al robot al principio de uno de los pasillos del mundo departamental, tal y como muestra la figura 5.4 en dicho pasillo existen 14 puertas. Como resultado de varias pruebas hemos determinado que cuando el robot navega por encima de los 2,7 m/s de velocidad constante el algoritmo pierde precisión situando las puertas y no es capaz de representar todas las puertas del pasillo.

Ésta velocidad es muy alta para robots reales, sería inseguro que el robot navege a ésta velocidad en interiores, por lo que el algoritmo perceptivo funcionará bien en robots reales, que suelen moverse a una velocidad máxima de 1 m/s.

Análisis de precisión espacial

La precisión con la que se sitúan las puertas es centimétrica, tal y como muestra el cuadro 5.1 que representa la posición original de cada puerta, la posición donde el sistema la reconstruye y el error en la posición.

Puerta	Coordenadas originales	Coordenadas reconstrucción	Error (cm)
1	(730,0, 2535,0, 0,0)	(730, 2535,4, 0,0)	0,4
2	(1120,0, 2535,0, 0,0)	(1118,2, 2535,1, 0,0)	1,8
3	(1510,0, 2535,0, 0,0)	(1509,8, 2535,1, 0,0)	0,2
4	(1930,0, 2470,0, 0,0)	(1930,3, 2470,4, 0,0)	0,5
5	(1800,0, 2220,0, 0,0)	(1801, 2220,6, 0,0)	1,2
6	(1930,0, 2170,0, 0,0)	(1930,6, 2169,3, 0,0)	0,9
7	(1800,0, 1920,0, 0,0)	(1800,8, 1920,2, 0,0)	0,8
8	(1930,0, 1920,0, 0,0)	(1932, 1916,8, 0,0)	3,8
9	(1800,0, 1690,0, 0,0)	(1801,4, 1690,9, 0,0)	1,7
10	(1930,0, 1690,0, 0,0)	(1931,5, 1688,8, 0,0)	1,9
11	(1800,0, 1460,0, 0,0)	(1800,4, 1460,6, 0,0)	0,7
12	(1930,0, 1460,0, 0,0)	(1930,3, 1462,6, 0,0)	2,6
13	(1800,0, 1220,0, 0,0)	(1800,5, 1220, 0,0)	0,5
14	(1930,0, 1220,0, 0,0)	(1930,3, 1220,9, 0,0)	0,9
15	(1800,0, 920,0, 0,0)	(1801,1, 918,8, 0,0)	1,6
16	(1930,0, 850,0, 0,0)	(1930,8, 850,7, 0,0)	1,1
17	(1800,0, 385,0, 0,0)	(1800,7, 388, 0,0)	3,1
18	(1930,0, 500,0, 0,0)	(1930,2, 501,2, 0,0)	1,2
19	(910,0, 135,0, 0,0)	(910,6, 135,2, 0,0)	0,6
20	(340,0, 490,0, 0,0)	(340,1, 493,4, 0,0)	3,4
21	(460,0, 380,0, 0,0)	(460,9, 379,9, 0,0)	0,9
22	(460,0, 930,0, 0,0)	(460,4, 928,5, 0,0)	1,6
23	(340,0, 860,0, 0,0)	(339,8, 862,3, 0,0)	2,3
24	(460,0, 1210,0, 0,0)	(463,2, 1213,9, 0,0)	5,0
25	(340,0, 1200,0, 0,0)	(340,6, 1199,9, 0,0)	0,6
26	(460,0, 1460,0, 0,0)	(460, 1458,8, 0,0)	1,2
27	(340,0, 1450,0, 0,0)	(340,3, 1451,6, 0,0)	1,6
28	(460,0, 1690,0, 0,0)	(461,7, 1692,2, 0,0)	2,8
29	(340,0, 1690,0, 0,0)	(340,8, 1689,1, 0,0)	1,2
30	(460,0, 1930,0, 0,0)	(461,2, 1931,9, 0,0)	2,2
31	(340,0, 1930,0, 0,0)	(341,7, 1930,4, 0,0)	1,7
32	(460,0, 2220,0, 0,0)	(460,6, 2219, 0,0)	1,2
33	(340,0, 2180,0, 0,0)	(341,3, 2178,6, 0,0)	1,9
		Error medio	1.62

Cuadro 5.1: Precisión del posicionamiento de puertas

Según muestra el cuadro 5.1 hemos comparado la posición estimada en X y en Y con la posición real en el simulador para las 33 puertas que contiene el mundo departamental y el error medio es de 1,62 centímetros. A la luz de estos datos podemos asegurar que el sistema realiza una representación fidedigna de los pasillos con precisión centimétrica.

5.2. Reconstrucción de sólidos sencillos

En este experimento deseamos comprobar la precisión con la que se sitúan los segmentos 3D triangulados. Para ello, llevamos a cabo una reconstrucción mediante segmentos de un conjunto de paralelepípedos. Moveremos el robot a diferentes posiciones desde donde tenga distintos puntos de vista de los sólidos y así pueda, tal y como ilustra la figura 5.5, llevar a cabo una reconstrucción 3D de los mismos.

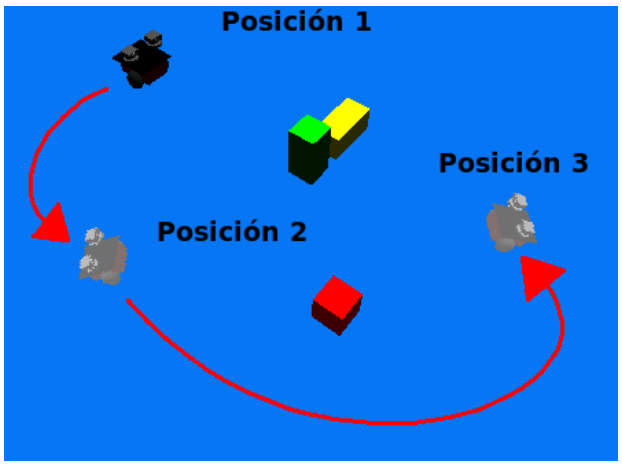

Figura 5.5: Mundo con paralelepípedos y movimientos del robot

Debido a los diferentes cambios de posición del robot se producen oclusiones de algunas aristas, esto provoca que el sistema elimine algunos segmentos 3D memorizados.

En la figura 5.6 se muestran los segmentos almacenados en memoria y la representación de los cubos. La representación está hecha a partir de la posición exacta de los volúmenes que es conocida por el código escrito para el mundo de Gazebo. De manera, que nos sirve para determinar la precisión con la que el sistema sitúa los segmentos 3D. Los segmentos horizontales encontrados en las imágenes de las cámaras representan la mayor dificultad para el sistema.

El sistema genera grandes errores en la triangulación de los segmentos horizontales respecto al plano correspondiente al suelo. Es incapaz de emparejar correctamente los

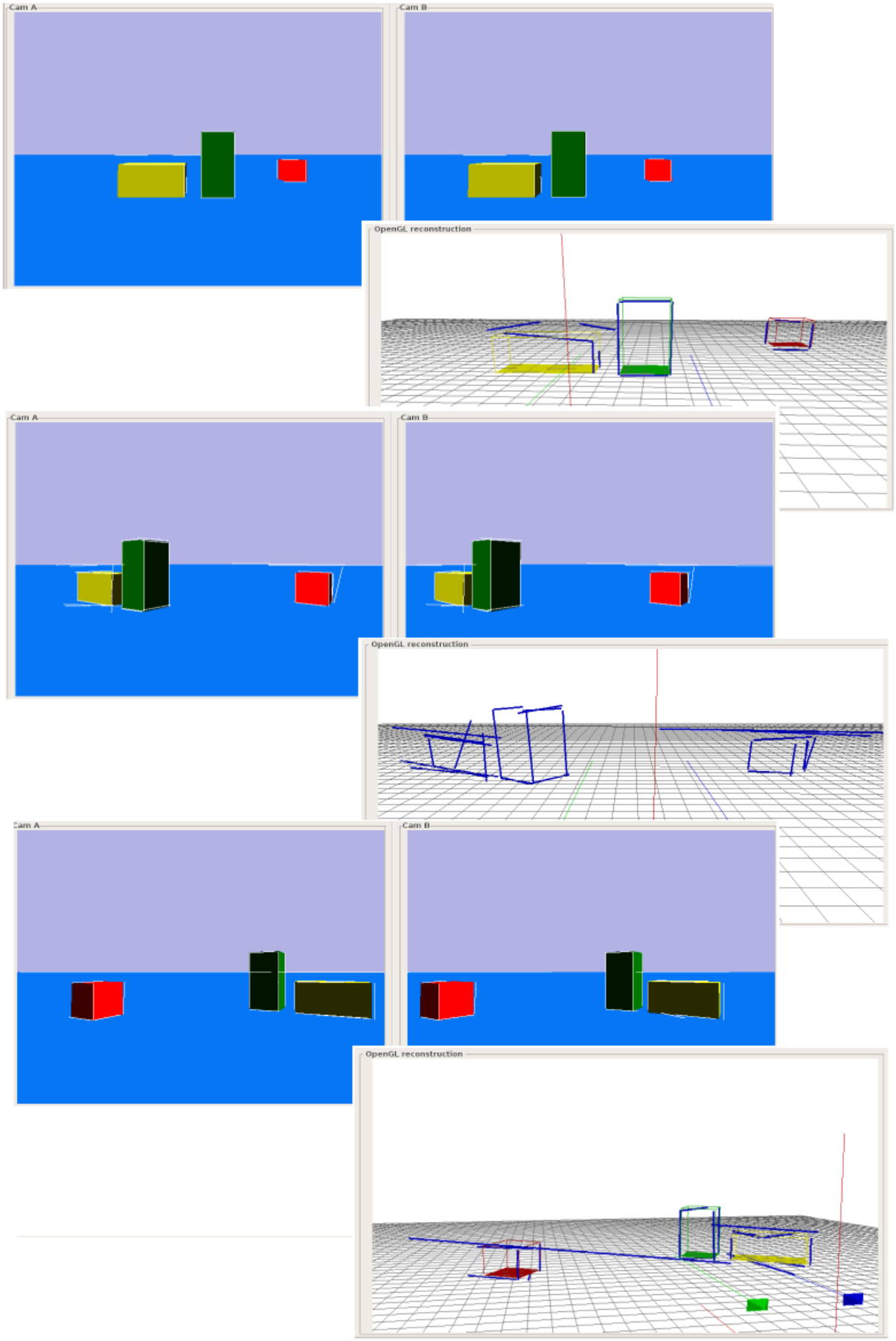

Figura 5.6: Imágenes de las cámaras y reconstrucción realizada en cada momento

segmentos horizontales encontrados en la imagen maestra con sus homólogos en la cámara esclava.

Una de las causas de este error es la falta de texturas en el simulador, todos los colores son puros y esto dificulta el emparejamiento correcto de píxeles homólogos entre las dos imagenes del par estéreo.

La segunda causa es la alineación del par estéreo en un plano horizontal, esto provoca que las líneas epipolares que obtiene el sistema para el cálculo de homólogos sean siempre horizontales. Al querer emparejar segmentos horizontales la intersección entre epipolar y segmento en la image esclava no es limpia.

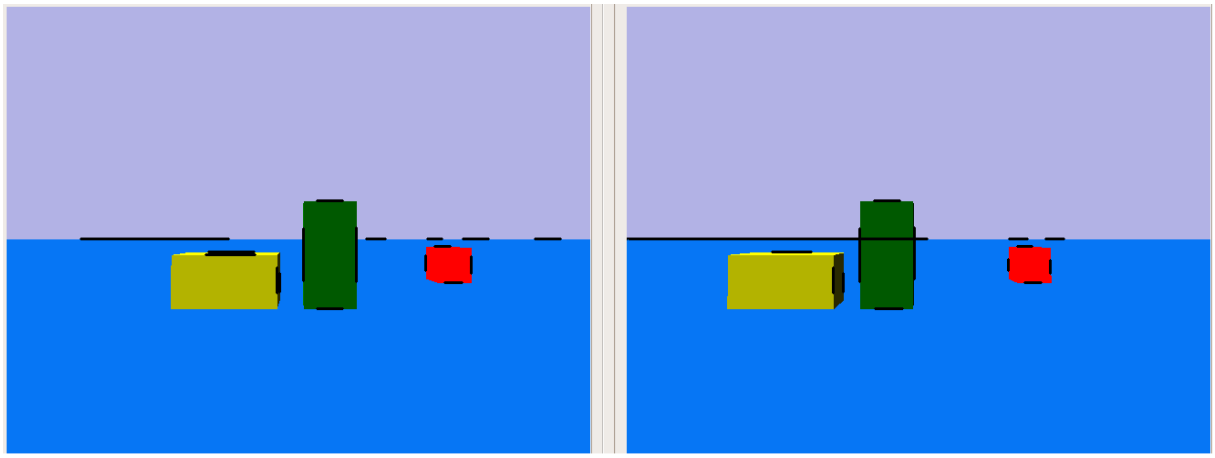

Figura 5.7: Segmentos encontrados en la imagen principal, y sus homólogos en la secundaria

Por ejemplo, para triangular el segmento horizontal en la línea del horizonte de la figura 5.7 se aprecia cómo se lleva a cabo un mal emparejamiento. La línea epipolar horizontal coincide exactamente con la línea del horizonte, donde existen numerosos candidatos erróneos para el emparejamiento debido a los colores puros del simulador.

Como resultado obtenemos un mini-segmento erróneo, y posteriormente un segmento mal triangulado del horizonte del mundo 5.8.

Por esta razón el sistema desarrollado incorpora la alternativa de ignorar aquellos segmentos detectados en el plano imagen que son horizontales, valiéndose tan solo de los segmentos verticales para realizar la reconstrucción 5.9.

Para comprobar la precisión en el posicionamiento de los segmentos en el espacio 3D ejecutamos el sistema con esta opción de ignorar los segmentos horizontales activada. En la figura 5.10 se muestran los segmentos rectos verticales sobre los que centraremos esta prueba.

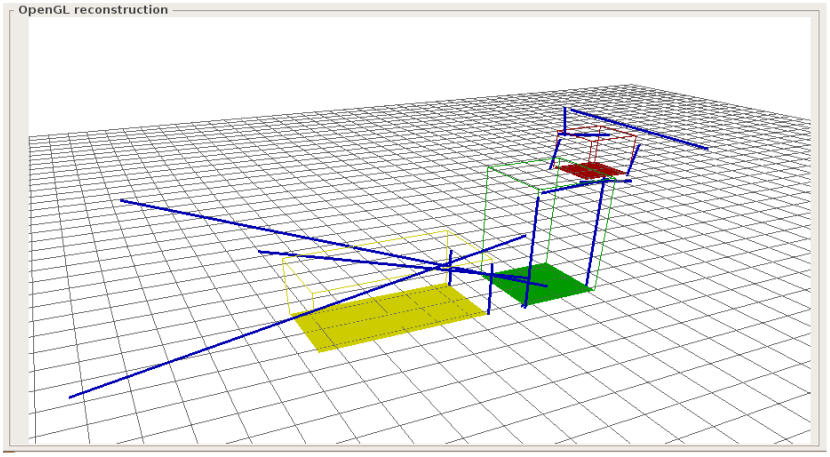

Figura 5.8: Representación de los segmentos en memoria con OpenGL

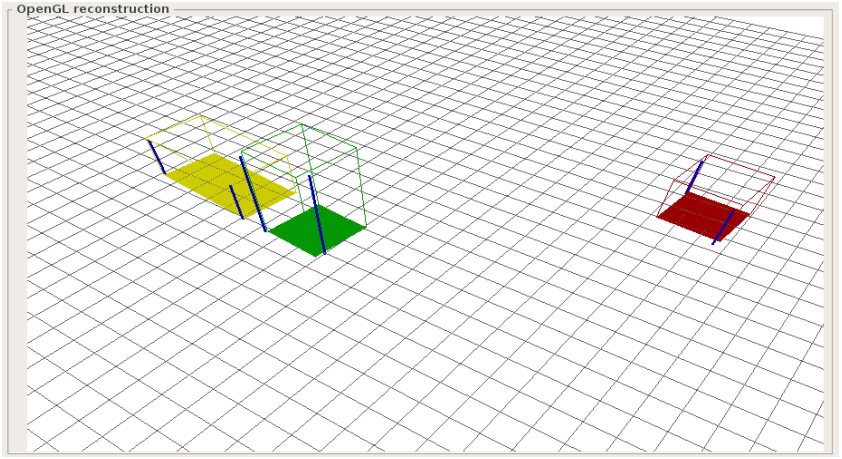

Figura 5.9: Reconstrucción con segmentos verticales

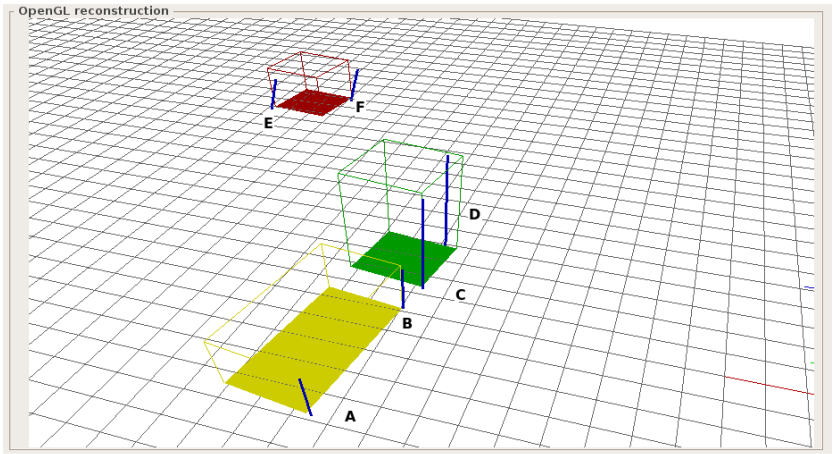

Figura 5.10: Segmentos 3D analizados para comprobar la precisión de la triangulación

Los segmentos analizados se encuentran a una distancia de entre dos y tres metros respecto de las cámaras del robot. En el cuadro 5.2 podemos ver la relación entre los extremos de las aristas reales y la reconstrucción que lleva a cabo el sistema, así como el error que se produce.

Punto	Coordenadas originales	Coordenadas reconstrucción	Error (cm)
A_1	(925.0,-612.5, 0.0)	(925.2,-613.8, -0.7)	1.49
A_2	(925.0,-612.5, 25.0)	(925.1,-613.7, 22.9)	2.42
B_1	(975.0,-612.5, 0.0)	(974.7,-613.1, 0.3)	0.73
B_2	(975.0,-612.5, 25.0)	(974.7,-613.1, 23.5)	1.64
C_1	(987.5,-612.5, 0.0)	(987.1,-613.0, -0.5)	0.81
C_2	(987.5,-612.5, 50.0)	(987.1,-613.1, 47.2)	2.89
D_1	(1012.5,-612.5, 0.0)	(1012.7,-608.3, -1.8)	4.57
D_2	(1012.5,-612.5, 50.0)	(1012.8,-607.6, 48.7)	5.08
E_1	(1087.5,-487.5, 0.0)	(1087.3,-484.9, -2.8)	3.83
E_2	(1087.5,-487.5, 25.0)	(1084.7,-493.2, 21.1)	7.45
F_1	(1112.5,-512.5, 0.0)	(1111.7,-512.3, -2.0)	2.16
F_2	(1112.5,-512.5, 25.0)	(1108.7,-519.4, 22.4)	8.3
		Error medio	3.45

Cuadro 5.2: Precisión de reconstrucción de las aristas de solidos sencillos

Los resultados obtenidos son óptimos, con una media de error de 3,45 centímetros. Por ello podemos afirmar que nuestro sistema consigue una precisión centimétrica. No por casualidad los segmentos que generan mayor error son aquellos más lejanos al robot, es decir los segmentos E y F.

5.3. Calibración de cámaras y modelo geométrico

Para que nuestro sistema pueda triangular correctamente los segmentos 3D necesitamos cámaras calibradas. El modelo geométrico es una pieza fundamental del sistema, ya que nos permite mover las cámaras libremente sin que éstas dejen de estar calibradas. Igualmente, podemos mover el robot a través del mundo sin perder la referencia de dónde se encuentra.

En las primeras pruebas realizadas para la triangulación de segmentos en el espacio, nos dimos cuenta que a larga distancia perdía mucha precisión, de modo que implementamos

una herramienta para la calibración fina de las cámaras en el simulador. Dado que son cámaras ideales conocíamos los parámetros intrínsecos, por lo que determinamos que el error debía encontrarse en los parámetros extrínsecos.

La caracterización de las cámaras del sistema se expresa a través de dos ficheros que contienen los parámetros de ambas cámaras. Con nuestra herramienta podemos variar estos parámetros en tiempo de ejecución. En la figura 5.11 se muestra el interfaz de nuestra aplicación, y el mundo creado en el simulador que nos ayuda en la calibración. En un primer momento realizamos una calibración aproximada con las cámaras fijas a partir de la información que nos ofrece el código del mundo Gazebo sobre la posición de las cámaras.

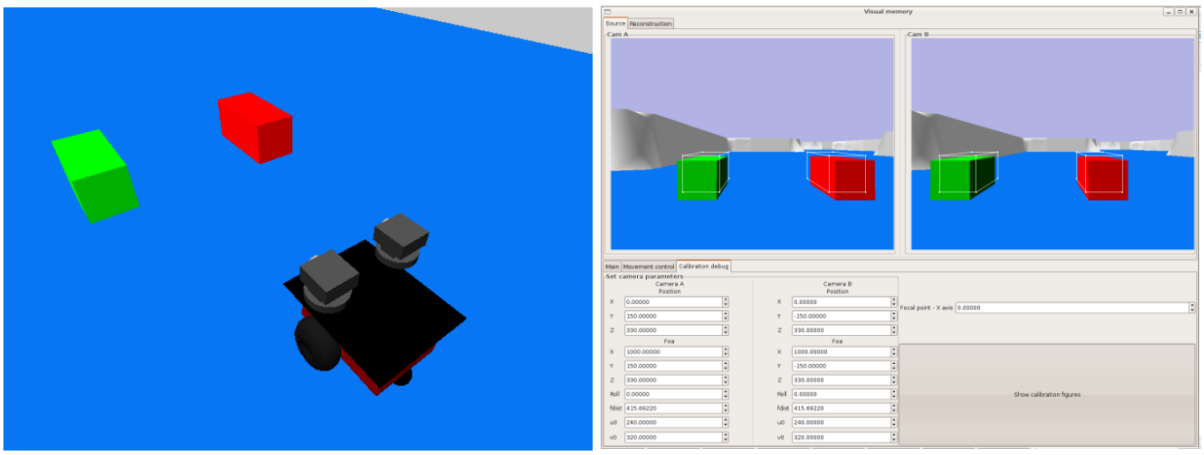

Figura 5.11: Escenario simulado e interfaz del calibrador

El sistema proyecta en el plano imagen los vértices de los paralelepípedos y sus aristas, si la calibración fuera ideal, estos deberían coincidir fielmente con aquellos en el mundo simulado, pero como se aprecia en la figura 5.11 esto no ocurre. Esto era debido a que el parámetro de la posición de la cámara era erróneo. Este parámetro refleja en Progeo la posición del "punto focal"de la cámara, no la posición donde se encuentra el centro físico de la cámara que es el que maneja Gazebo. Por lo que modificando los valores en el eje Z y en el eje X, hicimos coincidir la imagen proyectada con los volúmenes simulados como muestra la figura 5.12.

A continuación probamos el modelo de movimiento de las cámaras moviendo ambas cámaras en el eje horizontal (pan) y vertical (tilt). La proyección de las aristas se movía en función de los movimientos de las cámaras como muestra la figura 5.13, pero no coincidía de forma precisa con los paralelepípedos.

Para obtener un modelo geométrico preciso debemos tener en cuenta que con el movimiento de las cámaras no sólo varía el punto de atención (FOA), si no también la

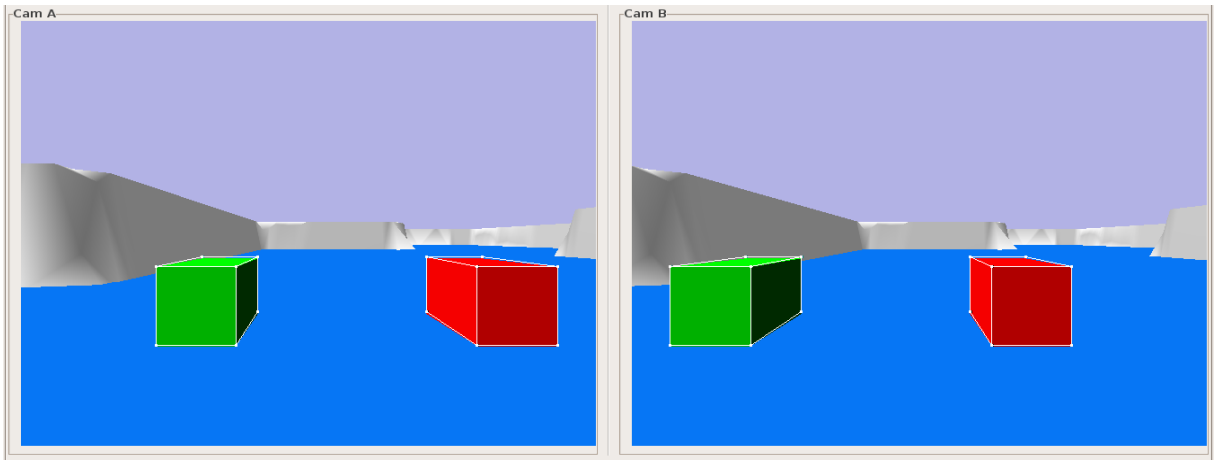

Figura 5.12: Cámaras simuladas calibradas

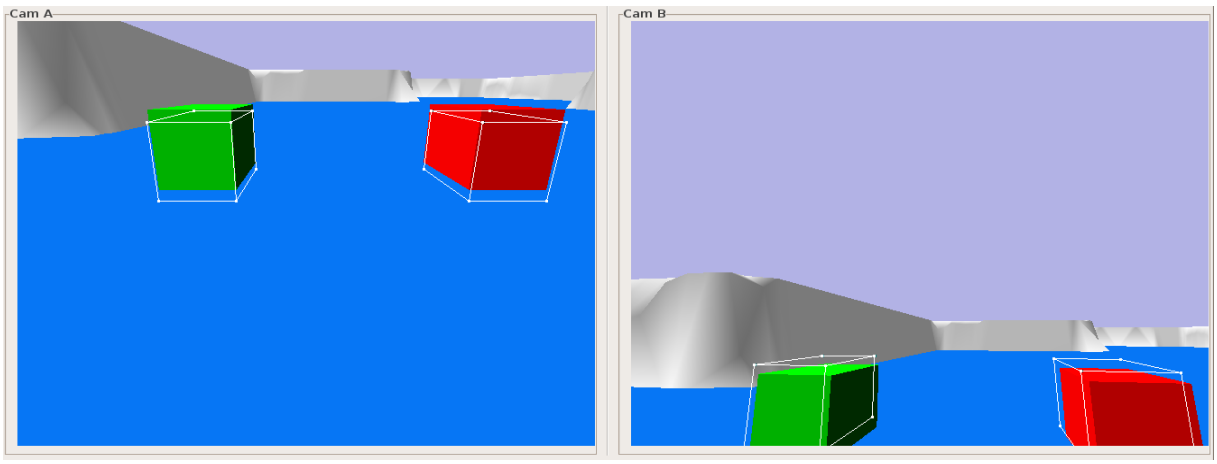

Figura 5.13: Cámaras sin el modelo de movimiento depurado

posición del "punto focal". Por lo que llegamos a la conclusión de que existía un eje de giro fijo situado en el centro físico de la cámara.

El resultado de este experimento se puede apreciar en la figura 5.14 donde a pesar de los diferentes movimientos de las cámaras las arista se proyectan correctamente sobre los paralelepípedos.

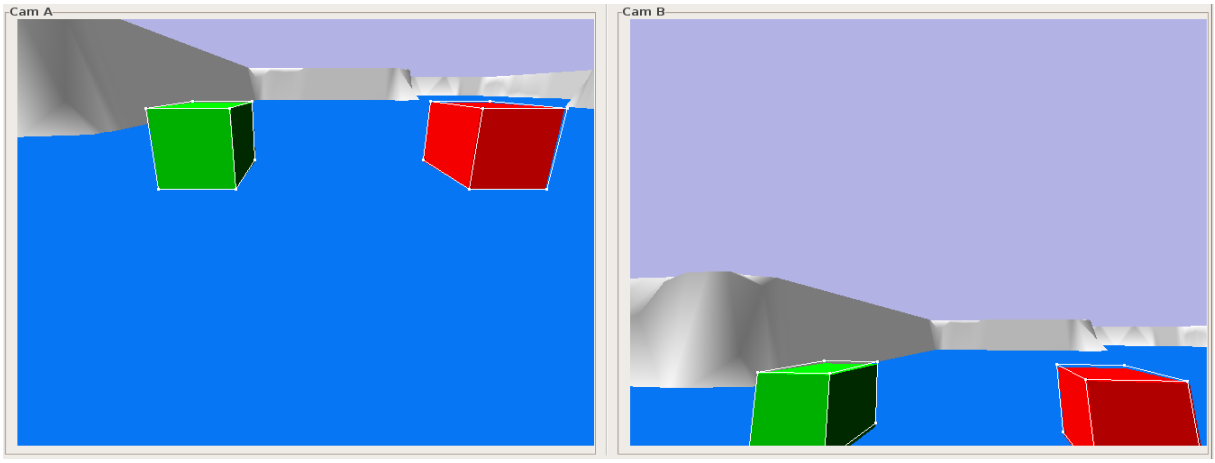

Figura 5.14: Cámaras calibradas correctamente y con el modelo geométrico depurado

El eje de giro de la cámara varía según cada modelo de cámara. El modelo geométrico es útil para aquellas cámaras que poseen el mismo eje de giro para pan y para tilt. Del mismo modo, el "punto focal"varía de una cámara a otra pero el sistema permite configurar esta distancia respecto del eje de giro para cada cámara concreta.

5.4. Incorporación de novedades y corrección en la memoria 3D

En este experimento deseamos realizar un seguimiento de los segmentos presentes en memoria, ilustrar cómo se incorporan nuevos segmentos 3D instantáneos, cómo se corrigen aquellos que ya están almacenados y cómo se eliminan. Para este experimento ejecutamos el algoritmo con el robot moviéndose sobre el mundo departamental. En este entorno situamos al robot en un pasillo y lo teleoperamos para que avance a través de él. El sistema realiza una reconstrucción 3D de aquello que ve en cada momento.

En cada iteración del sistema el algoritmo de reconstrucción sitúa en el espacio tridimensional los segmentos detectados por el sistema de procesamiento 2D. Los segmentos

3D instantáneos obtenidos son añadidos a la memoria de segmentos 3D, y en caso de estar cercanos al robot se incluyen también en la memoria caché. En la figura 5.15 se aprecia cómo se incorporan a memoria un conjunto de segmentos 3D instantáneos verticales, todos estos segmentos tendrán un valor de incertidumbre, algunos de ellos por estar lejanos al robot no están fielmente situados y será necesaria la corrección de su posición más adelante.

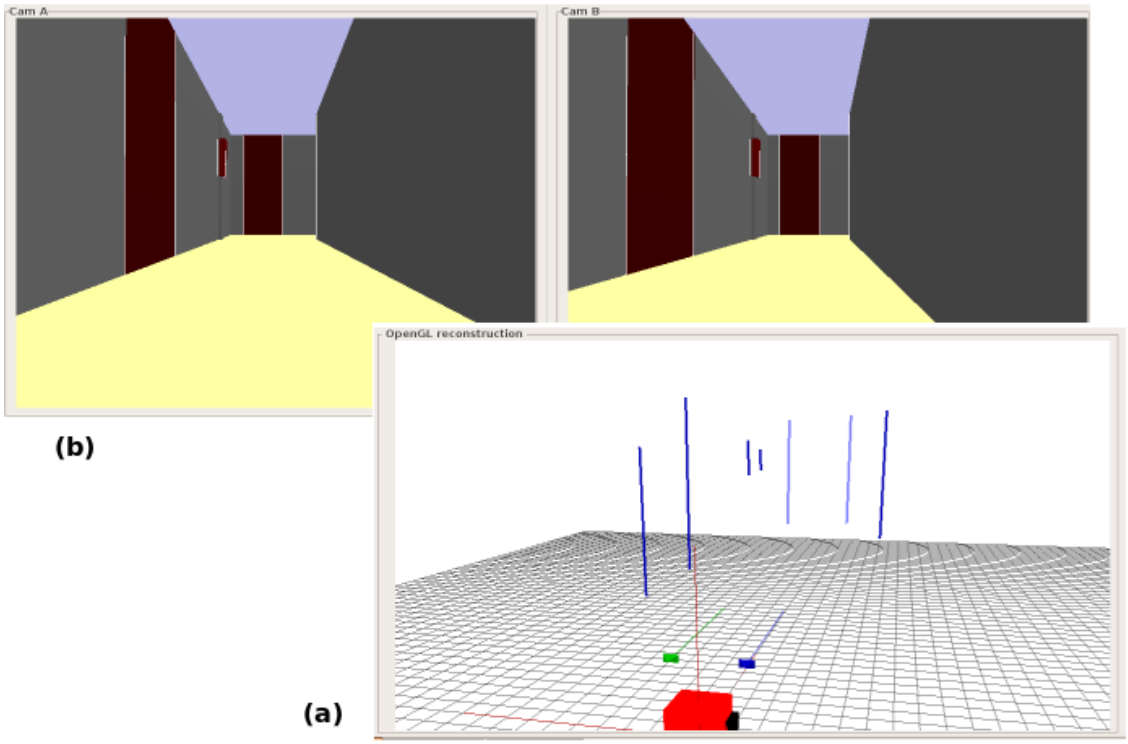

Figura 5.15: (a) Segmentos 3D incorporados a memoria y (b) proyección de la memoria en el plano imagen

Dependiendo de la distancia a la que se encuentren estos segmentos el sistema los incorporará a memoria caché. Los segmentos almacenados en la memoria 3D provocan que se generen predicciones. Haciendo uso de la biblioteca de geometría proyectiva *Progeo* el sistema proyecta aquellos segmentos 3D almacenados en memoria sobre el plano imagen. El sistema de procesamiento 2D analizará las imágenes captadas por la cámara maestra y comparará los segmentos 2D obtenidos con el conjunto de las predicciones realizadas.

En la figura 5.16 se ve cómo a medida que el robot se aproxima a la puerta de la izquierda del pasillo, el sistema corrige el segmento 3D memorizado que define el lateral izquierdo del marco de la puerta, mientras que el lateral derecho se conserva y su proyección sobre la imagen ahorra un valioso tiempo de cómputo al algoritmo. Dado que la proyección de la

memoria no coincide fielmente con el segmento real el sistema calcula un nuevo segmento 3D que tendrá un valor de incertidumbre menor. El nuevo segmento 3D triangulado se fusionará con el antiguo tal y como se explicó en el capítulo anterior.

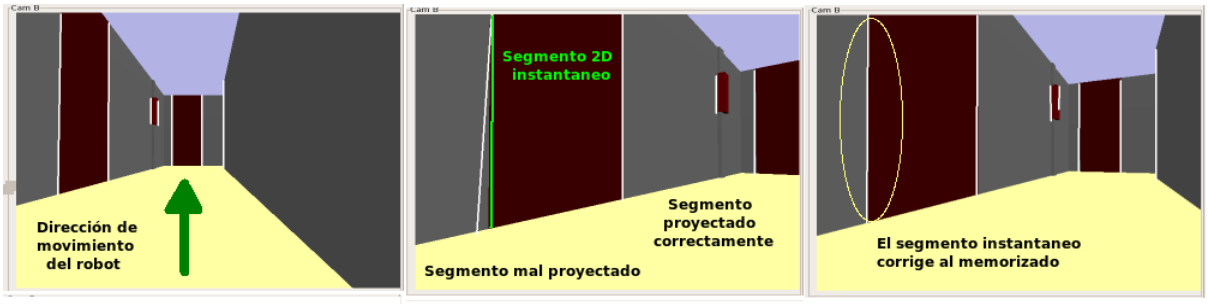

Figura 5.16: Secuencia de corrección de un segmento 3D memorizado

Por otro lado, aquellos segmentos 3D memorizados que generan una predicción que no es confirmada por ningún segmento real detectado en el plano imagen serán inmediatamente eliminados de la memoria de segmentos de forma permanente. La comparación entre las predicciones y los segmentos 2D instantáneos es bastante tolerante, lo que quiere decir que sólo se eliminarán aquellos segmentos que realmente no existan en el mundo.

En la figura 5.17 se ve cómo el robot continua avanzando por el pasillo y se aproxima lo suficiente a la puerta al fondo del pasillo como para realizar una mejor reconstrucción de los bordes laterales de la puerta. En este proceso elimina aquellos segmentos que el sistema había triangulado mal debido a la distancia a la que se encontraba, al mismo tiempo que calcula los nuevos.

Con estos experimentos hemos probado las siguientes conclusiones que adelantamos en el capítulo anterior. En un entorno lleno de segmentos rectos el sistema de procesamiento 2D realiza una eficiente comparación con la predicción generada a partir de los segmentos 3D en memoria. El sistema saca partido de estas predicciones para ahorrar tiempo de cómputo a la triangulación de segmentos, corrige aquellos segmentos que a larga distancia estaban mal posicionados y elimina segmentos espúreos que no coinciden con ningún segmento en la realidad.

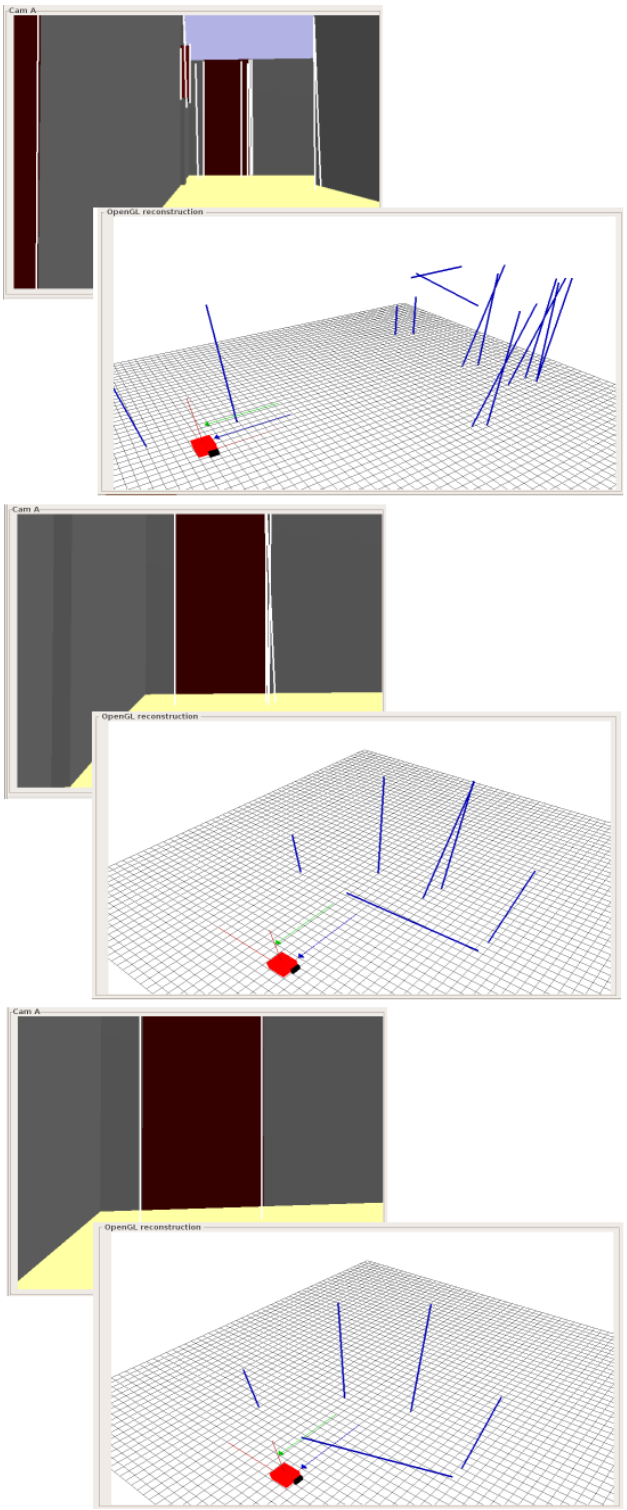

Figura 5.17: Secuencia de eliminación de segmentos 3D memorizados

5.5. Generación de estímulos estructurados

Deseamos comprobar que la generación de hipótesis perceptivas y su validación se realiza correctamente. Para este experimento ejecutamos el algoritmo de reconstrucción sobre el mundo departamental.

Uno de los estímulos estructurados capaces de identificar nuestro sistema son las puertas. Cada vez que se completa la ejecución del algoritmo de reconstrucción se lleva a cabo una comprobación de los segmentos incluidos en memoria caché para determinar si dos de ellos definen una puerta.

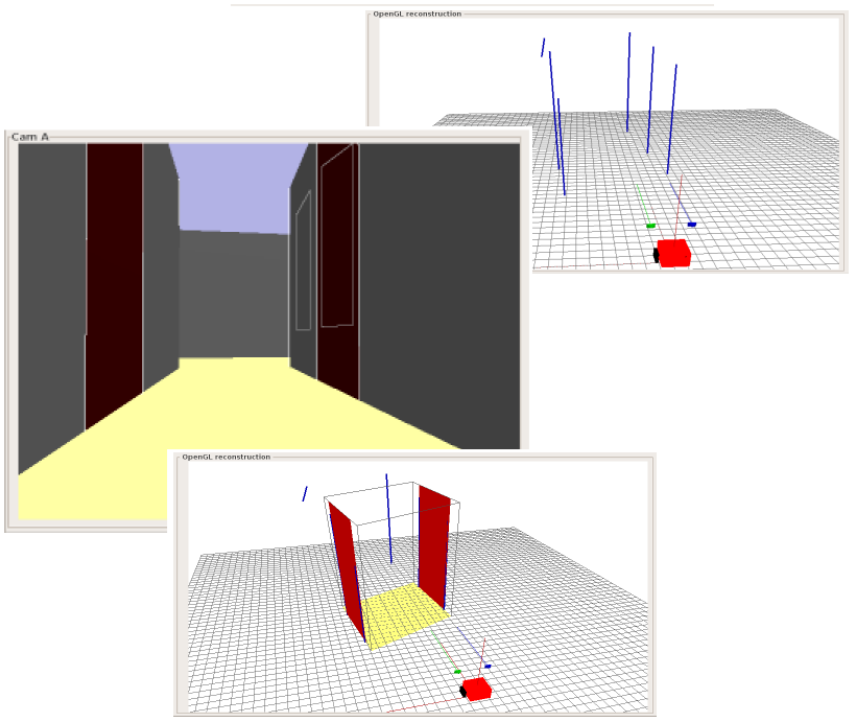

Figura 5.18: Secuencia de detección de estímulos estructurados, puerta

En la figura 5.18 se aprecia la secuencia de generación de la hipótesis perceptiva puerta y de la validación de la puerta de la derecha mediante el filtro de color sobre la imagen.

También se ve el caso en el que dos segmentos verticales y próximos entre sí generan una hipótesis refutada en la imagen. En la comprobación se detecta que es una tentativa de puerta entre la puerta real y el final del pasillo, la puerta es eliminada de la memoria.

A partir de un conjunto de puertas el sistema construye los pasillos del mundo. El sistema examina el conjunto de puertas almacenadas cada determinado número de ejecuciones del algoritmoen memoria. Comprueba en primer lugar qué puertas forman

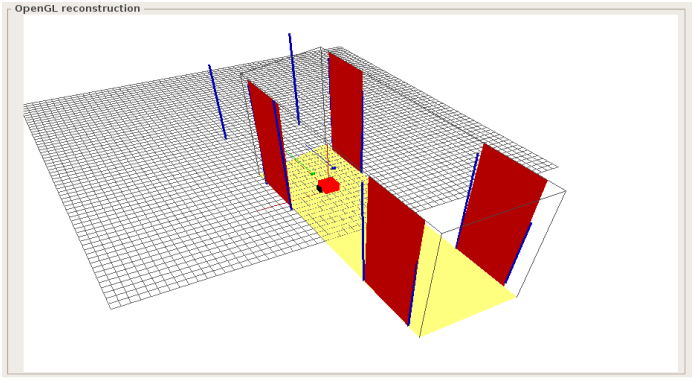

Figura 5.19: Secuencia de detección de estímulos estructurados, pasillo

parte de una misma pared, a continuación el algoritmo busca entre las paredes obtenidas aquellas que son paralelas y están separadas una distancia igual a la anchura del pasillo previamente definida. Este proceso se ilustra en la figura 5.19.

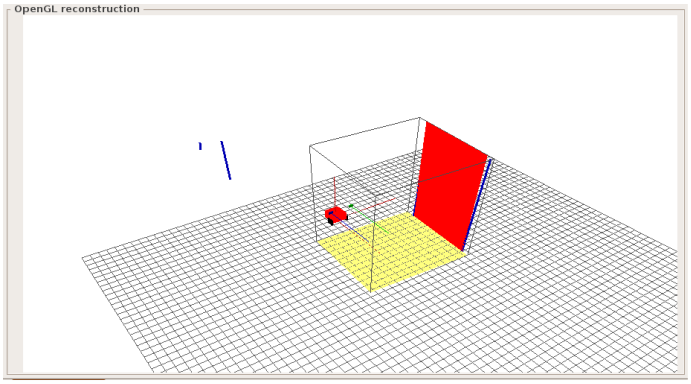

Figura 5.20: Secuencia de creación de una sección del pasillo

En segundo lugar el algoritmo puede formar pasillos a partir de una unica pared con puertas, esto es el caso de aquellos pasillos que tiene puertas tan solo en una de sus paredes. Cuando el algoritmo termina de crear pasillos a partir de las paredes paralelas, creará nuevos pasillos a partir de las paredes que no hayan sido utilizadas hasta el momento. De este modo se crearán pequeñas secciones de pasillo delante de la puerta que generó la pared, tal y como se muestra en la figura 5.20.

El último paso que realiza el algoritmo para obtener una representación fidedigna de los pasillos es llevar a cabo una intersección de los mismos. Esto significa prolongar las paredes de los pasillos obtenidos hasta el punto de unión con la pared de otro pasillo.

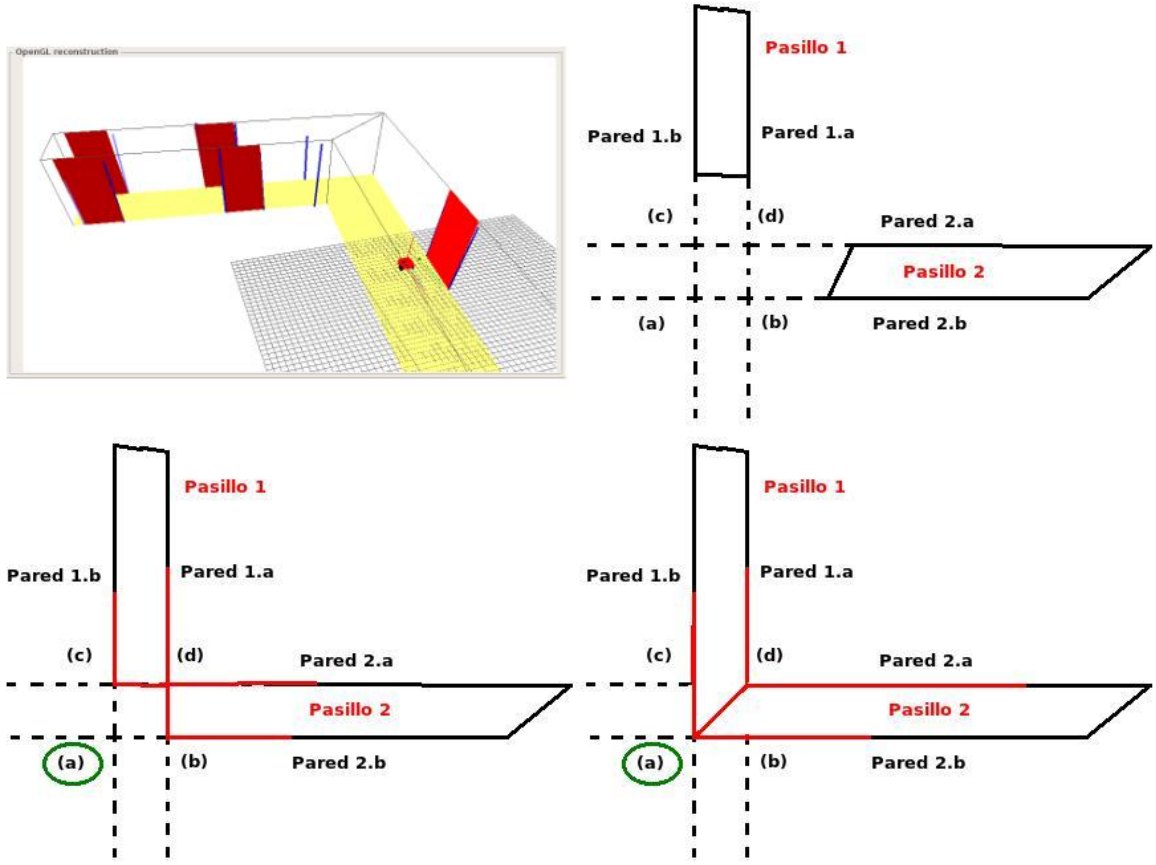

Figura 5.21: Secuencia de intersección de dos pasillos

En la figura 5.21 se muestra el proceso completo de la intersección de dos pasillos, primero calculados los puntos de intersección de la prolongación de las paredes de ambos pasillos, a ,b ,c y d. En segundo lugar prolongamos los pasillos hasta los puntos los dos primeros puntos de intersección, en este caso para el pasillo uno serán c y d, y para el pasillo 2 será d y b. En tercer lugar comprobamos los cuatro puntos de intersección y obtenemos el punto a como el único que no pertenece a los pasillos. Por último prolongamos las dos paredes que contiene a este punto a hasta este punto. Este proceso maneja mucho conocimiento ad hoc del mundo en el que se mueve el robot: pasillos rectos y perpendiculares entre sí.

Como resultado obtenemos la intersección de ambos pasillos que permite representar correctamente ambos pasillos y sus paredes, como muestra la figura 5.22

De este experimento sacamos la conclusión de que el algoritmo sitúa correctamente la posición de las puertas presentes en el entorno. En aquellos casos en los que dos segmentos que no son una puerta real generen la hipótesis perceptiva puerta la comprobación por

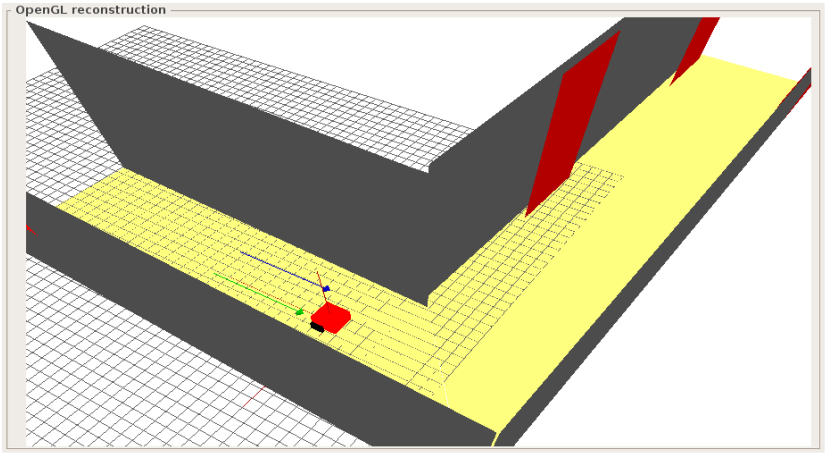

Figura 5.22: Visualización de la intersección de dos pasillos

medio de las imágenes de las cámaras desecha dicha hipótesis. Además comprobamos que los parámetros que utiliza el sistema para definir los estímulos perceptivos pared y pasillo son correctos. La manera en que se lleva a cabo la intersección de pasillos es válida y evita que se produzca un error en la visualización situando una sección de la pared en mitad de un pasillo.

Capítulo 6

Conclusiones y trabajos futuros

En capítulos anteriores se ha descrito el problema abordado en el proyecto y la solución propuesta junto con una serie de experimentos que la validan y caracterizan. En este capítulo se exponen las conclusiones globales del trabajo realizado y se proponen un conjunto de líneas útiles de investigación que pueden continuar este trabajo.

6.1. Conclusiones

Los objetivos propuestos para este proyecto fin de carrera implicaban que la aplicación fuera capaz de realizar una reconstrucción tridimensional de un mundo creado en el simulador Gazebo, a partir de la información recogida por su par estéreo de cámaras móviles.

La conclusión general es que se ha conseguido desarrollar un sistema de reconstrucción para el robot muy completo. El sistema es capaz de situar en el espacio tridimensional los segmentos rectos captados por sus cámaras en cada instante. La memoria visual guarda los segmentos 3D a largo plazo, permitiendo al robot moverse por el mundo para reconstruir otras áreas. Esta memoria 3D está acoplada con las imágenes de entrada, esto permite una extracción más eficiente de los segmentos a tratar. Además consta de percepción abductiva, que permite la representación del mundo reconstruido mediante estimulos más abstractos.

Los objetivos y subobjetivos propuestos en el segundo capítulo han sido cubiertos con éxito. Este sistema lo hemos dividido en tres bloques para facilitar el desarrollo. Cada uno de ellos realiza una tarea específica, pero se combinan para formar el sistema completo:

1. **Sistema de procesamiento 2D:** En este subobjetivo se proponía la detección de los segmentos rectos que aparecían en las imágenes captadas por las cámaras. El

sistema utiliza la librería gráfica *OpenCV*, y en concreto las funciones del filtro de Canny para la obtención de bordes, y de la transformada de Hough para la obtención de segmentos rectos en la imagen.

El sistema además tiene la posibilidad de ignorar los segmentos horizontales, por las causas explicadas en el capítulo de experimentos.

Nuestro sistema de procesamiento 2D no es un ente aislado del resto del sistema, por el contrario, se encuentra muy acoplado con la memoria visual 3D, valiéndose de las predicciones generadas por la misma, ignora segmentos 2D ya almacenados o ayuda a eliminar aquellos sin confirmación.

Antes de pasarle el conjunto de segmentos 2D que deben ser reconstruidos al sistema de reconstrucción 3D, se realiza un post-tratamiento de los segmentos 2D, para unir aquellos segmentos que la transformada de Hough a considerado que eran dos segmentos distintos. Esto mejora el posicionamiento final de los segmentos 3D en el espacio.

2. **Segmentos instantáneos 3D:** En este subobjetivo se pedía un sistema de reconstrucción 3D que ubicara en el espacio de forma precisa los segmentos 2D encontrados en el plano imagen. Para resolver este subobjetivo hemos creado un algoritmo para la triangulación de segmentos rectos en el espacio tridimensional, utilizando el par estéreo de cámaras y valiéndonos del método clásico de triangulación de puntos en el espacio mediante la búsqueda del píxel homologo en la otra cámara. Este algoritmo es especialmente preciso para segmentos verticales, como se explica en el experimento 5.6, pero pierde precisión cuando aumenta la distancia hasta el segmento real.

   El sistema obtiene el conjunto de segmentos 3D que ve en cada momento puesto que se trata de un sistema perceptivo pasivo.

   En ocasiones, el sistema puede representar un único segmento real por medio de dos o más segmentos 3D, por lo que el sistema realiza un post-tratamiento de estos segmentos, previo a incorporarlos a memoria 3D, esto repercute en una mejora de la representación.

3. **Memoria de objetos 3D:**

   En este subobjetivo se pedía implementar una memoria de segmentos 3D a largo plazo, estos segmentos deben representarse en coordenadas absolutas para que su posición no se modifique con el movimiento del robot, concediéndole la propiedad

de persistencia a la información sensorial instantánea. Para guardar la posición de estos objetos se ha construido un modelo geométrico robusto capaz de expresar estos objetos según un sistema de referencia solidario con el robot o con coordenadas absolutas del mundo.

Los segmentos 3D instantáneos calculados en la etapa anterior son incorporados a memoria 3D. Se ha implementado una memoria caché de segmentos para que la incorporación se lleve a cabo de forma eficiente.

La memoria visual genera un conjunto de predicciones a partir de los segmentos 3D almacenados que deberían aparecer en el plano imagen de las cámaras. Con esta predicción el sistema de procesamiento 2D es capaz de eliminar segmentos espúreos, corregir segmentos mal triangulados y ahorrar tiempo de cómputo de aquellos segmentos reales ya almacenados en memoria.

A partir de los segmentos 3D almacenados se generan también un conjunto de hipótesis perceptivas, puertas, paredes, pasillos o cuadrados, que permiten tener una representación más abstracta del mundo. Estas hipótesis perceptivas se convierten en estímulos estructurados, al confirmarse por medio de las imágenes captadas por las cámaras.

Además de los objetivos, se han satisfecho los requisitos funcionales exigidos:

1. El sistema funciona correctamente bajo la plataforma GNU/Linux concretamente sobre la distribución Ubuntu 8.04.

2. El diseño se ha concebido siguiendo la arquitectura de JDErobot. Se ha tenido que modificar el driver Gazebo incluido en la versión 4.3 de JDErobot, para soportar dos cámaras moviles. La aplicación es un esquema de JDErobot. El código del esquema esta íntegramente escrito en los lenguajes C y C++.

3. El sistema implementado funciona correctamente sobre el simulador Gazebo, en concreto sobre la versión 0.7.0.

4. El sistema de reconstrucción 3D implementado realiza sus cálculos de forma vivaz y eficiente, gracias al ahorro de cómputo fruto de las predicciones instantaneas.

5. La precisión del sistema es mayor de lo que cabía esperar, consiguiendo una precisión centimétrica aún con el robot en movimiento. Esto es así porque en el simulador la auto-localización del robot es perfecta, y porque la corrección continua

de segmentos desde imágenes permite refinarlos continuamente, especialmente con imágenes obtenidas de cerca, tal y como se ha expuesto en el capítulo de experimentos.

El sistema ha sido validado experimentalmente con imágenes simuladas, como se vió en el capítulo cinco. Estos experimentos han consumido gran parte del tiempo de desarrollo del sistema perceptivo, pero han permitido afinar el sistema y evaluar diversas alternativas.

Este proyecto de ingeniería tiene un marcado carácter de investigación. Comenzamos con un problema abierto que debía ser resuelto con técnicas novedosas y del que no se sabía cuales sería los resultados finales, o cuán positivos iban a ser éstos. Por ello ha sido muy importante tener un diseño preliminar sujeto a cambios, una implementación que nos permitiera experimentar con diferentes configuraciones, y un plan de pruebas robusto para poder decidir la configuración ideal en cada situación.

Son aportes genuinos de este proyecto:

1. El algoritmo de reconstrucción de una escena con segmentos 3D.
2. El modelo geométrico implementado para el robot Pioneer, y el par estéreo en un mundo simulado de Gazebo.
3. La memoria de segmentos 3D, que nos permite la proyección de los segmentos 3D sobre las imágenes de las cámaras.
4. La memoria de estímulos estructurados, que nos permite la visualización fidedigna del mundo reconstruido por el sistema.
5. Los aportes a la infraestructura de JDErobot son el *esquema ColorTuner* y las mejoras realizadas al *driver Gazebo*.

Los resultados de este proyecto, documentación, código y material multimedia están disponibles en la pagina web de JDErobot [1].

6.2. Trabajos futuros

Las diferentes líneas de investigaciones futuras que se puedan realizar a partir de este trabajo, pueden agruparse en avances a corto, medio y largo plazo. A corto plazo sin duda se podrían aumentar el número de primitivas para la reconstrucción, dando lugar

[1] https://www.jderobot.org/index.php/Miangolarra

a un mayor numero de hipótesis perceptivas que el sistema sería capaz de identificar, como paralelepípedos ó esferas. Se podría introducir el concepto de opacidad de todos estos objetos para evitar la eliminación de segmentos almacenados en memoria que no son visibles en un instante dado por alguna oclusión. Se podría mejorar la representación de lo reconstruido sacando el máximo partido de la API de OpenGL, introduciendo iluminación de la escena o texturas en los objeto 3D. Además, se podrían incorporar al sistema técnicas de atención visual centrando la atención en aquellas partes del mundo sin reconstruir, o que se desea reconstruir con mayor nitidez.

A medio plazo sería imprescindible probar el sistema en el mundo real. Para esta tarea no sería necesario realizar una gran refactorización del código, puesto que el componente sobre el robot real se apoya sobre la plataforma JDErobot para recibir la imágenes de entrada o para manejar los cuellos mecánicos de las mismas. Sobre el robot real el sistema utilizaría el mismo interfaz para recibir las imágenes ya sean reales o simuladas, igualmente, utilizaría el mismo interfaz para manejar los motores de los cuellos mecánicos ya sean reales o simulados. Por otro lado, para obtener óptimos resultados en el mundo real sería necesario solventar los problema de auto-localización dentro del mundo y obtener una calibración fina del par estéreo de cámaras.

A largo plazo es siempre difícil detallar qué clase de aplicaciones se pueden desarrollar, pero se podría construir un mapa local del mundo con estas técnicas, y a partir de este mapa investigar como utilizarlo para realizar una auto-localización del robot. También se podrían realizar mejoras para que el sistema fuera capaz de representar objetos dinámicos, como personas, vehículos u otros robots.

Bibliografía

[Abella Dago, 2009] Gonzalo José Abella Dago. Percepción 3d con atención visual en robots con cámaras. *Proyecto fin de carrera - Ingeniería técnica en informática de sistemas - Universidad Rey Juan Carlos*, 2009.

[Agüero Durán, 2010] Carlos Agüero Durán. Técnicas de percepción compartida aplicadas a la asignación dinámica de roles en equipos de robots móviles. *Tesis doctoral - Universidad Rey Juan Carlos*, 2010.

[Calvo Palomino, 2008] Roberto Calvo Palomino. Reconstrucción 3d mediante un sistema de atención visual. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2008.

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

[Cañas Plaza, 2005] Jose María Cañas Plaza. Overt visual attention inside jde control architecture. *EPIA*, 2005.

[Cañas Plaza, 2008] Jose María Cañas Plaza. Manual de programación de robots con jde. *Universidad Rey Juan Carlos*, noviembre 2008.

[Esteban Pacios, 2007] José María Esteban Pacios. Reconstrucción 3d visual con triangulación. *Proyecto fin de carrera - Ingeniería técnica en informática de sistemas - Universidad Rey Juan Carlos*, 2007.

[García Martínez, 2007] Iván García Martínez. Reconstrucción 3d visual con algoritmos evolutivos. *Proyecto fin de carrera - Ingeniería técnica en informática de sistemas - Universidad Rey Juan Carlos*, 2007.

[Gates, 2007] Bill Gates. A robot in every home. *Scientyfic American*, 2007.

[Kachach, 2008] Redouane Kachach. Calibración automática de cámaras en la plataforma jdec. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2008.

[León Cadahía, 2006] Olmo León Cadahía. Navegación de un robot con un sistema de atención visual 3d. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2006.

[Manuel Domínguez, 2009] Jose Manuel Domínguez. Autolocalización probabilística visual de un robot en el simulador gazebo. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2009.

[Martínez de la Casa Puebla, 2005] Marta Martínez de la Casa Puebla. Sistema de atención visual en escena. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2005.

[Mendoza Baños, 2008] Manuel Mendoza Baños. Reconstrucción 3d visual mediante triangulación con cámaras. *Proyecto fin de carrera - Ingeniería técnica en informática de sistemas - Universidad Rey Juan Carlos*, 2008.

[Perdices García, 2009] Eduardo Perdices García. Autolocalización visual en la robocup basada en detección de porterías 3d. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2009.

[S. Birchfield, 1999] C. Tomasi S. Birchfield. Depth discontinuities by pixel-to-pixel stereo. *International Journal of computer Vison*, 1999.

[Sanchez Cadenas, 2007] José Antonio Sanchez Cadenas. Cálculo y aplicaciones del flujo óptico en tiempo real. *Proyecto fin de carrera - Ingeniería técnica en informática de sistemas - Universidad Rey Juan Carlos*, 2007.

[Shreiner *et al.*, 2007] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. Opengl(r) programming guide, v2.1. *Addison-Wesley Professional*, 2007.

[Vega Pérez, 2008] Julio Manuel Vega Pérez. Navegación y autolocalización de un robot guía de visitantes. *Proyecto fin de carrera - Ingeniería en informática superior - Universidad Rey Juan Carlos*, 2008.