INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2008-2009

Proyecto Fin de Carrera

Percepción 3D con atención visual en robots con cámaras móviles

Autor: Gonzalo José Abella Dago

Tutor: José María Cañas Plaza

A mi familia

Agradecimientos

Quiero dar las gracias a todos los miembros del Grupo de Robótica de la Universidad Rey Juan Carlos por su apoyo y colaboración. Y en especial a Julio y Emma y a toda la gente que hemos estado casi a diario sufriendo en el laboratorio.

Y de manera especial quería agradecerle a José María todo el trabajo y esfuerzo que me ha dedicado durante todo el año y en este *sprint* final.

A todos, ¡¡muchas gracias!!

Resumen.

Los sistemas de visión son hoy en día uno de los elementos sensoriales más atractivos para su investigación en la robótica. Las cámaras nos aportan mucha información sobre el entorno del robot y, comparado con otros sensores, son bastante más baratos. Pero tienen una desventaja, extraer información de las imágenes capturadas es una tarea difícil y costosa. Los sistemas de atención visual nos facilitan esta tarea, seleccionando los puntos o áreas de interés de la escena.

El presente proyecto aborda el desarrollo de un sistema perceptivo de atención abierta para un robot. Por medio de un par de cámaras móviles, utilizando imágenes reales, localizamos los objetos interesantes de la escena y realizamos un seguimiento de ellos. Para esta tarea hemos diseñado un sistema que combina atención 2D de las imágenes con atención 3D para la memoria de puntos ya reconstruidos, enriqueciendo notablemente la información de la escena. También hemos propuesto un algoritmo de reconstrucción 3D novedoso basado en el movimiento ocular de las cámaras. Con este algoritmo proponemos una alternativa algoritmo clásico de reconstrucción 3D mediante el cálculo de la epipolar y triangulación y, exprimimos todo el potencial al movimiento de las cámaras.

La implementación *software* de los algoritmos propuestos se ha diseñado y programado en C++. El componente *software* ha sido implementado en forma de esquema de JDE, aplicación que nos facilita la programación en entornos robóticos. También nos hemos apoyado en numerosas bibliotecas populares en la comunidad del software libre.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este capítulo se dará una visión panorámica del contexto del proyecto. Presentaremos el entorno en el que se sitúa el estudio realizado desde un punto de vista más general, la robótica, hasta centrarnos en el área concreta que hemos investigado, la atención visual.

1.1. Robótica

La robótica es la disciplina encaminada a diseñar y construir aparatos y sistemas capaces de realizar tareas propias de un ser humano. A estos aparatos se les conoce como *robots*.

El término robot deriva de la palabra checoslovaca *robota*, acuñada por el escritor Karel Capek en su obra R.U.R. (Robots Universales de Rossum), que quiere decir literalmente siervo y figuradamente trabajo duro o penoso. Un robot está compuesto básicamente por tres componentes: sensores, actuadores y procesadores.

Los sensores son dispositivos capaces de transformar magnitudes físicas o químicas en magnitudes eléctricas. Cosas que se pueden medir son, por ejemplo, distancia recorrida, temperatura, intensidad lumínica, etc. Los actuadores son dispositivos que reciben y ejecutan órdenes. Al igual que los sensores, hay actuadores de muchos tipos aunque los más habituales son los motores o los servo-motores. Y por último, tenemos los procesadores. Estos son los responsables de recoger y comprender los datos suministrados por los sensores y devolver una respuesta al mundo real por medio de los actuadores. Podríamos decir que, analógicamente, estas son las funciones desempeñadas en los humanos por los sentidos, el cerebro y los músculos.

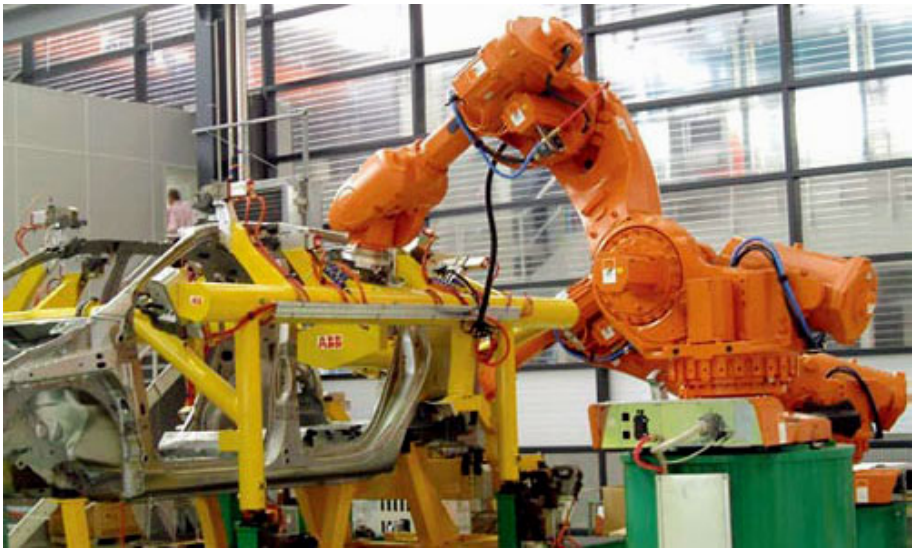


Figura 1.1: Ejemplo de un brazo mecánico trabajando en una cadena de montaje de automóviles.

Los robots en la actualidad van cobrando cada vez más relevancia. Son usados masivamente para la realización de tareas peligrosas, pesadas o repetitivas que pueden resultar difíciles para un ser humano. Uno de los sectores donde más ventaja ha tomado la investigación y el uso de robots ha sido en el de la industria. Estos son especialmente útiles en cadenas de montaje. Otro sector que usa bastante los robots es el de la manipulación de material peligroso, como productos radiactivos o explosivos, y para tareas de exploración oceánica o espacial.

A finales de la década de 1970 y principios de la década siguiente surge una rama de la robótica conocida como *robótica autónoma*. Se define como el área que desarrolla robots capaces de actuar en entornos complejos sin intervención humana. En estos casos la información recibida por los sensores es vital, ya que según los datos recibidos se tomarán distintas decisiones. Es un campo estrechamente relacionado con la inteligencia artificial.

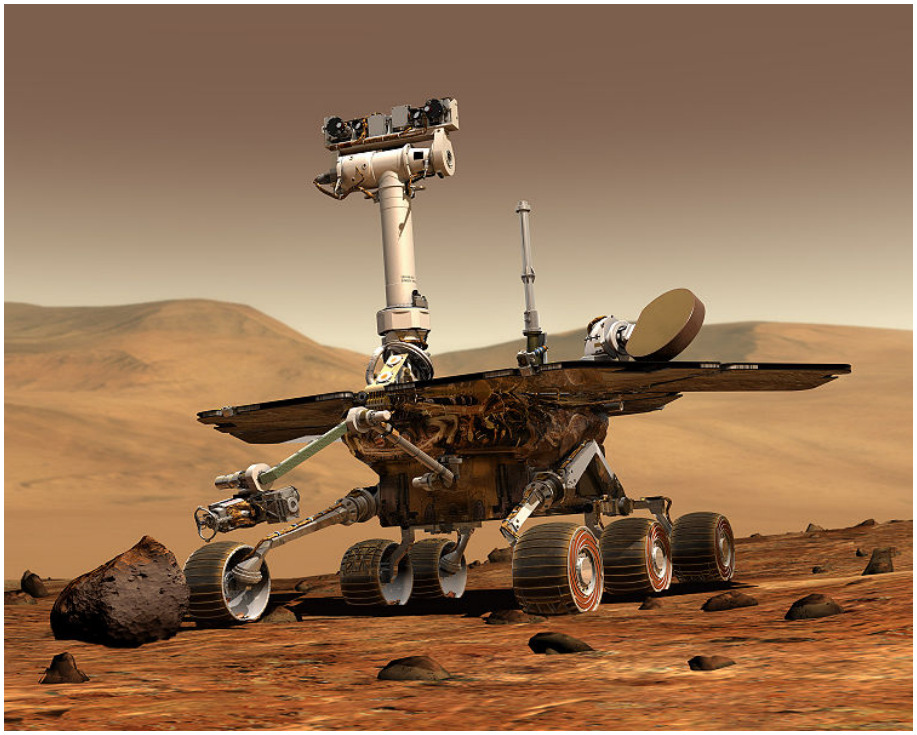


Figura 1.2: La sonda espacial *Spirit*.

Un ejemplo de este tipo de robots son las sondas enviadas a Marte *Spirit* y *Opportunity*. Estos robots gemelos tienen todo tipo de sensores para medir e interpretar el entorno. En un principio fueron enviadas para cumplir una misión de reconocimiento de Marte de 90 días, pero el pasado día 25 de enero de 2009 las sondas cumplieron cinco años. En este tiempo han tomado un cuarto de millón de fotografías, recorrido 21 kilómetros, subido montañas, explorado cráteres, sobrevivido a tormentas de arena y han dejado más de 36 gigabytes de información a la NASA. Por el momento siguen operativos y veremos lo que les depara el futuro.

Pero también tenemos ejemplos de robots al alcance de todo el mundo y que son muy útiles reemplazando al ser humano en tareas cotidianas. No es necesario irse al mundo de la exploración espacial para poder ver resultados. Un robot que ha alcanzado cierta fama es el robot *Roomba*, fabricado por la empresa *iRobot*. Este pequeño aparato consiste en un robot aspirador autónomo. Es capaz de navegar por una habitación, limpiarla y, cuando es necesario o cuando ha terminado el trabajo, volver él solo a la estación de carga.

Figura 1.3: El robot *Roomba* en acción.

1.2. Visión en robótica

La visión computacional es una rama de la inteligencia artificial. En términos sencillos, lo que busca esta ciencia es conseguir que una máquina sea capaz de usar una cámara, entendiendo y usando esa información para el fin que sea.

La visión artificial es una disciplina relativamente nueva. Surge a partir de la idea de conectar una cámara a un ordenador. Uno de los primeros trabajos realizados en este campo es el programa 'mundo de micro-bloques', creado por Larry Roberts (creador de ARPANet) en 1961. Esta aplicación analizaba una estructura de bloques situada encima de una mesa y la reproducía desde otra perspectiva. De esta manera demostraba que era capaz de extraer información de los datos de la imagen y, en cierta medida, comprenderla.

En los últimos años el precio de las cámaras ha bajado bastante. Este abaratamiento comenzó a mediados de la década de 1990 al reemplazar la tecnología CCD por tecnología CMOS. Los sensores CCD tienen mayor sensibilidad a la luz, más calidad y también un precio más elevado. En cambio, los sensores CMOS son menos sensibles y de menor calidad, pero al ser fáciles de fabricar son bastante más baratos. Tradicionalmente, las cámaras profesionales y semiprofesionales usan tecnología CCD, mientras que las cámaras baratas y de aficionado cuentan con tecnología CMOS. A favor de esta última tecnología cuenta también el consumo energético. Se ha optimizado tanto esta característica que un sensor CMOS consume, más o menos, unas 100 veces

menos que un sensor CCD[1]. Esto le da una clara ventaja en dispositivos móviles, donde el tiempo de autonomía de la batería es fundamental.

Figura 1.4: Una cámara web actual.

Gracias a estos avances tecnológicos, el uso de las cámaras con los ordenadores se ha popularizado enormemente. Por ejemplo, la videoconferencia es un servicio que se ha visto muy beneficiado por estos cambios. La robótica también se ha beneficiado por esto. La mayoría de los robots que ahora mismo están en el mercado, llevan una o varias cámaras. Y es más, en muchos de ellos la cámara es el sensor principal. Un par de ejemplos de este tipo de robots son el humanoide *Nao* y el perrito *Aibo*. Se pueden ver en la figura 1.5. Estos dos robots en concreto, son muy populares en proyectos de investigación gracias a la relación calidad/precio que tienen. De hecho, en el departamento de Robótica de la URJC disponemos de varios.

Poco a poco, las cámaras se han convertido en una parte muy importante de los robots. Existe una competición internacional llamada *Robocup*[2]. La competición consta de varias pruebas, siendo una de las más populares, el partido de fútbol. En él, se enfrentan dos equipos de tres jugadores. Uno es el portero y los otros dos jugadores de campo. El objetivo es sencillo: ganar el partido. Por supuesto, gana el equipo que más goles marca. En esta prueba la cámara del robot adquiere un carácter fundamental. Gracias a ella, el robot es capaz de localizar la pelota, localizar el resto de jugadores, localizar las porterías, autolocalizarse en el campo, etc. La Universidad participa todos los años en esta competición.

[1] http://blog.fotolia.com/es/archive/001654.html
[2] http://www.robocup.org/

Figura 1.5: A la izquierda, el humanoide Nao con la pelota con la que se juegan las competiciones y, a la derecha, el perrito Aibo.

1.3. Atención visual en robots móviles

Dentro de la visión computacional se encuentra la atención visual. Una imagen contiene muchísima información, pero para una máquina procesar toda esa información es muy costoso. Ahí es donde surge la atención visual. Se encarga de seleccionar las zonas o puntos de interés. Por ejemplo, podemos usar un filtro de bordes para que sólo nos llame la atención la silueta de los objetos o usar un filtro de color para solamente trabajar con los puntos de un color determinado.

Por ahora, el mejor sistema de visión conocido es el humano. Las cámaras son los ojos y el cerebro, el ordenador. El ojo humano contiene en la retina unas células nerviosas especiales que, al chocar un fotón con ellas, hacen que este se descomponga y produzca una señal bioeléctrica. Esta señal es transmitida al cerebro por los nervios ópticos y luego interpretada. Los fotorreceptores se dividen en bastones, que permiten la visión en blanco y negro, y en conos, que permiten ver en color. La retina tiene una mayor concentración de fotorreceptores en su parte central. A esta zona se le

conoce como fóvea, figura 1.6. A medida que nos vamos alejando de ella la densidad de fotorreceptores va disminuyendo. Esto provoca que tengamos mucha nitidez en el centro de la imagen y, de alguna manera, quede desatendida la zona externa o periferia. De esta manera conseguimos filtrar una cantidad enorme de datos e interesarnos únicamente en la parte central.

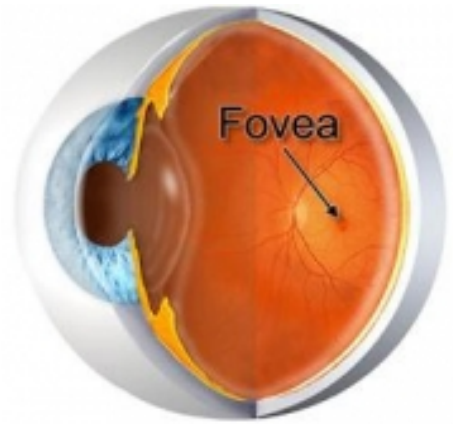


Figura 1.6: La fovea es la parte central del ojo humano, donde hay una mayor concentración de fotorreceptores.

Los *movimientos sacádicos* son movimientos muy rápidos, cortos y precisos. Son los que realiza el ojo para obtener la mayor cantidad de información de la escena. Cabe resaltar que, cuando el ojo está moviéndose, no se recibe ningún estímulo a través de él, es como si se cerrase. En la figura 1.7 podemos ver un experimento muy curioso, el recorrido que realiza un sistema de atención humano cuando analiza una imagen. Este experimento fue realizado por Alfred Yarbus[3] en 1967. El experimento consiste en registrar los movimientos oculares de los participantes al contemplar un cuadro. Éstos disponían de tres minutos para observar una escena sobre la que previamente conocían preguntas a las que debían responder. La idea estaba en que observaran la imagen del cuadro con el objetivo de buscar pistas que les ayudasen a conocer la respuesta.

Los resultados no pueden ser más esclarecedores. A diferentes cuestiones, diferentes patrones de exploración ocular sobre la obra.

[3]http://en.wikipedia.org/wiki/Alfred_L._Yarbus

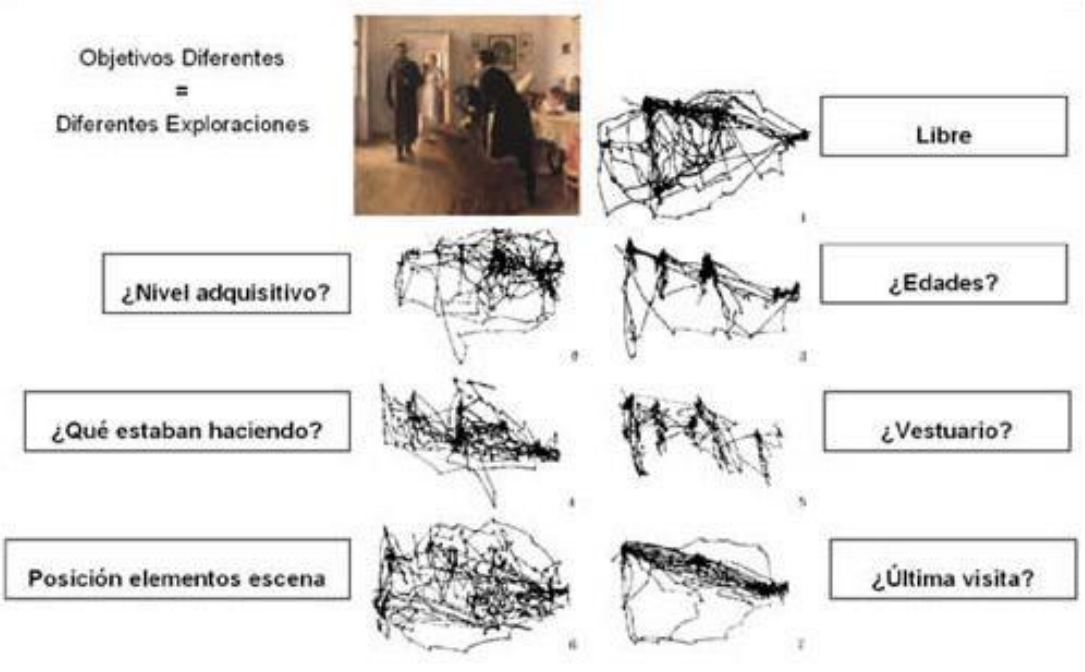


Figura 1.7: Experimento de Yarbus realizado sobre un cuadro. Se puede ver que a diferentes cuestiones, diferentes patrones de exploración ocular sobre la obra.

Otro experimento muy peculiar es el realizado por la Universidad de Illinois[4] para evaluar la capacidad de atención y concentración de una persona. La prueba consiste en ver un vídeo donde hay dos equipos, uno de blanco y otro de negro. Cada uno de estos equipos se pasan un balón de baloncesto entre ellos y el participante tiene que contar el número de pases que hace el equipo blanco. La mayoría de la gente, no se dará cuenta de que a mitad del vídeo aparece un gorila que pasa por el medio del grupo. Esto muestra el grado de atención que es capaz de generar el sistema de atención humano para abstraerse de cualquier situación no relevante de la escena. Gracias a esto, nos concentramos en lo que nos interesa, descartando el ruido de la imagen.

En la robótica, es muy importante realizar un control de atención visual. Las cámaras proveen un flujo de datos continuo enorme, del que hay seleccionar lo que nos resulta interesante e ignorar el resto, esto se conoce como atención selectiva. Existen dos vertientes, la atención local y la global. La atención local o *covert attention* consiste en seleccionar los datos interesantes de la imagen. De este modo las cámaras están fijas y sólo pueden ver los objetos que tienen enfrente. La atención global o *overt attention* consiste en seleccionar del entorno que rodea al robot aquellos objetos que interesan,

[4] http://ibasque.com/experimento-psicologico-de-atencion-visual/

y a los que se dirigirá la mirada [Cañas Plaza, 2005].

Un ejemplo de un sistema de visión es el *Ojo de halcón*, utilizado en los grandes premios de tenis. Mediante ocho cámaras colocadas alrededor de la pista calcula la posición de la bola cada cuatro milímetros. Sirve de ayuda a la toma de decisiones de los árbitros en jugadas polémicas y es muy útil en la recogida de estadísticas de los partidos.

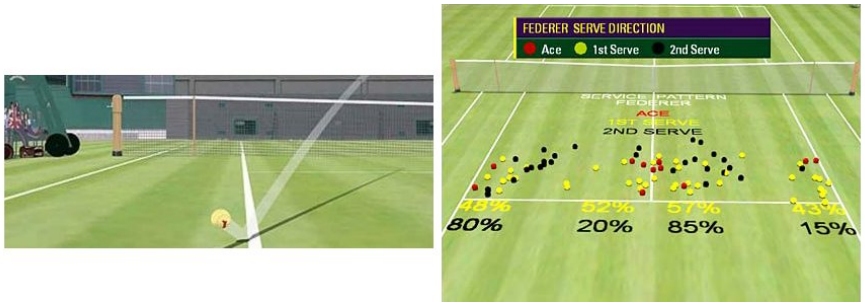


Figura 1.8: A la izquierda, una imagen del *Ojo de halcón* y, a la derecha, recogida de estadísticas de un partido.

La Universidad dispone de un grupo que se dedica a la elaboración de proyectos de visión artificial. Se llama Grupo de Reconocimiento Facial y Visión Artificial (FRAV)[5]. Los principales temas de investigación que trabajan son el reconocimiento facial, la visión artificial aplicada al tráfico y la seguridad vial y la visión artificial para seguridad aeroportuaria. Se pueden ver más detalles de sus proyectos en su página web.

1.4. Reconstrucción en tres dimensiones con movimiento de cámaras

La motivación de este proyecto es crear un sistema que represente en tres dimensiones los objetos interesantes de una escena. Esta información luego puede ser usada como entrada perceptiva de un robot móvil.

En el Grupo de Robótica de la Universidad Rey Juan Carlos se han abordado problemas de visión artificial, localización visual y sistemas de atención desde diferentes puntos de vista. En un principio, se investigó un sistema de atención con una única

[5]Face Recognition & Artificial Vision Group - http://www.frav.es/principal.html

cámara que mediante movimientos sacádicos se 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 triangulación y segmentación sobre imágenes simuladas [Pacios y Cañas Plaza, 2007] y sobre imágenes reales [Mendoza Baños, 2008]. También existe una línea de investigación que reconstruye objetos en tres dimensiones realizando algoritmos evolutivos [García Martínez, 2007].

Siguiendo un poco esta línea de investigación de reconstrucción en tres dimensiones planteamos el presente proyecto. En esta investigación, estudiamos un nuevo algoritmo para la reconstrucción en tres dimensiones basándonos en el movimiento de las cámaras. De este modo, planteamos una alternativa al sistema clásico del cálculo de profundidad por triangulación desde pares estéreo, creando una línea de investigación nueva sobre pares estero de cámaras móviles. El algoritmo lo explicaremos en profundidad más adelante.

Esta memoria detalla todos los aspectos relevantes del desarrollo de esta investigación. La memoria está dividida en seis capítulos. El segundo capítulo trata de los objetivos 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 apoyar y defender nuestra investigación. Por último, en el sexto capítulo se presentan las conclusiones extraídas de este estudio y se proponen posibles líneas futuras de investigación.

Capítulo 2
Objetivos

Una vez presentado el contexto en el que se sitúa el siguiente proyecto, pasamos a explicar los objetivos que se pretenden cumplir en esta investigación. En este capítulo, además de los objetivos, abordaremos otros puntos como los requisitos, la metodología y la planificación.

2.1. Descripción del problema

El objetivo principal del proyecto es el de diseñar y mantener una aplicación *software* que permita a un robot dotado de cámaras móviles crear y mantener una representación en 3D de los objetos interesantes que se encuentran en una escena determinada mediante un algoritmo de atención visual 3D.

El objetivo principal lo hemos dividido en varios subobjetivos para reducir la complejidad del problema:

1. **Sistema de atención 2D**: El sistema de atención 2D debe ofrecer información sobre los objetos que llaman la atención en la imagen. Realizará una atención selectiva sobre la imagen para hacer más liviano el cómputo del algoritmo. El sistema de atención 2D será configurable mediante características visuales *ascendentes* que utilizan como fuente las propias imágenes para determinar qué objetos producen atención.

2. **Reconstrucción 3D**: La reconstrucción 3D se hará mediante reparto de mirada entre puntos interesantes utilizando el movimiento de cámaras. Evitaremos utilizar el clásico algoritmo de reconstrucción 3D de cálculo de la epipolar y triangulación. Se intentará que el algoritmo opere en el espacio 3D.

3. **Atención 3D**: La aplicación implementará un sistema de atención 3D que se encargue de gestionar la memoria 3D y la mantenga viva. La atención de este sistema de debe ser global, al contrario que en el sistema de atención 2D, que

es local. Se facilitará el manejo de distintas primitivas perceptivas con diferente grado de abstracción. También incorporará características de atención cognitiva de alto nivel donde se haga un razonamiento espacial para realizar hipótesis de puntos 3D que proporciona puntos de interés en tres dimensiones a los que merece la pena mirar.

Todos los bloques en los que hemos dividido el proyecto deberán funcionar armoniosamente en conjunto para, entre todos, formar un sistema atentivo completo. Además, realizaremos numerosas pruebas y experimentos para comprobar el correcto funcionamiento de los algoritmos empleados y validar experimentalmente la solución desarrollada.

2.2. Requisitos

El desarrollo de la investigación, siguiendo la línea marcada por los objetivos, deberá ajustarse y cumplir los siguientes requisitos:

1. El lenguaje de programación usado será C ó C++, y la implementación se hará en forma de esquema de *JDE*.

2. Se ejecutará sobre un sistema GNU/Linux, pudiéndose elegir libremente la distribución. De todas formas, se aconseja la distribución basada en Debian *Ubuntu*.

3. Las imágenes tendrán una resolución de 640 píxeles de ancho por 480 píxeles de alto para aumentar el nivel de detalle de las imágenes.

4. Se utilizarán cámaras móviles.

5. La reconstrucción tridimensional se realizará a través de un algoritmo que se aleje del mecanismo clásico de reconstrucción 3D mediante cálculo de epipolar y triangulación.

2.3. Metodología y planificación

El plan de trabajo escogido es el *modelo de desarrollo en espiral basado en prototipos*. Este modelo consiste en dividir en pequeñas subtareas la planificación del proyecto. La mayor ventaja de este modelo es la flexibilidad frente a cambios en las especificaciones. En trabajos de investigación es una cualidad muy útil ya que muchas

veces no conoces el resultado *a priori*.

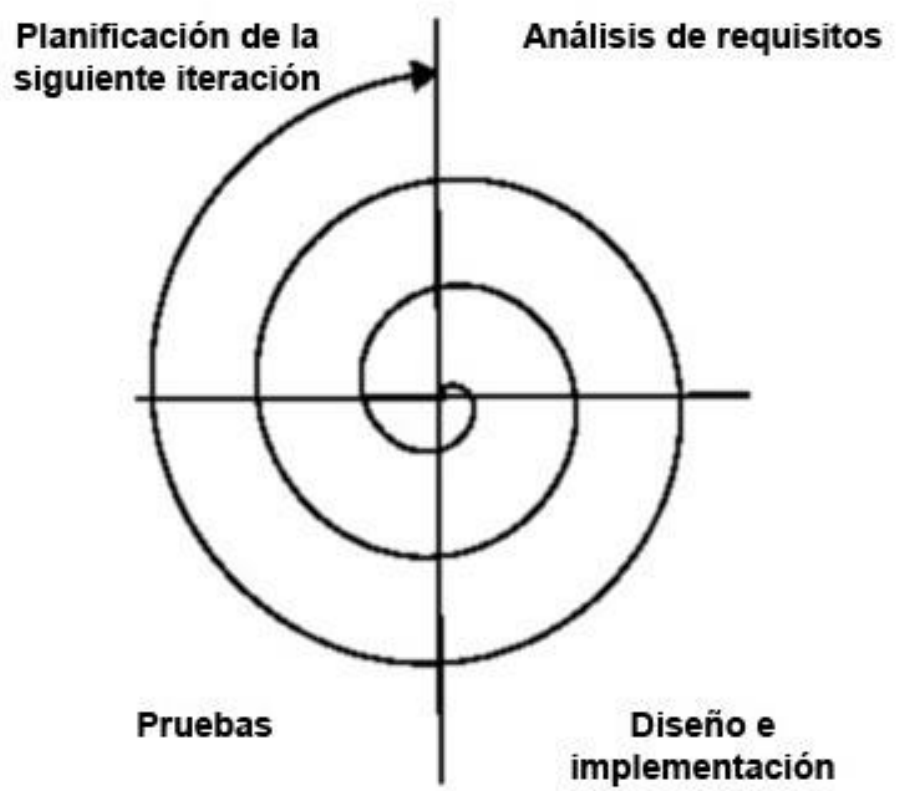


Figura 2.1: Modelo de desarrollo en espiral

Para nuestro proyecto hemos decidido utilizar esta planificación debido a que nos presenta más ventajas que inconvenientes. Para resolver el problema citado anteriormente, tenemos el asesoramiento continuo del tutor. Además, se mantendrán reuniones semanales con él para comentar los resultados obtenidos y planificar los siguientes objetivos.

La planificación es una parte muy importante del proyecto. Conociendo la base de la que se parte y el objetivo final de la investigación, el proyecto se divide en pequeñas subtareas más asequibles. La planificación debe marcar la ruta principal a seguir, pero al mismo tiempo, permitir cierta flexibilidad, ya que no conocemos el resultado final al que llegaremos.

Nuestro proyecto lo hemos dividido en varias etapas de diversa índole, desde tareas de familiarización y aprendizaje del entorno, que en realidad no aportan ningún dato al

proyecto, hasta tareas de investigación. Todo estos pasos han quedado reflejados en una bitácora[1]. En ella hemos ido poniendo los avances más significativos de la investigación.

Las fases en las que dividimos el proyecto antes de comenzarlo, son las siguientes:

- **Fase 1.** En la primera fase, como es lógico, se quiere aprender a usar la plataforma de programación de *JDE* y el hardware que posteriormente utilizaremos para el proyecto. Asimismo, se debe realizar un esquema de aprendizaje para familiarizarnos con el uso de elementos comunes en la programación de algoritmos de visión. El esquema resultado puede verse en el cuaderno de bitácora del proyecto y también puede ser descargado el código[2] desde el repositorio SVN[3] de *JDE*.

- **Fase 2.** Se debe comenzar con el esquema principal. En una primera instancia nos centraremos en la parte de atención visual. Se deben estudiar los proyectos base de los que partimos para ver qué partes se pueden reutilizar. En los proyectos anteriores se han usado imágenes más pequeñas por lo que lo más probable es que haya que revisar el código y portarlo para que cumpla nuestros requisitos.

- **Fase 3.** En esta fase se adaptará el esquema para su uso con el par estéreo de cámaras. Implementaremos el sistema de atención para que filtre y nos devuelva las zonas de interés de la imagen. Al mismo tiempo aprenderemos cómo funciona el calibrador presente en la suite de *JDE* y lo utilizaremos. En esta fase seguramente tengamos que modificar las herramientas utilizadas para poder usarlas con nuestro tamaño de imágenes de 640x480. Todos los esquemas realizados anteriormente usan un tamaño de 320x240. Entre otras tareas haremos una réplica del proyecto de Roberto Calvo [Calvo Palomino, 2008] sobre reconstrucción de una escena tridimensional usando un sistema de atención visual y cámaras fijas. Nos servirá como guía de aprendizaje y como referencia a la hora de realizar las pruebas..

- **Fase 4.** Estudiaremos como implementar un modelo de movimiento tridimensional para el cuello de las cámaras. Este modelo debe relacionar las magnitudes físicas de mesura del cuello mecánico con las coordenadas cartesianas tridimensionales.

[1] http://jde.gsyc.es/index.php/Gdago
[2] http://svn.jde.gsyc.es/users/gdago/project
[3] http://es.wikipedia.org/wiki/Subversion

- **Fase 5.** En esta fase implementaremos un algoritmo para calcular puntos en tres dimensiones utilizando el movimiento de cámaras. Se realizarán pruebas exhaustivas para comprobar el correcto funcionamiento del algoritmo.

- **Fase 6.** En esta última fase, con la capacidad de seleccionar y reconstruir puntos en 3D del sistema, implementaremos el sistema de atención 3D y la memoria 3D para gestionar los puntos ya reconstruidos. También se implementarán funcionalidades tales como la atención cognitiva o la reconstrucción de segmentos.

Capítulo 3
Entorno y plataforma de desarrollo

En este capítulo explicaremos los recursos hardware y software que hemos necesitado en el proyecto. Hablaremos tanto de bibliotecas populares en la comunidad de software libre, como de otras desarrolladas y usadas dentro del grupo de robótica.

El sistema operativo escogido es GNU/Linux, en concreto la distribución de Ubuntu 8.04 LTS (Hardy Heron). LTS (*Long Term Support*) significa que la distribución tiene soporte de larga duración, hasta octubre de 2011. Ubuntu, es una distribución enfocada a computadores personales y se ha convertido en los últimos tiempos en una de las distribuciones Linux más importantes a nivel mundial. Su éxito radica en que se concentra en la facilidad y libertad de uso y la fluida instalación en prácticamente cualquier ordenador. Es la distribución que está instalada actualmente en las máquinas del laboratorio de robótica[1].

El lenguaje utilizado para el software de nuestra aplicación es C y C++. Lo hemos escogido porque la suite de JDE, en la que se apoya la aplicación de este proyecto, está escrita en estos mismos lenguajes. Además, al ser el lenguaje más popular en el grupo de investigación, la reutilización de código de otros componentes e integración de partes se vuelve mucho más sencilla.

3.1. Plataforma Hardware

Como hemos comentado, en este proyecto hemos desarrollado mecanismos de atención visual en dos y en tres dimensiones para robots. El algoritmo de reconstrucción 3D se basa por completo en el movimiento de las cámaras del robot. En la figura 3.1 se puede ver el robot con el que hemos trabajado.

[1] http://robótica-urjc.es

Figura 3.1: El robot con las cámaras montadas.

En la figura 3.2 se puede ver un diagrama de cómo se conecta el sistema. Todo ello esta montado sobre el *Robot Pioneer*. Este robot es muy completo, tiene multitud de sensores y motores, pero no nos han hecho falta porque solamente hemos probado nuestro algoritmo en entornos estáticos. El robot lleva un microprocesador Siemens 88C166, que funciona a 20 MHz. Los actuadores principales son dos motores independientes de corriente continua conectados a las ruedas del robot. Los sensores de que dispone, son un anillo de sónares en la parte delantera y *encoders* en las ruedas.

Hemos probado el algoritmo en el portátil. El portátil es un Dell Inspiron 8600 y lleva un Intel(R) Pentium M 1.6GHz como procesador. No obstante hemos desarrollado y hecho las pruebas de la aplicación en una máquina que lleva el microprocesador Intel(R) Pentium Core(TM)2 Quad Q9300 @ 2.50GHz. Todo el software desarrollado en el proyecto se ejecuta en la máquina principal, por eso hemos escogido la máquina más potente para realizar las pruebas. Aunque también hemos realizado pruebas de concepto el portátil para comprobar la validez del algoritmo sobre esta plataforma.

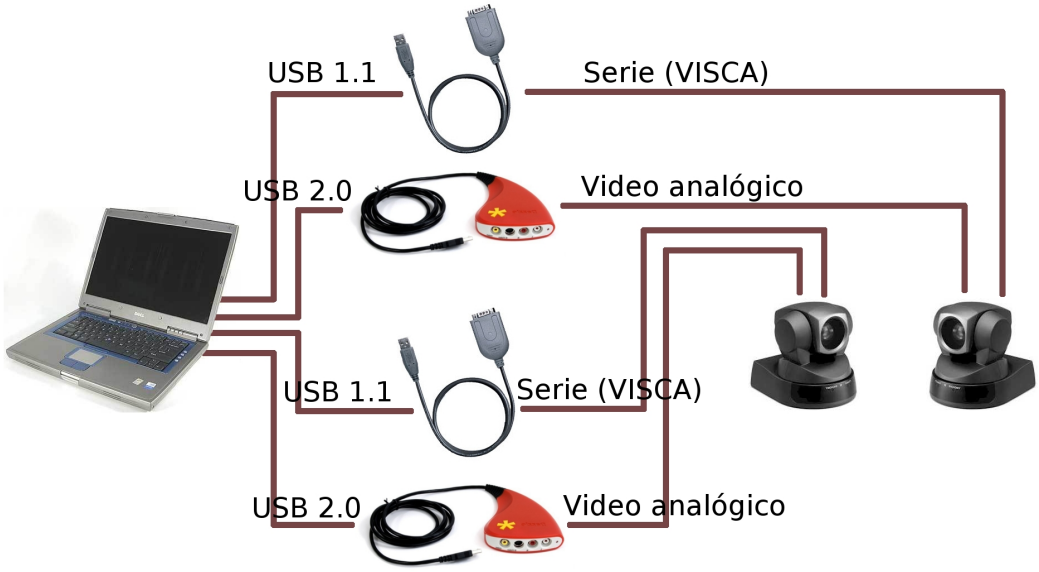


Figura 3.2: Diagrama de conexión del sistema.

Para poder ejecutar nuestro algoritmo, como mínimo necesitamos una máquina con dos buses independientes USB2.0 para poder procesar las imágenes a 640x480 y un bus USB1.1 ó 1.0 para controlar el movimiento de las cámaras. La aceleración gráfica no es imprescindible, pero sí muy recomendable, ya que libera a la CPU de mucha carga haciendo uso de la GPU a la hora de pintar imágenes tridimensionales. Por limitaciones de hardware, en el portátil hemos tenido que capturar las imágenes de las cámaras con un tamaño de 320x240, ya que sólo disponemos de un bus USB2.0 y el ancho de banda no es suficiente para soportar el tráfico de imágenes de las dos cámaras.

El modelo de cámara utilizado para el proyecto es la Sony EVI-D100P. Hemos escogido esta cámara por tener un cuello mecánico muy rápido y con dos grados de libertad, horizontal y vertical. También tiene funciones de zoom que no hemos utilizado para el proyecto, pero que en futuras mejoras podrían usarse.

De todas las cámaras disponibles en el mercado, pese a las desventajas descritas anteriormente, hemos escogido este modelo por su velocidad de giro. En horizontal tiene una amplitud desde -100º hasta 100º y puede alcanzar una velocidad máxima de 300º/s. En vertical llega desde -25º hasta 25º y alcanza una velocidad de 125º/s como máximo. Los movimientos rápidos y precisos de la cámara son vitales para un funcionamiento

correcto y rápido de los algoritmos de emparejamiento 3D con movimiento de cámaras.

El protocolo de control que usan las cámaras se llama *VISCA*. Es un protocolo inventado por Sony sobre el puerto serie. Las especificaciones de este protocolo se pueden ver en el manual de la cámara[2]. Como en nuestra máquina sólamente disponemos de un puerto serie, hemos utilizado un conversor de puerto serie a USB.

Figura 3.3: A la izquierda, imagen tomada con la webcam *Logitech Quickcam Pro 9000*; A la derecha, imagen tomada con la *Sony EVI-D100P*. En la izquierda de las imágenes se puede apreciar la distorsión introducida por la cámara en un palo recto.

Una de las desventajas de esta cámara es la calidad de la lente de la cámara. Produce gran distorsión radial en los bordes de la cámara. Según las especificaciones técnicas la distorsión es menor del 5 %, pero como se puede apreciar en la figura 3.3 la distorsión es claramente visible. Esto se puede traducir en una mala representación en tres dimensiones de los puntos que caen sobre esas zonas.

Una de las desventajas de esta cámara es que la salida de vídeo es analógica, y para conectarla al ordenador es necesario que pase por una capturadora de vídeo analógico. La capturadora de vídeo analógico que convierte la señal analógica en una señal digital y, por tanto, procesable por la máquina. Los interfaces habituales son por bus USB o por ranura PCI. Hemos elegido un modelo con interfaz USB para poder probar la aplicación en máquinas que no tengan ranuras PCI como son los portátiles, muy habituales en los robots. Tenemos una limitación importante respecto a la velocidad del bus USB: necesitamos que las capturadoras se conecten cada una a un USB 2.0 independiente.

[2]http://www.sony.es

Si no, el ancho de banda no es suficiente para procesar tamaños de imagen de 640x480 píxeles en tiempo real.

3.2. Plataforma JDE

La plataforma elegida es *JDE*[3], una suite de desarrollo de aplicaciones robóticas escrita en C. Proporciona un entorno de programación donde el control del robot se realiza a través de una colección de hilos asíncronos que funcionan concurrentemente y que se llaman *esquemas*. Facilita la comunicación con el hardware del robot mediante interfaces simplificadas. De hecho, los componentes software se pueden dividir en *drivers*, que son los encargados de comunicarse a bajo nivel con los sensores y los actuadores; *servicios*, son componentes con una funcionalidad especial, como las interfaces gráficas o un servidor *proxy* de imágenes; y *esquemas*, que son las aplicaciones que determinan el comportamiento del robot.

Los *drivers*, dan de alta esquemas virtuales y cada uno con su propia interfaz que nos simplificarán el acceso a los sensores y actuadores. Unos facilitan las lecturas de los sensores, mientras que otros facilitan el envío de órdenes a los actuadores.

Una aplicación de JDE es un conjunto de esquemas que se comunican entre sí. Los esquemas son componentes activables y desactivables a voluntad, que se ejecutan de forma iterativa. El ritmo de las iteraciones es controlado independientemente para cada esquema y *driver* a través de una variable que define la duración de cada ciclo. Esto permite tener mayor control del sistema, pudiendo ajustar la velocidad de respuesta de los sensores y actuadores a las exigencias del esquema.

Los esquemas se comunican entre sí intercambiándose datos a través de variables compartidas que se agrupan en interfaces. Estas variables compartidas, en ocasiones pueden dar lugar a condiciones de carrera, pero gracias a la naturaleza iterativa de los esquemas esa irregularidad es resuelta en la siguiente iteración sin apenas consecuencias. En el caso de que la variable fuese crítica, siempre se puede proteger con un semáforo o cualquier otra técnica de concurrencia. Pese a todo, la ventaja más relevante que proporciona esta interfaz es la facilidad de su uso a la hora de programar.

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

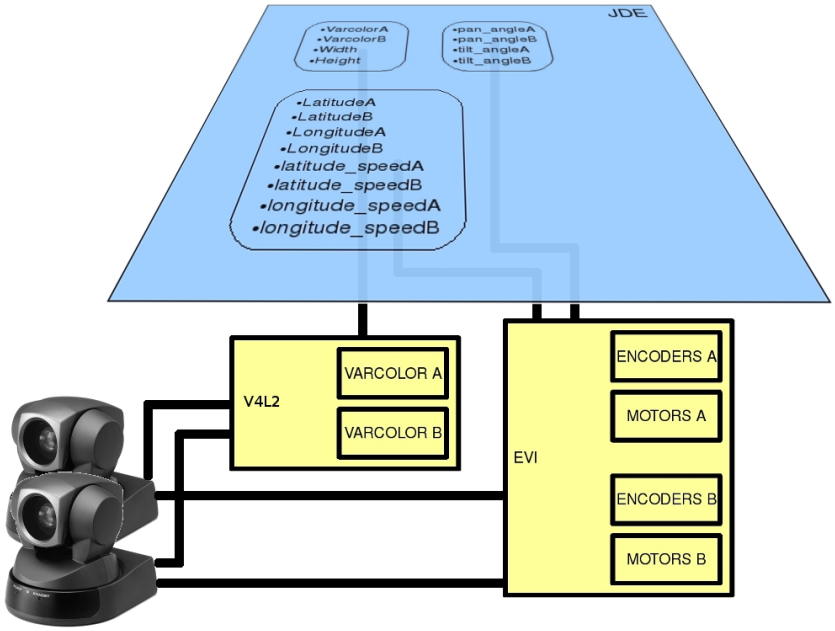


Figura 3.4: Diagrama de conexión entre esquemas de nuestra aplicación en la plataforma de JDE.

Las aplicaciones de JDE se pueden ver como un conjunto de componentes. En nuestro caso concreto la aplicación está formada por un esquema principal, *vergency*, y por los distintos *drivers* que nos facilitan el control del sistema. Para captar las imágenes de la cámara usamos el driver *video4linux2*. Este driver lo hemos creado para la realización del proyecto y añadido posteriormente a la suite de JDE. Hablaremos más en detalle sobre este *driver* un poco más adelante. Para controlar el cuello mecánico de las cámaras hemos utilizado el *driver evi*. Este driver nos divide el control del cuello en dos esquemas virtuales, los *encoders*, que nos permiten ver el estado del cuello en cada momento; y los *motors*, que nos permiten mandar órdenes al cuello para que las ejecute.

En esta plataforma se lleva trabajando ya varios años en el grupo de Robótica de la Universidad y alcanza la versión 4.3, que es con la que hemos trabajado. Las mejoras más destacables de la última versión son la incorporación de nuevos *drivers* para distintos dispositivos y la creación de un repositorio de paquetes Debian para Ubuntu, facilitando su instalación. Se trata de un proyecto de Código Abierto.

A continuación, explicaremos brevemente los componentes y las bibliotecas de JDE que hemos utilizado.

3.2.1. Driver video4linux2

Hemos creado el driver video4linux2 para la plataforma de *JDE*. Este nos simplifica el uso de imágenes de vídeo en nuestro esquema. No nos tenemos que preocupar de cómo conseguir las imágenes de eso se encarga el driver, sino sólamente de procesarlas.

El driver usa a su vez una API integrada en el núcleo de Linux, Video4Linux2. Esta biblioteca se encuentra ya en su segunda versión y sirve para capturar vídeo. Muchas webcams USB, sintonizadores de TV y otros periféricos multimedia son soportados, lo que nos facilita enormemente la tarea.

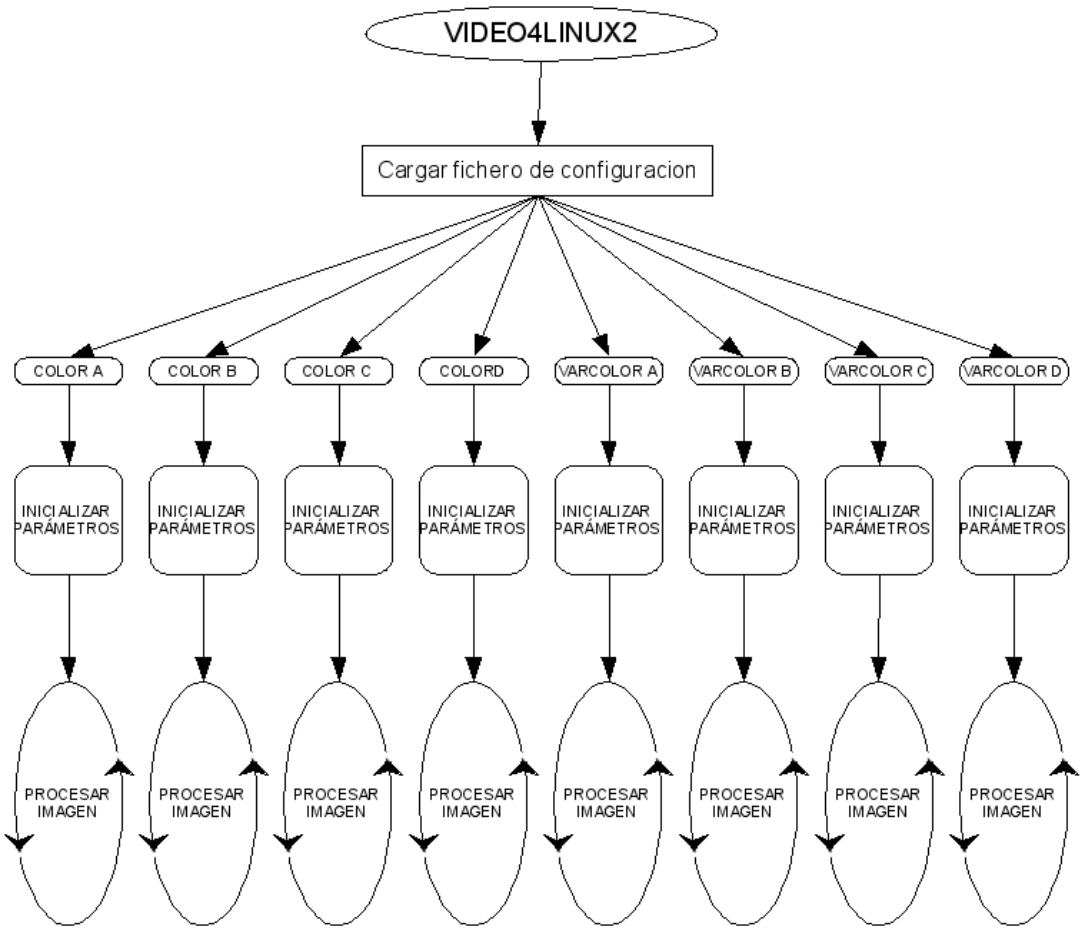


Figura 3.5: Diagrama de bloques de cómo está implementado el driver video4linux2.

El driver soporta *streaming*, que es la capacidad de reproducir contenido multimedia sin descargar el fichero completamente al ordenador. Para hacer más ligero el proceso de capturar imágenes, la transferencia de información entre la cámara y el ordenador se hace por DMA[4]. El *driver* sólo soporta los formatos de video RGB de 24 bits y YUYV[5], que son los formatos más habituales en los que codifican los dispositivos hardware el video. También se puede elegir el canal del dispositivo del que se quiere capturar. En cámaras web, este canal es siempre el cero, pero nuestras capturadoras tienen dos entradas de video: una entrada de video compuesto[6] y otra de S-Video[7]. Por ello hemos implementado el driver para que seamos capaces de seleccionar la entrada que queramos.

La interfaz que presenta el driver es la misma que tiene su predecesor *Video4linux*. En el fichero de configuración se especifica qué dispositivo se relacionará con qué variable. Tenemos disponibles ocho variables distintas divididas en dos grupos. Las variables de tipo *ColorX* tienen un tamaño fijo de 320x240 píxeles y la imagen es almacenada con el formato RGB de 24 bits. La *X* puede ser cualquier letra entre la *A*, *B*, *C* y *D*. El otro grupo está compuesto por las variables *VarcolorX*, la imagen es almacenada en el buffer con el mismo formato y *X* puede tomar los mismos valores. También podemos recibir imágenes de hasta ocho dispositivos, pero si hemos escogido un tamaño grande hay que tener en cuenta el ancho de banda del bus en el que se conecta el dispositivo hardware. La diferencia entre *VarcolorX* y *ColorX* es, que en el fichero de configuración, para *VarcolorX* hay que indicar el tamaño de las imágenes.

El driver también nos aporta una serie de variables que nos dan información sobre las imágenes. Tenemos la variable *width*, que nos indica el ancho en píxeles de la imagen; la variable *height*, que indica la altura en píxeles de la imagen y, por último, la variable *clock*, que indica si la imagen ha cambiado. Se trata de un reloj lógico que cada vez que se actualiza la imagen, este contador se incrementa en una unidad. Gracias a esto, podemos saber de una manera muy simple si la imagen se ha actualizado o aún no.

Para más detalles de este *driver*, se puede consultar la página del manual[8] de JDE.

[4] http://es.wikipedia.org/wiki/Acceso_directo_a_memoria
[5] http://en.wikipedia.org/wiki/YUV
[6] http://es.wikipedia.org/wiki/Vídeo_compuesto
[7] http://es.wikipedia.org/wiki/S-Video
[8] http://jde.gsyc.es/index.php/Manual#Video4Linux2

3.2.2. Driver EVI

El driver EVI nos permite comunicarnos con el cuello mecánico de las cámaras. Para nuestro proyecto hemos utilizado el modelo de cámara EVI-D100P y la comunicación es perfecta, pero este driver también soporta el resto de cámara de la familia EVI. Este driver proporciona cuatro esquemas virtuales distintos: *PTmotors*, *PTencoders*, *zoom_encoders* y *zoom_motors*. Los nombres, por sí solos, son bastante descriptivos. Unos proporciona la interfaz para manejar los motores del cuello mecánico, otro proporciona la interfaz para ver el estado de esos mismos motores, otro proporciona la interfaz para manejar el motor del zoom y el último proporciona la interfaz para ver el estado del zoom.

PTencoders nos proporciona una serie de variables para ver el estado de los motores del cuello mecánico. *Pan_angle* nos da la posición, en grados, del cuello mecánico en el plano horizontal. *Tilt_angle* devuelve la posición, también en grados, del cuello mecánico en el plano vertical. Y la variable *clock* que, al igual en el driver de Video4linux2, nos indica si los datos han cambiado con respecto a la lectura anterior.

PTmotors nos permite enviar órdenes de movimiento al cuello mecánico para que las ejecute. Se hace un control en posición del cuello con velocidad variable. Las variables más importantes son: *longitude*, que nos permite introducir los grados a los que queremos que gire el cuello mecánico en horizontal; *latitude*, igual pero para girar en vertical; *longitude_speed*, indica la velocidad a la que queremos que gire el cuello en horizontal; y por último, *latitude_speed*, indica la velocidad de giro del cuello en grados/segundo verticalmente.

Por último tenemos las interfaces de control del zoom. En nuestro proyecto no hemos utilizado esta capacidad de las cámaras, por lo que no entraremos en detalle las correspondientes interfaces.

3.2.3. Calibrador de cámaras

Para la calibración de las cámaras hemos utilizado el esquema calibrador que viene en la suite de JDE. El calibrador fue creado a partir de un proyecto fin de carrera [Kachach, 2008]. Este calibrador devuelve los parámetros intrínsecos y extrínsecos de las cámaras respecto de un patrón 3D. Para poder realizar los cálculos 3D para los

algoritmos de atención visual es imprescindible tener calibradas las cámaras.

Para poder utilizar el calibrador para nuestra investigación hemos tenido que modificar el esquema para que acepte cámaras con un tamaño de imagen de 640x480 píxeles. Por defecto, sólamente acepta imágenes con un tamaño de 320x240 píxeles.

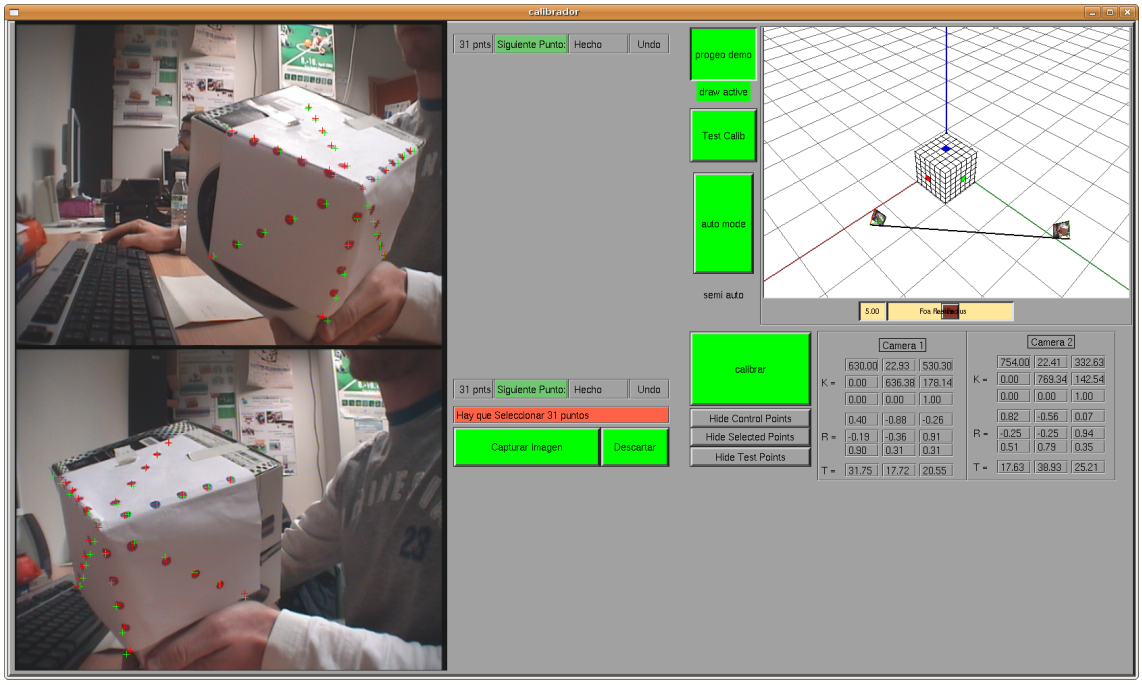


Figura 3.6: Proceso de calibración de las cámaras utilizando el calibrador de JDE.

Para resolver este problema, el esquema utiliza un algoritmo basado en la técnica DLT[9]. A partir de un patrón de calibración 3D del que se conoce perfectamente su geometría y la posición de ciertos puntos significativos, construye un sistema de ecuaciones con la información de correspondencia entre esos puntos, resuelve el sistema optimizando la matriz solución y extrae los parámetros de la cámara descomponiendo esta última matriz solución.

En nuestro proyecto sólamente usamos los parámetros intrínsecos devueltos por el esquema. La calibración de los parámetros extrínsecos la realizamos a mano desde nuestro esquema gracias al ajustador de parámetros extrínsecos que hemos implementado en el mismo. Lo hemos hecho así, porque el origen de coordenadas devuelto por el calibrador original no es válido. Nosotros necesitamos un sistema de

[9]http://en.wikipedia.org/wiki/Direct_linear_transformation

referencia común para las dos cámaras y que sea solidario con el robot para que los cálculos de los puntos en 3D se hagan respecto del robot y no respecto de un patrón de calibración. El calibrador original nos devolvía un sistema de referencia que cumplía el primero de los requisitos, pero no el segundo. En la figura 4.7 se puede ver el sistema de referencia que hemos elegido.

3.2.4. Biblioteca Progeo

Esta herramienta es imprescindible para realizar las triangulaciones para la reconstrucción en tres dimensiones. Se trata de una biblioteca de geometría proyectiva usada para operar en un espacio tridimensional[10]. Está incluida dentro de la suite de *JDE*. Sirve para relacionar el mundo de las imágenes en dos dimensiones con el mundo real, que tiene tres dimensiones. Ha sido muy útil para nuestro proyecto ya que nos ha facilitado muchísimo las relaciones entre el mundo 3D y el plano de las imágenes. Las funciones más importantes dentro de esta biblioteca son:

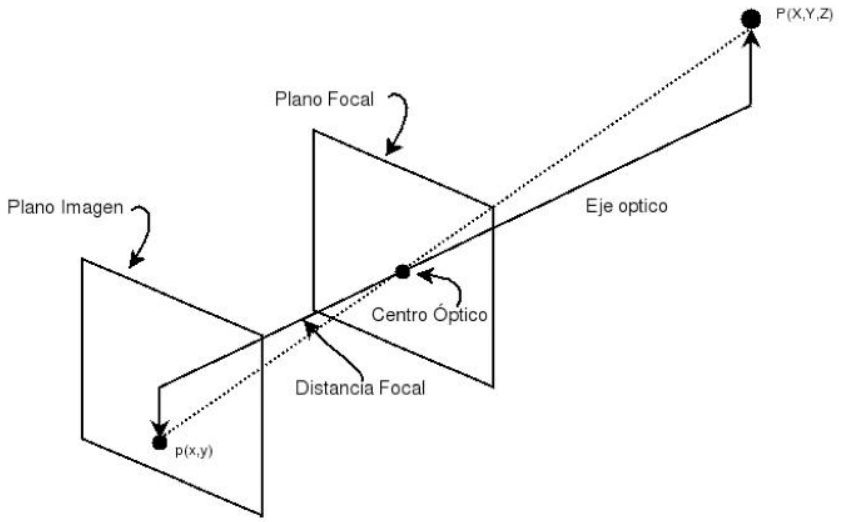


Figura 3.7: Modelo de cámara *pin-hole*.

- Proyectar. Conociendo un punto tridimensional nos permite conocer la proyección geométrica de dicho punto en el plano imagen de la cámara, obteniendo el píxel en el que proyecta.

[10]http://jde.gsyc.es/index.php/Manual

- Retro-proyectar. Conociendo un píxel de la imagen nos permite obtener el rayo óptico en tres dimensiones que contiene el conjunto de puntos que proyectan en ese píxel.

El modelo de cámara que usa esta biblioteca es *pin-hole*. En este modelo se asume que cualquier punto *P(X,Y,Z)* se proyecta en el plano imagen como *p(x,y)* a través de un único punto situado en el plano focal, conocido como centro óptico. El centro óptico es el punto del plano focal que se encuentra exactamente a la distancia focal. La *línea de proyección* es la recta que une el centro óptico con el punto en tres dimensiones. Y por último, para completar el modelo, el eje óptico es la línea perpendicular al plano imagen que atraviesa el centro óptico.

3.3. GTK

GTK es un conjunto de bibliotecas y rutinas para desarrollar interfaces gráficas. Inicialmente fue creada para desarrollar la aplicación de edición de imágenes GIMP. Sin embargo, actualmente es una de las bibliotecas más usadas en los sistemas GNU/Linux. Una alternativa factible es la biblioteca Qt, que es más usada en el entorno de escritorio de KDE. Hemos escogido GTK porque Ubuntu GNU/Linux utiliza como entorno de escritorio por defecto Gnome. Nuestra aplicación hace un uso intensivo de interfaz gráfico para ayudarnos a la depuración de la aplicación y para visualizar los resultados obtenidos.

GTK dispone de editores tipo *WYSIWYG*[11] como *Glade*[12], que simplifica enormemente la creación de interfaces gráficas. Actualmente, todos los proyectos de investigación que están comenzando en *JDE* usan esta biblioteca. Nuestro proyecto utiliza estas bibliotecas para la visualización del esquema.

JDE 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. Este servicio proporciona una interfaz compuesta por tres funciones. La función *show* inicializa todos las variables y parámetros necesarios para cargar la interfaz gráfica. La función *guidisplay* es llamada iterativamente en un hilo distinto del hilo principal de JDE para refrescar la interfaz gráfica. Y por último, *hide* contiene todas las rutinas

[11] What You See Is What You Get - http://es.wikipedia.org/wiki/WYSIWYG
[12] http://glade.gnome.org

necesarias para desactivar la interfaz. De esta manera, la parte gráfica se vuelve independiente del hilo principal de ejecución del esquema. Programar el GUI de un esquema en JDE consiste en definir el interfaz gráfico con sus *widgets* y rellenar estas tres funciones.

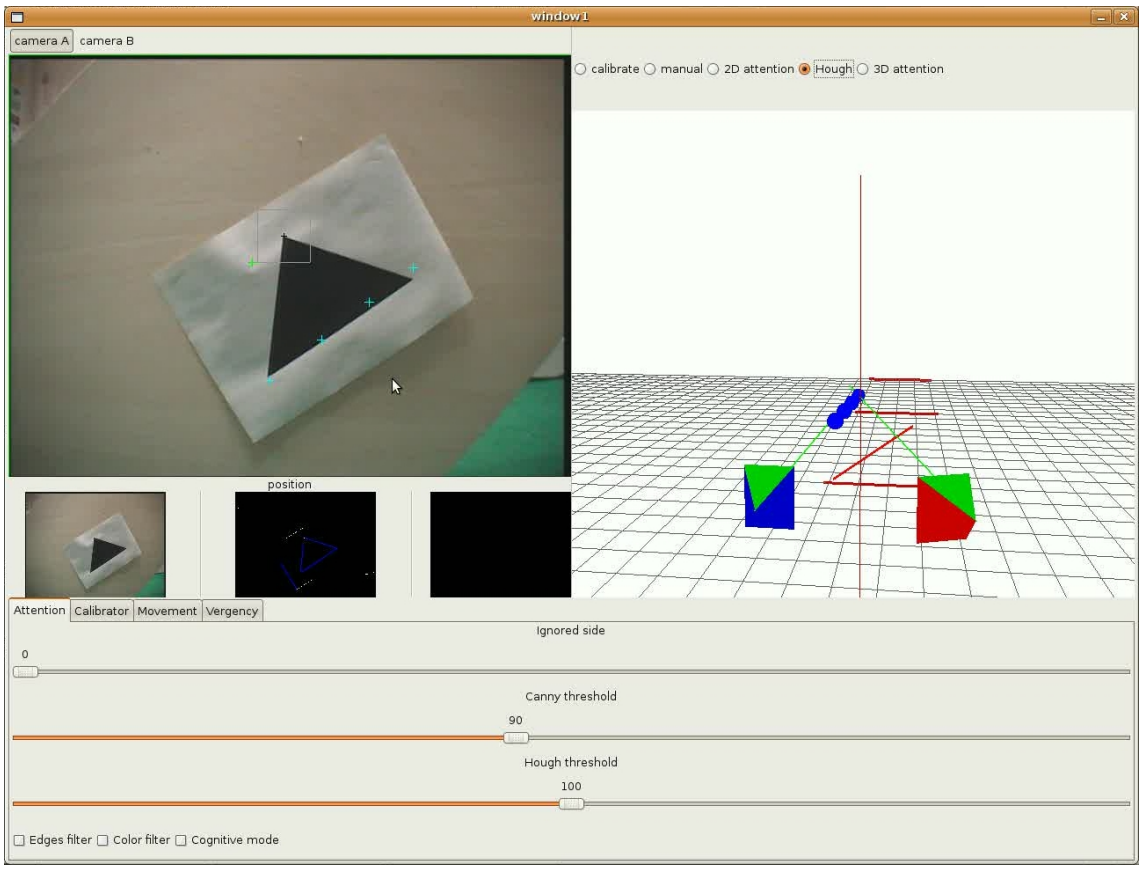


Figura 3.8: El interfaz gráfico de nuestro esquema esta realizado con GTK.

En este proyecto hemos utilizado la versión 2.0 de la biblioteca GTK. Este conjunto de bibliotecas es muy sencillo de instalar. Se puede instalar desde los repositorios oficiales de Ubuntu. Los nombres de los paquetes se pueden buscar en FAQ[13] de JDE.

3.4. OpenGL

OpenGL es una especificación estándar que define un API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos en dos y tres

[13]http://jde.gsyc.es/index.php/FAQ#GTK_Library

dimensiones. La interfaz consiste en rutinas que dibujan escenas tridimensionales complejas a partir de primitivas geométricas simples como puntos, líneas o triángulos. Se usa mucho en aplicaciones CAD, simuladores de vuelo y también en el desarrollo de videojuegos, donde compite con *Direct3D* en plataformas *Microsoft Windows*.

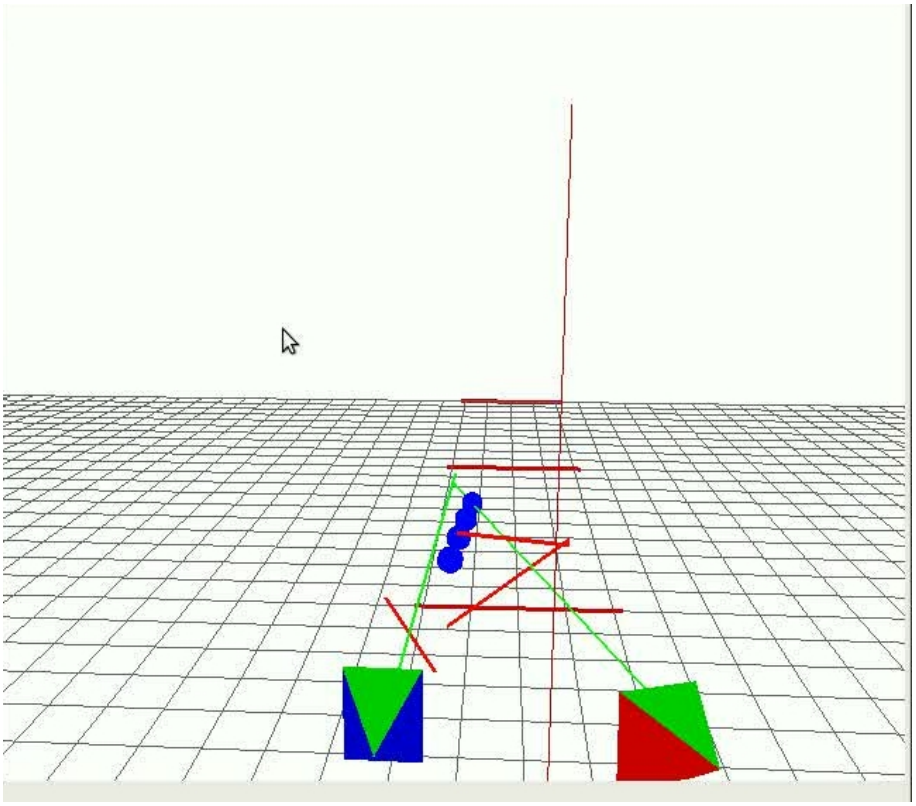


Figura 3.9: Representación de un escenario 3D pintado con OpenGL.

Para la implementación de dicha biblioteca usamos Mesa 3D, que ya alcanza la versión 7.0.3. Es una implementación de Código Abierto de las especificaciones de OpenGL. En nuestro proyecto, lo utilizamos para reconstruir y visualizar la escena 3D creada. Podemos navegar por ella y comprobar la similitud que tiene nuestra representación virtual con el mundo real.

Para instalar este conjunto de bibliotecas se puede hacer desde los repositorios oficiales de Ubuntu.

3.5. OpenCV

OpenCV[14] es una biblioteca de visión computacional desarrollada por *Intel* y lanzada bajo licencia BSD, lo que permite su uso libre, ya sean fines comerciales o de investigación. Es una biblioteca multi-plataforma que funciona bajo sistemas operativos tan diversos como Windows, Mac OS X, Linux y otros sistemas embebidos. Está enfocada al tratamiento de imágenes en tiempo real.

Ha sido utilizada en diversos proyectos, como son el sistema de visión del robot *Stanley*, que montado en un vehículo autónomo no tripulado fue el ganador del Gran Desafío DARPA en 2005. También es usada en sistemas de videovigilancia. En nuestro proyecto, usamos esta biblioteca para ayudarnos en el filtrado de las imágenes para el sistema de atención en dos dimensiones: filtro de bordes, filtro de color, etc.

En nuestro proyecto hemos utilizado la versión 1.0 disponible en los repositorios oficiales de Ubuntu. Los paquetes esenciales para que funciones son: libcv1, libcv-dev, libcvaux1 y libcvaux-dev. Existen más paquetes que añaden mayor funcionalidad a la biblioteca, pero estos son los básicos.

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

Capítulo 4
Descripción informática

Una vez explicados los requisitos y herramientas necesarias para el desarrollo del proyecto, explicaremos en profundidad el software diseñado y programado. Daremos un vistazo general al conjunto en la sección 4.1 y luego desglosaremos cada uno de los bloques en las siguientes secciones.

4.1. Diseño global

Como hemos comentado previamente, el objetivo principal del proyecto es el de crear un sistema perceptivo visual con atención tridimensional que gobierne el movimiento ocular de los dos ojos de un robot, para que el robot perciba la realidad 3D que le rodea usando sus dos cámaras y si es necesario mueva sus ojos para ello. La aplicación debe ser capaz de reconstruir en tres dimensiones los puntos interesantes que aparecen por primera vez a la vista del sistema y, al mismo tiempo, hacer un seguimiento de los puntos ya reconstruidos. La aplicación final engloba tres partes bien diferenciadas. Primero tenemos el sistema de atención 2D que nos devuelve los píxeles más interesantes dentro de la imagen actual de las cámaras. Segundo, el algoritmo de reconstrucción 3D, con el que llevaremos a tres dimensiones los píxeles devueltos por el sistema de atención 2D. Introduciremos estos puntos nuevos en la memoria 3D. Y por último, tenemos el sistema atentivo tridimensional, con el que haremos un seguimiento de los puntos almacenados en la memoria 3D.

En cuanto a la programación de la solución, siguiendo la línea de *JDE*, hemos creado un esquema que recibe las imágenes de las cámaras y decide hacia dónde moverlas en cada momento. El diagrama de conexión entre los bloques puede verse en la figura 3.2. El esquema *vergency* es el componente principal del sistema, es donde está implementada toda la lógica de la aplicación. Los *drivers* simplemente nos facilitan la comunicación con el hardware: la captura de imágenes, la lectura de los *encoders* y

el envío de las órdenes a los motores.

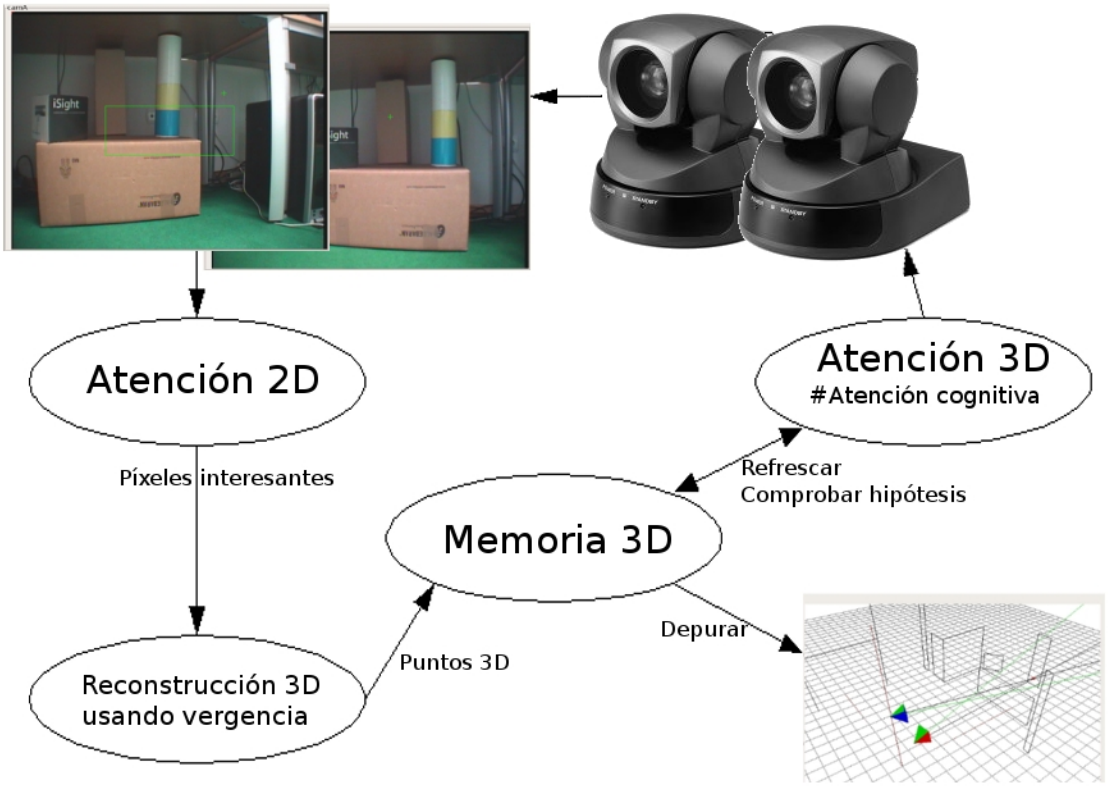


Figura 4.1: Diagrama de bloques del sistema.

El funcionamiento final de nuestro esquema queda reflejado en la figura 4.1. El flujo natural de trabajo comienza con el sistema de atención 2D. Alimentado por las imágenes de las cámaras, el sistema de atención 2D nos devuelve los píxeles más interesantes de la imagen. Este sistema es configurable para que le llamen la atención distintas características de la imagen: color, bordes, etc. El algoritmo de reconstrucción 3D nos reconstruye esos píxeles interesantes en puntos del espacio tridimensional y los almacena en la memoria 3D.

En ocasiones el sistema de atención 2D no nos devuelve ningún píxel porque no hay nada nuevo que nos llame la atención. Es entonces cuando le pedimos al sistema de atención 3D que nos devuelva el punto 3D que conviene mirar en este momento. Este punto puede ser uno ya conocido de la memoria, que aprovecharemos para refrescar si sigue en el mismo sitio de antes, o puede ser una hipótesis (la existencia de un punto en ciertas coordenadas) hecha con el algoritmo de atención cognitiva, en ese

caso verificaremos si el punto se encuentra realmente donde hemos calculado y, en caso afirmativo, lo insertaremos en la memoria. El ejemplo de concreto atención cognitiva que hemos implementado en este proyecto es la obtención de la cuarta esquina de un cuadrado conociendo la posición espacial de las otras tres esquinas. Al contrario que el sistema de atención 2D, el sistema de atención 3D necesita hacer uso del cuello mecánico de las cámaras para poder llevar a cabo su cometido.

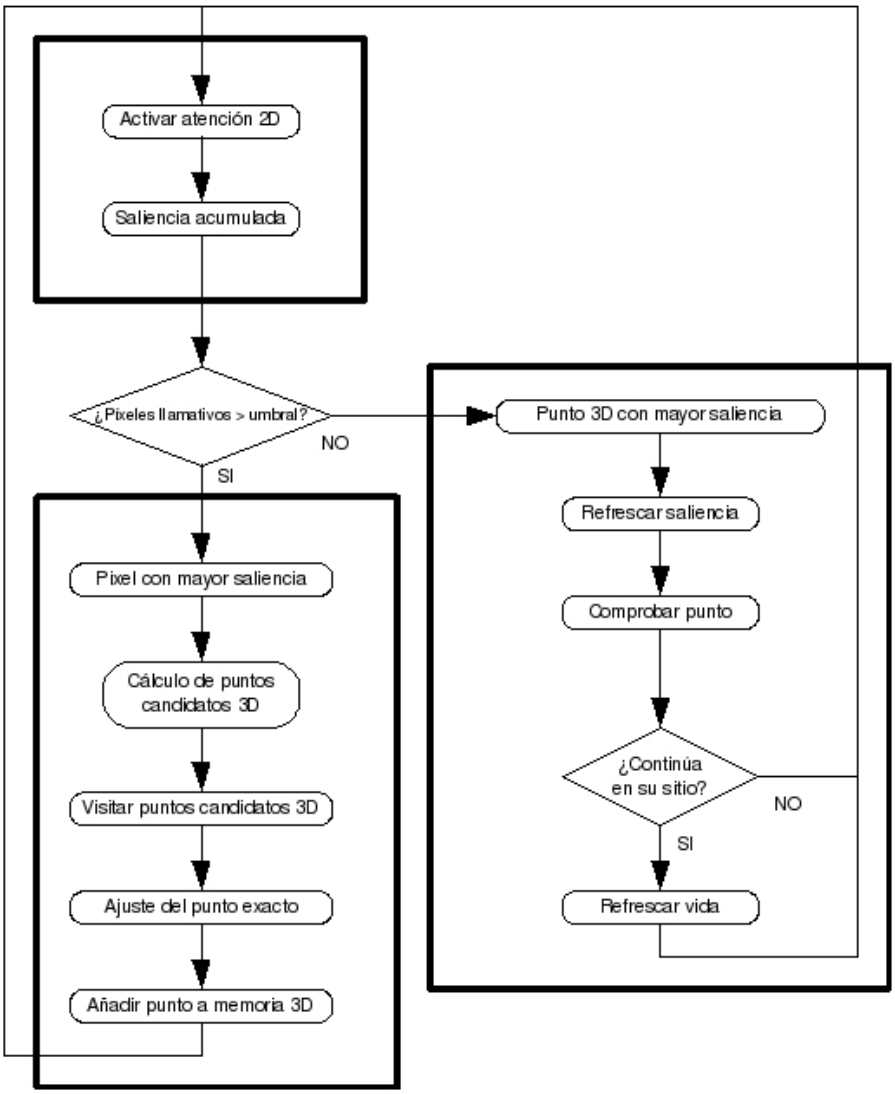


Figura 4.2: Flujo de control del comportamiento del esquema.

Después de dar una visión general de cómo funciona nuestro sistema, pasaremos a detallar las distintas partes del esquema en las distintas secciones del capítulo.

4.2. Sistema de atención 2D

El sistema de atención 2D nos devuelve el píxel más interesante de la imagen en el momento en el que se lo pedimos. En esta sección vamos a explicar los métodos que hemos seguido para recoger los puntos interesantes de la escena. Para resolver el problema de qué zona de la imagen nos resulta interesante hemos reutilizado parte del código escrito por Roberto Calvo [Calvo Palomino, 2008]. A continuación explicaremos los detalles del sistema atentivo 2D que, aunque no hemos programado en su totalidad, hemos tenido que estudiar a fondo el código para poder efectuar los cambios necesarios para que encaje en nuestro proyecto.

Las cámaras son una fuente de información muy rica y densa. Nuestras imágenes tienen un tamaño de 640x480 píxeles, y cualquiera de esos píxeles puede llamarnos la atención. Procesar todos los píxeles sería demasiado costoso computacionalmente, especialmente en un robot que tiene que responder de forma vivaz a estímulos de su alrededor. Por ello necesitamos ayudarnos de un algoritmo atentivo que nos seleccione los píxeles más interesantes en cada momento. Nuestro algoritmo de atención 2D opera en el plano imagen de las cámaras, lo único que llama la atención son los objetos que se ven en las imágenes. A este tipo de atención se le conoce como atención *cerrada* o *covert*.

Las saliencias instantáneas nos devuelven las características más llamativas de la imagen. Estas se pueden combinar entre sí para enriquecer más la información que nos aportan. La inhibición de retorno y la inhibición cognitiva que establecen una dinámica en la que el punto más llamativo, aquel en el que se concentra el procesamiento, se va alternando entre las distintas zonas relevantes de la imagen.

A continuación pasaremos a comentar la *dinámica de saliencia*, técnica que nos permite alternar entre los distintos puntos de atención y saber a dónde dirigir la mirada en el plano de la imagen ahorrando cómputo y concentrando la capacidad de procesamiento en las zonas interesantes.

4.2.1. Saliencias instantáneas

¿Qué es saliencia? Saliencia es todo aquello que nos llama la atención en una situación concreta. Cada píxel de la imagen tiene asociada una determinada saliencia. Cuanto mayor es la saliencia, más interesante nos resulta el píxel. De esta manera

tenemos un factor que nos ayuda a decidir el píxel más interesante en cada momento.

Como hemos comentado anteriormente, el sistema de atención 2D se puede configurar para determinar qué características le llaman la atención a partir del análisis directo de las imágenes (características ascendentes). Incluso podemos activar varias de ellas al mismo tiempo. A continuación explicaremos brevemente las tres características ascendentes que hemos implementado: color, bordes y líneas.

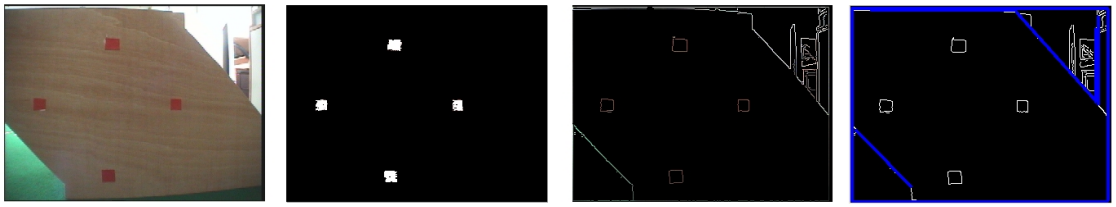
E 

Figura 4.3: De izquierda a derecha tenemos: la imagen original, la imagen después de aplicar un filtro de color, filtro de bordes y filtro de segmentos.

Una característica muy habitual es la de color. Dos de los espacios de color más conocidos son el RGB[1] y el HSV[2]. El RGB se compone gracias a la combinación de tres colores, el rojo, el verde y el azul. El HSV se compone por la saturación (pureza del color), tono (el color en sí mismo) y la intensidad (brillo). Hemos escogido este último espacio de color porque es menos sensible a los cambios de iluminación en el escenario. La componente de intensidad es la que recoge este factor. Por ejemplo hemos configurado este filtro para el color rojo, puede verse el resultado en la figura 4.3.

Los bordes son una característica muy relevante. Nos proporcionan una descripción bastante acertada del entorno. Los bordes se pueden interpretar como puntos de alta derivada espacial en luminancia y contienen mucha información de la imagen. Nos devuelven la posición de los objetos, la forma, el tamaño, etc. Para su implementación en el proyecto hemos utilizado la función `cvCanny` presente en la biblioteca de visión computacional *OpenCV*, biblioteca de la que ya hemos hablado en el capítulo anterior. En la figura 4.3 se puede ver una imagen filtrada por bordes, es la segunda empezando por la izquierda.

[1] http://es.wikipedia.org/wiki/Modelo_de_color_RGB
[2] http://es.wikipedia.org/wiki/Modelo_de_color_HSV

Por último, otra característica ascendente que nos ayuda a acelerar el proceso de representación de una escena es la de segmentos rectos. Las dos características anteriores las hemos heredado del trabajo de Roberto, pero esta última ha sido aporte genuino de este proyecto fin de carrera. Este algoritmo se apoya en el filtro de bordes y devuelve los segmentos rectos de la imagen. Para su implementación hemos utilizado la transformada de *Hough*, también disponible en la biblioteca de *OpenCV* en la función `cvHoughLines2`. En figura anterior puede verse un ejemplo de este tipo de filtrado al lado de la imagen de bordes.

$$sal_{inst}(pixel(t)) = sal_{color}(pixel(t)) + sal_{bordes}(pixel(t)) + sal_{segmentos}(pixel(t)) \quad (4.1)$$

Si las todas las saliencias instantáneas están activas, la saliencia instantánea general de la imagen se calcula combinando todas ellas, es decir, para cada píxel, se suman los valores de saliencia devueltos por cada filtro de ese mismo píxel. En nuestro caso los filtros disponibles son color, bordes y segmentos. Si alguna de estas saliencias no está activa, su valor será cero para que no afecte ni positiva ni negativamente al resultado final. La ecuación matemática queda reflejada en 4.1.

4.2.2. Saliencia acumulada e inhibición de retorno

Con las tres saliencias instantáneas implementadas, ya somos capaces de configurar el sistema según el escenario al que tenga que enfrentarse el robot. Pero esto no es suficiente, ya que si lo dejásemos como está, al no tener memoria, inevitablemente el sistema atentivo nos devolvería siempre el mismo píxel. Además no tenemos un modo de saber qué píxeles hemos visitado ya y cuáles no. Para evitar esta situación hemos añadido un mapa de saliencia acumulada para que actúe como memoria de los píxeles.

La saliencia acumulada se alimenta en cada iteración de la salida de todas las saliencias instantáneas que tengamos activas. Se actualiza acorde a la siguiente fórmula:

$$sal_{acc}(pixel(t)) = \alpha * sal_{acc}(pixel(t-1)) + (1-\alpha) * sal_{inst}(pixel(t)) \quad (4.2)$$

Si nos fijamos en la fórmula de refresco de la saliencia acumulada, veremos que tenemos una variable de ponderación α, que puede tomar valores entre 0 y 1. Este valor nos indica la prioridad que se da a la saliencia ya acumulada. Cuanto mayor sea su valor, menos importancia se le darán a los cambios que se hayan producido en el

escenario. A nosotros nos interesa justamente lo contrario, por lo que le asignaremos un valor bastante bajo.

Pese a haber enriquecido el sistema de atención 2D con el mapa de saliencia general, aún no hemos resuelto el problema de la hambruna porque puede seguir dándose el caso de que siempre salga el mismo píxel ganador. La inhibición de retorno es una técnica que nos resuelve este problema. Consiste en marcar como visitado cada píxel que nos devuelva el sistema. Actualizamos el valor de los píxeles visitados según la siguiente fórmula:

$$sal_{acc}(pixel\_visitado(t)) = -30{,}0 \tag{4.3}$$

A cada píxel visitado se le asigna un valor arbitrario de -30. Como la información que nos aportan dos píxeles que están muy juntos es muy escasa, marcamos como visitados una pequeña ventana de píxeles alrededor de él. Gracias a este método se puede hacer un reparto de la mirada mucho más eficiente.

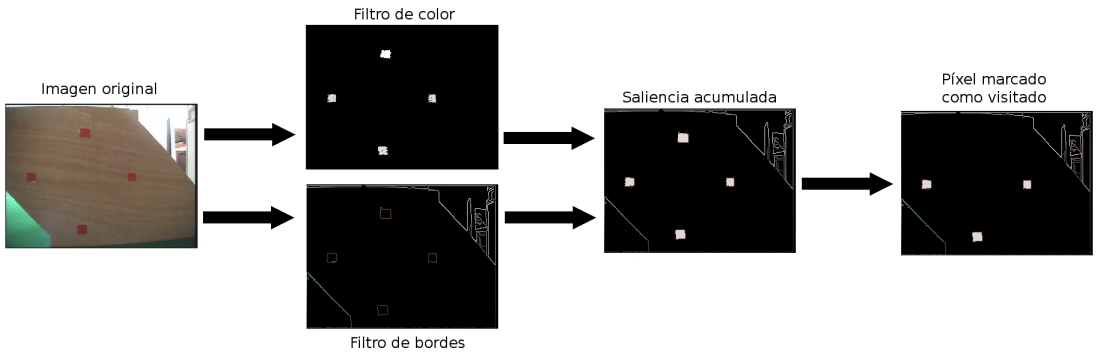


Figura 4.4: Proceso que se sigue para actualizar el mapa de saliencia acumulado.

En la figura 4.4 puede verse el proceso completo desde que se recibe la imagen desde la cámara hasta que actualizamos el mapa de saliencia acumulado. En este caso el píxel ganador ha sido uno que se encuentra en el cuadrado rojo superior. Inmediatamente después de que nos devuelva ese píxel el sistema de atención 2D, lo marcamos como visitado. Con esto conseguimos generar una dinámica que reparte la mirada entre las distintas zonas relevantes de la imagen.

El sistema de atención 2D implementa un algoritmo de atención local. El "mundo" para este sistema es únicamente lo que ven las cámaras. Por eso, cuando las movemos, es necesario reiniciar el mapa de saliencia acumulado poniendo la saliencia de todos los píxeles a cero.

4.2.3. Inhibición cognitiva

Como hemos visto en la primera sección de este capítulo, nuestro sistema atentivo se compone de dos sistemas de atención, 2D y 3D, que conviven y se complementan el uno al otro. Después de varias iteraciones, el sistema de atención 2D no es capaz de recordar si un píxel ya ha sido reconstruido o no. Esto puede dar lugar a que reconstruyamos dos veces el mismo punto.

Para evitar esta situación, hemos introducido la inhibición cognitiva. Esta consiste en usar la memoria 3D de puntos conocidos para impedir reconstruir dos veces el mismo punto. Para ello proyectamos los puntos de la memoria 3D en la imagen de la cámara principal y marcamos los píxeles correspondientes como visitados.

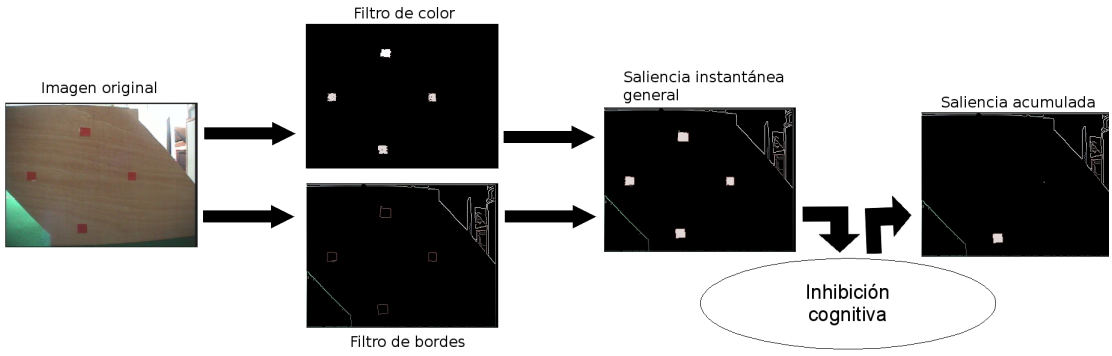


Figura 4.5: Proceso de la inhibición cognitiva.

En la figura 4.5 se puede ver el funcionamiento de la inhibición cognitiva. Suponemos que nos encontramos una iteración ya avanzada y que ya hemos reconstruido los tres cuadrados rojos superiores, pero la saliencia instantánea nos devuelve otra vez todos los píxeles salientes. La inhibición cognitiva, usando la memoria 3D marca los píxeles correspondientes a los puntos ya reconstruidos para que no vuelvan a salir como ganadores. En la última imagen puede verse cómo quedaría el mapa de saliencia

acumulado que guía la mirada precisamente hacie el vértice inferior, el único punto que aún no ha sido reconstruido en 3D.

La inhibición cognitiva le aporta al sistema cierta "inteligencia". Conseguimos que lo ya conocido no le llame la atención. La inhibición cognitiva viene de una capa superior, donde no operamos directamente con la información de la cámara y utilizamos cierto conocimiento para enriquecer la información devuelta por las cámaras.

La combinación de estas dos inhibiciones hace que el sistema alterne entre los distintos puntos candidatos evitando que se repitan siempre los mismos optimizando en buena medida el sistema, es decir, conseguimos que no se pierda el tiempo en analizar lo que ya se conoce. La primera técnica se basa en el tiempo que ha pasado desde que se visitó por última vez dicho punto y, la segunda, evita que vuelvan a salir puntos ya recogidos en nuestra memoria 3D.

4.3. Reconstrucción 3D con movimiento de cámaras

El sistema de atención 2D nos devuelve el píxel con mayor saliencia en ese momento. A partir de aquí se pone en marcha el algoritmo de reconstrucción 3D para calcular la posición de dicho punto en el espacio tridimensional. En esta sección explicaremos con detalle el algoritmo utilizado para la reconstrucción en 3D de los puntos interesantes del escenario.

El mecanismo clásico de reconstrucción 3D consiste en, una vez que tienes seleccionado el píxel más saliente, buscar en la imagen de la otra cámara su píxel homólogo. Roberto [Calvo Palomino, 2008], en su proyecto, usó una serie de restricciones para agilizar este proceso. Una de ellas fue acotar la búsqueda de dicho píxel en la recta epipolar. Esta recta se calcula proyectando en la cámara auxiliar la recta de retro-proyección del píxel más saliente de la cámara principal. Para acotar aún más la búsqueda, sólo comprobó los píxeles de la recta epipolar que tenían suficiente saliencia. Por último, una vez que tienes los dos píxeles homólogos, hay que intersectar las rectas de retro-proyección de los píxeles. Haciendo esta triangulación, hallamos el punto en el espacio tridimensional.

En nuestro proyecto hemos propuesto otra técnica alternativa de reconstrucción 3D aprovechando la libertad de movimiento de nuestras cámaras. Usaremos este movimiento para recorrer en profundidad, con la cámara auxiliar (la derecha), la recta de retro-proyección en 3D del píxel con mayor saliencia de la cámara principal (cámara izquierda), tal como muestra la figura 4.6. Este proceso no sigue los mecanismos clásicos de cálculo de profundidad por triangulación del par estéreo de cámaras quietas, sino que explora las posibilidades del movimiento ocular.

El algoritmo comienza igual que el de Roberto, calculando el píxel con mayor saliencia en la cámara dominante (*p'*). Una vez que lo tenemos, calculamos la recta retro-proyectiva de dicho píxel (r). La diferencia más notable con respecto al otro algoritmo, es que a partir de aquí, trabajaremos únicamente en el espacio tridimensional. Para buscar el píxel homólogo calcularemos una serie de puntos 3D pertenecientes a la recta retro-proyectada, que serán nuestros puntos candidatos (*P1, P2, P3, P4, P5* y *P6*). Uno por uno iremos mirando cada punto y comparando la zona central de la cámara auxiliar (la cámara que se mueve) con la zona del píxel hallado al principio. Para la comparación utilizaremos parches de tamaño variable según el tipo de escenario. Esta opción es configurable en nuestro esquema aunque habitualmente utilizamos un tamaño de 60x60 porque da buenos resultados. La comparación se hace píxel a píxel como en [Calvo Palomino, 2008]. Al final de este proceso se escoge el parche que tenga la menor diferencia. En la gráfica colocada en la esquina superior izquierda de la figura 4.6 se puede ver un ejemplo. El parche ganador es correspondiente al punto más bajo porque es el punto menos diferente. Una vez que tenemos un punto ganador, ya no tenemos que realizar triangulación de ningún tipo, ya que hemos trabajado directamente en el espacio tridimensional.

La diferencia entre ambos algoritmos es el espacio en el que se trabaja. El algoritmo de reconstrucción 3D clásico trabaja en el espacio 2D, buscando la correlación entre píxeles en la recta epipolar y en el último momento realiza los cálculos necesarios para pasar a 3D. En cambio, nuestro algoritmo trabaja desde el primer momento en tres dimensiones. La búsqueda del píxel homólogo la realizamos en el rayo 3D de retro-proyección.

Para ayudarnos en esta tarea de reconstrucción 3D hemos utilizado la biblioteca de *Progeo*, que según vimos en el capítulo 3, nos ofrece las herramientas necesarias para

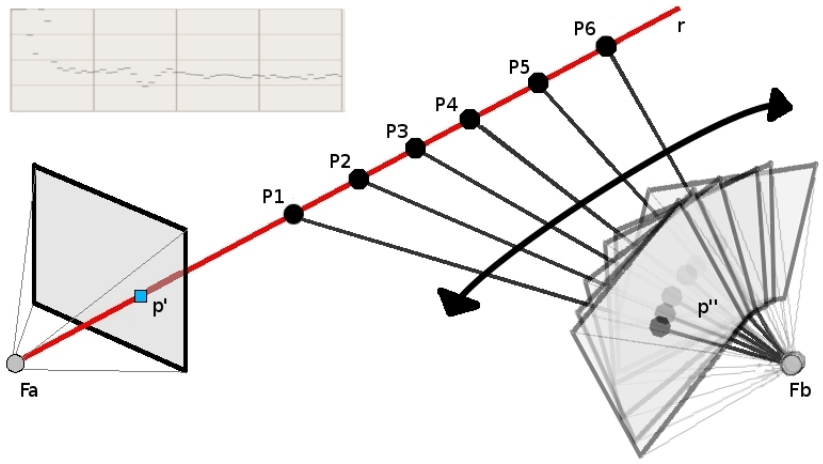


Figura 4.6: Reconstrucción 3D usando vergencia.

trabajar en un mundo en tres dimensiones. A continuación explicaremos el modelo de movimiento y el resto de técnicas y restricciones que hemos implementado para realizar la reconstrucción 3D.

4.3.1. Modelo de movimiento del cuello mecánico

El modelo de movimiento nos ayuda a reflejar los cambios físicos que se producen en las cámaras, nos permite tenerlas caracterizadas aunque cambien de posición. Este modelo utiliza un sistema de referencia solidario con el robot. Su función principal es la de devolver los parámetros extrínsecos de las cámaras cuando el cuello mecánico cambia de posición.

El modelo de movimiento utiliza varios sistemas de referencia. El sistema de referencia principal es solidario con el robot. El origen del eje de coordenadas se encuentra a ras de suelo en un punto intermedio entre las dos cámaras. Para realizar cálculos intermedios utiliza dos sistemas de referencia más. Cada uno de ellos tiene su origen en una de las bases de las cámaras. En la figura 4.7 se ven perfectamente dónde están colocados cada uno de los sistemas de referencia.

Los tres sistemas de referencia tiene todos los ejes paralelos. El cambio de base de un sistema a otro es muy sencillo, ya que sólo implica una traslación. A la derecha de la figura 4.7 se puede ver una representación gráfica de una traslación. En este caso queremos trasladar el eje de coordenadas al punto $P0(X_0, Y_0, Z_0)$ y expresar el punto

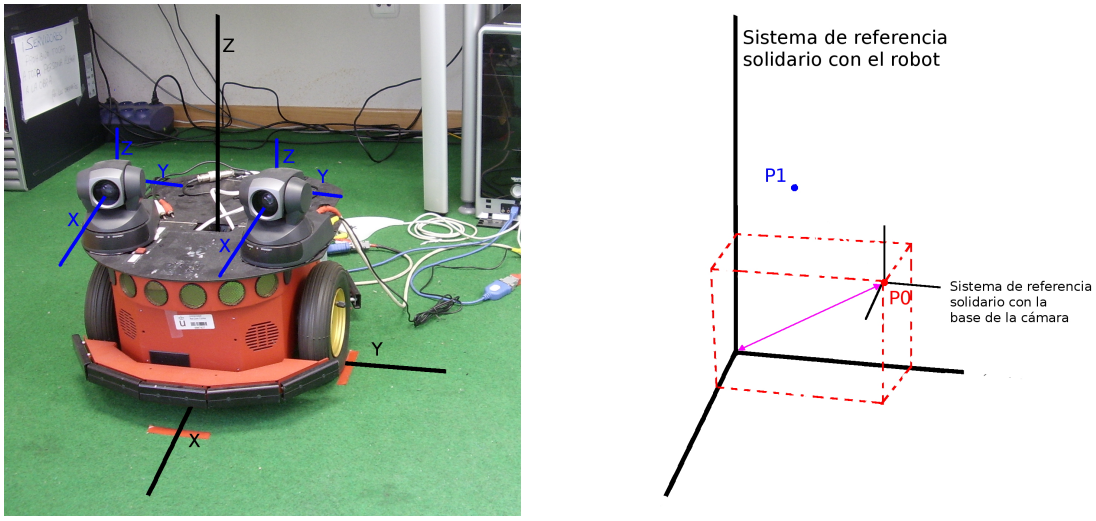


Figura 4.7: A la izquierda, los distintos sistemas de referencia que utiliza el robot. En la derecha, representación gráfica de la traslación de un punto.

P1 respecto de ese sistema de referencia. Este punto con las nuevas coordenadas lo llamaremos *P1'*. Las nuevas coordenadas de *P1'* se hallan de la siguiente manera:

$$P1' = \begin{cases} X_{P1'} = X_{P1} - X_0 \\ Y_{P1'} = Y_{P1} - Y_0 \\ Z_{P1'} = Z_{P1} - Z_0 \end{cases} \tag{4.4}$$

Los sistemas de referencia solidarios con las bases de las cámaras los necesitamos para relacionar los valores devueltos por el cuello mecánico con el espacio tridimensional. Para esta operación nos ayudamos de las fórmulas matemáticas que relacionan las coordenadas esféricas (φ,θ,r) y las coordenadas cartesianas (x',y',z'). La relación de las coordenadas esféricas con las variables de movimiento del cuello es directa: φ se relaciona con *pan* y θ con *tilt*. Las relaciones matemáticas son las siguientes:

$$\begin{cases} x = r & \sin\theta & \cos\varphi \\ y = r & \sin\theta & \sin\varphi \\ z = r & \cos\theta & \end{cases} \tag{4.5}$$

$$\begin{aligned} r &= \sqrt{x^2+y^2+z^2} \\ \theta &= \begin{cases} \arctan(\frac{\sqrt{x^2+y^2}}{z}) & z > 0 \\ \frac{\pi}{2} & z = 0 \\ \pi + \arctan(\frac{\sqrt{x^2+y^2}}{z}) & z = 0 \end{cases} \\ \varphi &= \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \frac{\pi}{2} sgn(y) & x = 0 \\ \pi + \arctan(\frac{y}{x}) & x < 0 \end{cases} \end{aligned} \tag{4.6}$$

Las fórmulas en 4.5 relacionan las coordenadas esféricas con las coordenadas cartesianas y las fórmulas en 4.6 realizan la operación inversa.

En un entorno ideal, el foco de atención de las cámaras sería paralelo al eje X del sistema de referencia principal y coincidiría con el eje X del sistema de referencia propio. Como conseguir esto es prácticamente imposible, tenemos que hacer unos cálculos previos para averiguar el desplazamiento que tiene el foco de atención de la cámara con respecto al eje X del sistema de referencia propio, es decir, hallar unos valores que nos facilitan el cambio de base desde el sistema de referencia de las cámaras al sistema de referencia de las bases de las cámaras. Estos valores van a ser un número de grados horizontales y verticales determinados, ya que la cambio sólamente implica una rotación en *pan* y otra en *tilt*. En este momento trabajamos en coordenadas esféricas porque simplifican mucho los cálculos.

Lo primero para poder usar el modelo es necesario hacer una primera calibración de las cámaras estando el cuello mecánico en su posición inicial, es decir, 0º de *pan* y 0º de *tilt*. De esta primera calibración calcularemos *pan_inicial* y *tilt_inicial* que se definen como el *desplazamiento inicial del cuello mecánico*. Este desplazamiento es la diferencia en grados entre el punto 0º de *pan* y 0º de *tilt* de la cámara (*foa*) con el del sistema de referencia solidario con la base de esa misma cámara y paralelo al del robot. En la figura 4.8 se ve perfectamente este concepto.

La función principal del modelo es devolver los parámetros extrínsecos de la cámara cuando las variables del cuello mecánico, *pan* y *tilt*, son modificadas. Para conseguir estos nuevos parámetros, se suman los *pan* y *tilt* actuales con los *desplazamientos del cuello mecánico*. Estas nuevas coordenadas esféricas, que son solidarias con el sistema de referencia de la base de la cámara, las transformamos en cartesianas con las ecuaciones en 4.5. Este punto que acabamos de calcular es el nuevo foco de atención de la cámara, pero con respecto al sistema de referencia de esa misma cámara. Para hacerlo solidario con el sistema de referencia el robot, hacemos un cambio de base desde el sistema de referencia de la base de la cámara al del robot mediante las fórmulas en 4.4. Este proceso se debe de hacer por separado para cada cámara. El movimiento afecta a los parámetros extrínsecos de las cámaras y con el modelo de movimiento quedan

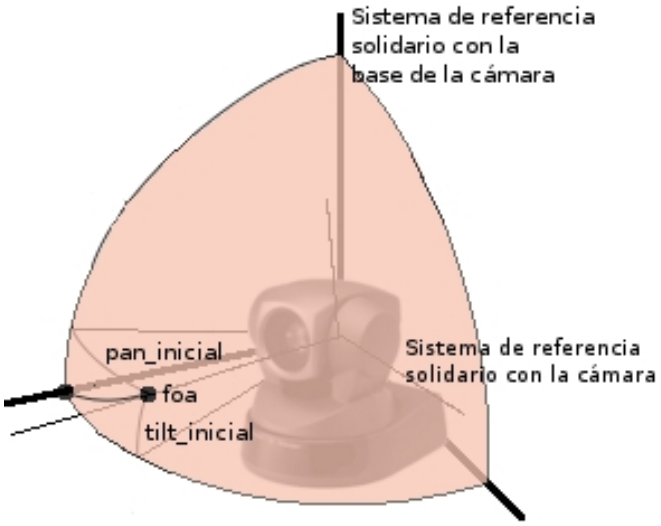


Figura 4.8: Desplazamiento del cuello mecánico en la calibración inicial.

perfectamente caracterizado para cualquier posición *pan* y *tilt*.

El modelo de movimiento también tiene la funcionalidad inversa. Definido un punto 3D, el modelo nos devuelve las órdenes *pan* y *tilt* que hacen que las dos cámaras miren a ese punto. Las cálculos que hay que realizar para esta funcionalidad son parecidos a los anteriores. Lo primero es cambiar de base el punto en cuestión desde el sistema de referencia solidario con el robot al sistema de referencia de la base de la cámara. Mediante las ecuaciones en 4.6 sacamos las coordenadas de longitud y latitud del punto en cuestión. Para calcular las órdenes *pan* y *tilt* del cuello mecánico, restamos a estos valores el *desplazamiento del cuello mecánico*. Por último, introducimos estos valores de *pan* y *tilt* en el cuello mecánico, y las cámaras mirarán al punto inicial.

Hay un detalle importante a destacar cuando queremos que el sistema mire a un punto en concreto. El cuello mecánico de la cámara, por ser mecánico, es menos preciso que los cálculos que realizamos. Esto puede dar lugar a que al decirle que mire a un punto, no esté mirando exactamente a ese punto, sino a otro cercano a él. Para que este error no se refleje en los cálculos, caracterizamos las cámaras en cada iteración con los valores devueltos por los *encoders* del cuello mecánico en vez de con las órdenes enviadas. Por ejemplo, queremos mirar un punto concreto y para ello necesitamos girar el cuello mecánico 5º, pero este, por la pequeña holgura que tiene, solamente puede llegar hasta 4,95º. Para evitar ese pequeño margen de error de 0.05º, calculamos los parámetros extrínsecos de las cámaras con el valor devuelto por los *encoders*, 4,95º.

Una vez que tenemos el modelo de movimiento bien definido, pasaremos a explicar las técnicas utilizadas en el algoritmo de reconstrucción 3D.

4.3.2. Puntos homólogos y movimientos de vergencia

Para emparejar cada píxel con su homólogo, utilizaremos la técnica de correlación [S. Birchfield, 1999]. Esta técnica consiste en coger el entorno cercano al píxel (ventana de correlación) a emparejar y buscar la ventana de la imagen de la otra cámara que mejor encaje.

La vergencia limita la búsqueda del punto homólogo al rayo óptico calculado a partir de la retro-proyección de un píxel en la cámara maestra. El rayo óptico lo discretizamos en una serie de puntos que serán los puntos candidatos. Cuanto mayor densidad de puntos tengamos, mayor precisión tendrá nuestra reconstrucción 3D, pero también aumentará el tiempo de cómputo.

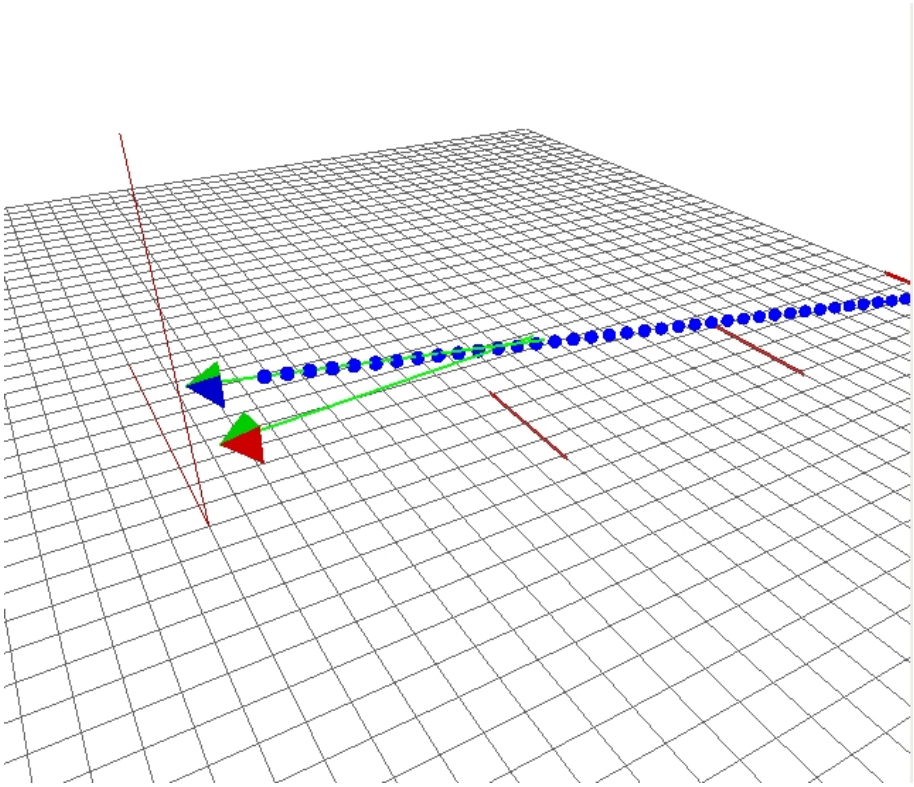


Figura 4.9: Movimientos de vergencia representados con OpenGL.

En la figura 4.6 puede verse un diagrama explicativo. Dado un píxel *p'* en la cámara maestra, proyectamos una línea que pasa por el foco de la cámara *Fa* y por dicho píxel. Esta recta de proyección la discretizamos en una serie de puntos. Por último, movemos la cámara secundaria haciendo que el punto en tres dimensiones quede en la recta de proyección de su centro óptico y compararemos mediante el uso de ventana de correlación. Este proceso lo repetiremos tantas veces como puntos tengamos calculados, de tal manera que siempre coincidirán el centro óptico de la cámara con el nuevo punto 3D que se está explorando. De este modo, el punto 3D de la ventana de la cámara esclava cuyo parche se asemeje más al parche de correlación de la cámara maestra, será el punto correcto.

Aunque la calibración de las cámaras es bastante precisa, nunca llega a ser perfecta. Esto implica que las retro-proyecciones de los puntos calculados en la recta de proyección no sean exactamente los que deberían de ser. Para evitar esta imperfección usamos la alternativa de *vergencia con holgura* (figura 4.10). El algoritmo es igual que con vergencia pura excepto que al final, una vez que tenemos el píxel candidato, hacemos un último ajuste comprobando los píxeles cercanos a este, con su parche correspondiente, con el parche de la imagen principal. De todas estas comparaciones nos quedamos con el más semejante. Con este método, no sólo se obtienen resultados más ajustados, sino que podemos reducir el número de puntos calculados en la recta de proyección aumentado así la vivacidad del proceso.

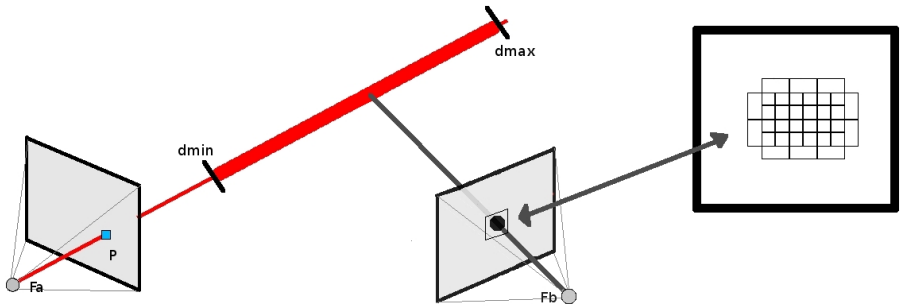


Figura 4.10: Técnica de vergencia aplicando cilindro de holgura.

Otra restricción muy simple y heurística es la restricción por distancia. Conociendo *a priori* el entorno sobre el que vamos a trabajar, restringimos los puntos calculados en la recta de proyección a que se encuentren dentro de un rango de distancias determinado.

En la figura 4.10 se pueden ver esos valores en *dmin* y *dmax*. Este factor es totalmente heurístico y dependiendo del escenario puede cambiar radicalmente. Por ejemplo, en una habitación pequeña, unos valores lógicos para el rango serían un mínimo de cincuenta centímetros y un máximo de cinco metros.

4.4. Atención 3D

En las secciones anteriores hablamos de cómo se obtienen puntos nuevos en 3D. A medida que se consiguen, se incorporan a una memoria espacial. Ahora se desea que el robot haga un seguimiento de esos puntos conocidos y que vaya un poco más allá, realizando hipótesis de posibles puntos que puedan existir aunque que en ese momento no caigan en el campo visual actual. También se ha implementado la capacidad para olvidar los puntos que desaparecen de la escena. Para poder explorar todo el escenario nos ayudamos del cuello mecánico de las cámaras y los movimientos sacádicos que permiten.

Este sistema junto con el modelo de movimiento dotan al robot de una capacidad de atención *overt*. El robot es capaz de ver más allá del campo visual instantáneo, abarcando toda la escena. También hemos implementado la capacidad de incorporar primitivas a la memoria para poder trabajar con segmentos y cuadrados, y así aumentar la información que tenemos de la escena.

4.4.1. Memoria 3D

El algoritmo de atención 3D se encarga de refrescar, olvidar, decidir cuál es el punto más interesante o comprobar que una hipótesis es correcta. Para ello se ayuda de una saliencia asignada a cada punto. Esta saliencia, como pasa en el sistema de atención 2D, indica las ganas que se tiene de mirar al punto correspondiente. Toda esta información es almacenada en la memoria de puntos 3D para que no se pierda en el tiempo. Básicamente es un conjunto de puntos a los que les asociamos saliencia y vida.

Este algoritmo trabaja en el espacio tridimensional. Para mirar a los puntos interesantes, no tenemos que hacer ningún tipo de cálculo, ya que el modelo de movimiento nos va a ayudar a traducir las coordenadas del punto 3D en valores

entendibles por el cuello mecánico.

Cuando el algoritmo de atención 2D no detecta ningún píxel saliente, bien porque no hay o porque ya son conocidos y se han interiorizado, el sistema de atención 3D nos devuelve los puntos ya reconstruidos más interesantes. La dinámica de saliencia será la encargada de este cometido.

Para prestar atención a toda la escena las cámaras saltan de un punto a otro para comprobar que siguen ahí. Mientras las cámaras están en movimiento, no analizamos las imágenes, es como si el sistema se quedase ciego por unos momentos. Este comportamiento es una imitación de cómo se comportan los ojos humanos cuando los movemos. Estos movimientos son conocidos como *movimientos sacádicos*. Ya hemos hablado de estos movimientos en el primer capítulo. Son movimientos cortos, rápidos y precisos. Cuando el ojo está realizando este movimiento no recibe ningún estímulo, volviendo a recibirlos cuando se posa la mirada en algún otro punto.

El sistema de atención 3D hace uso de dos dinámicas concurrentemente, la dinámica de saliencia y la dinámica de vida [León Cadahía, 2006]. La dinámica de saliencia permite repartir la mirada entre los distintos puntos de atención y saber cuál será nuestro siguiente objetivo.

Dinámica de saliencia 3D

Al igual que en el sistema de atención 2D, cada punto de la memoria tiene asignada una saliencia que indica qué punto es el más interesante. La diferencia es que aquí hacemos un seguimiento de objetos 3D concretos en vez de píxeles. El mapa de saliencia es mucho menos denso.

Una forma de decidir la saliencia que posee cada punto de atención, es en función del tiempo que hace que no se visita. Un punto que no se ha visitado causará mayor atracción que otro que haya visitado recientemente, al igual que pasaba con los píxeles en el sistema de atención 2D. Hemos implementado una inhibición de retorno, pero aplicada a objetos en vez de a píxeles.

$$sal(t+1) = sal(t) + \Delta Sal \tag{4.7}$$

La implementación de esta dinámica consiste en que cuando se visita un punto, su saliencia disminuya drásticamente, mientras que la del resto de puntos, aumente. Gracias a esta implementación se consigue un reparto de la mirada entre todos los puntos interesantes. En la figura 4.11 puede verse el comportamiento de la saliencia para uno y dos puntos.

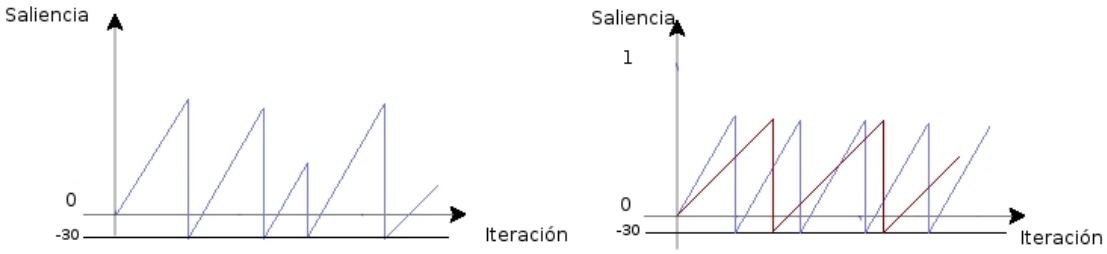


Figura 4.11: Dinámica de saliencia para un punto (izquierda) y para dos (derecha).

Dinámica de vida

La dinámica de vida, como su propio nombre indica, se encarga de gestionar la vida de los puntos de la memoria. Con esta dinámica lo que se pretende es saber cuándo un punto ha salido de la escena o si aún sigue en ella. Si la vida supera cierto umbral, significa que sigue en la escena, pero si está por debajo es que ha desaparecido.

La dinámica de vida es la encargada de incrementar y decrementar la vida de los puntos. La vida funciona a la inversa que la saliencia. Un punto que acaba de ser visitado tendrá la vida más alta que otro que aún no ha sido visitado. Cuando la vida de un punto es inferior a un umbral, el punto es olvidado, y por tanto, borrado de la memoria. La vida disminuye de la siguiente manera:

$$vida(t+1) = vida(t) - \Delta vida \tag{4.8}$$

Si en cambio, se encuentra el punto en la imagen la vida queda así:

$$vida(t+1) = vida\_maxima \tag{4.9}$$

La implementación de esta dinámica consiste en que, cada vez que se visita un objeto, se incrementa su vida, mientras que la de los objetos no visitados, disminuye. Esta implementación permite asignar prioridades distintas a los objetos para que tengan

más vida o esta disminuya mas despacio. En nuestro proyecto hemos asignado la misma prioridad a todos los puntos. En la figura 4.12 puede verse un ejemplo como se comporta esta dinámica de vida.

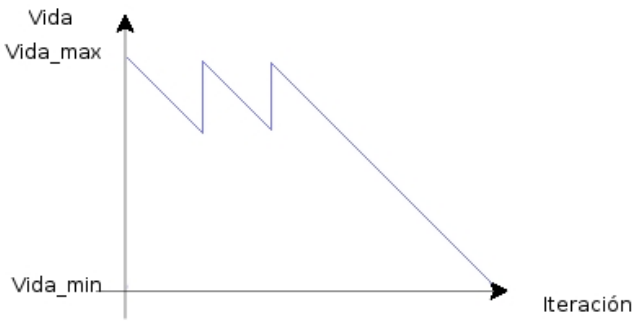


Figura 4.12: Dinámica de vida de un objeto que entra en la memoria y luego es olvidado.

4.4.2. Primitivas abstractas, segmentos

A la hora de introducir puntos nuevos en la memoria 3D, podemos seleccionar entre tres tipos de primitivas distintas como son el punto (que es lo que hemos comentado hasta ahora), el segmento y el cuadrado. Para crear un punto sólamente necesitamos un punto, pero para un segmento y para un cuadrado necesitamos dos y cuatro puntos respectivamente. Estas primitivas enriquece la información que tenemos sobre el escenario.

Para detectar los segmentos en la imagen utilizamos la transformada de *Hough*. A partir del filtro de bordes, esta transformada detecta los segmentos rectos y nos devuelve los puntos extremos. Para ellos nos ayudamos de la biblioteca *OpenCV*.

El proceso de memorización de un segmento es el siguiente. Para realizar este proceso sólamente necesitamos dos puntos, los extremos del segmento. Una vez reconstruidos ambos extremos, antes de insertarlo definitivamente en la memoria, calculamos dos puntos intermedios del segmento y los proyectamos en la imagen. Si los píxeles proyectados tienen saliencia añadimos el segmento a la memoria pues corroborarán la existencia de este segmento. Esto relaciona los puntos que tenemos en la memoria espacial y ahorra tiempo a la hora de representar un escenario.

4.4.3. Atención cognitiva

La atención cognitiva es una parte importante de este proyecto. Al insertar esta funcionalidad en el sistema, le dotamos ya no sólo de la capacidad de representar un escenario, sino de comprenderlo y además realizar hipótesis perceptivas sobre él, que pueden acelerar la percepción del entorno. A partir de este momento, ya no sólo introducimos puntos y hacemos un seguimiento de ellos, sino que somos capaces de "pensar" en tres dimensiones y utilizar esa información para hacer hipótesis perceptivas.

Para nuestro proyecto hemos implementado un módulo capaz de, a partir de tres puntos en 3D, calcular la posición del cuarto punto, si todos ellos forman un cuadrado. De esta manera, sabemos dónde se encuentra ese cuarto punto y podemos mirarlo sin tener que haber hecho los cálculos previos de vergencia. Los tres puntos con los que realizamos la hipótesis no son unos puntos cualesquiera sino que tienen que cumplir un requisito. Deben estar a la misma distancia dos de los puntos con el tercero. Por supuesto, una vez hecha la hipótesis es necesario corroborarla para asegurarnos de que es cierta. Esta forma de actuar simula un aspecto de la percepción humana. A la hora de analizar una imagen, el cerebro es capaz de realizar hipótesis de cosas u objetos que no están en la imagen. Por ejemplo, un caso típico es el caso del cruce y el semáforo. Cuando vamos en coche y llegamos a un cruce, instintivamente miramos a la acera derecha para buscar el semáforo y ver qué disco tiene encendido. Este acto cognitivo acelera la percepción del entorno de un escenario. Nosotros en realidad no sabemos si el semáforo está en ese sitio o no, pero primero miramos ahí. En caso de no encontrarlo entonces pasaremos a buscarlo.

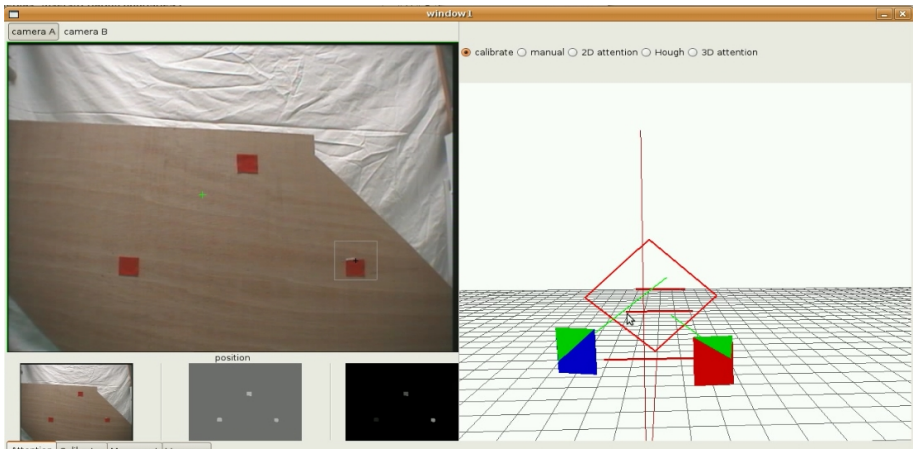


Figura 4.13: Reconstrucción de un cuadrado cognitivamente.

En la figura 4.13 se puede ver este proceso en acción. En la imagen de las cámaras (izquierda) podemos ver tres puntos que podrían ser un cuadrado. El sistema, después de reconstruir en 3D los tres puntos, ha calculado ya el cuarto punto, que ni siquiera se encuentra en la imagen, y lo ha situado en el espacio. Se puede ver en la reconstrucción 3D de la derecha. Ahora sólo falta comprobar la hipótesis mirando con las cámaras al nuevo punto.

Esta característica tiene una ventaja notable frente a reconstruir el cuadrado de sus cuatro puntos. La reconstrucción de los tres primeros puntos se hace igual con atención cognitiva que sin ella. Es a la hora de reconstruir el cuarto punto donde aparecen las bondades de esta característica. Para reconstruir el cuarto punto no se necesita hacer un recorrido de vergencia como hemos hecho con los otros puntos, acción muy lenta, sino que se hace una única comprobación directamente en tres dimensiones. El ahorro en tiempo es muy grande. Es importante comentar que el punto hipótesis se comprueba inmediatamente después de que se calcule.

Esta atención cognitiva se representa como una capa de abstracción de más alto nivel. Utilizamos la información que nos devuelve la cámara para ir más allá y adelantarnos a posibles acontecimientos. Hemos realizado un experimento con esta funcionalidad que explicaremos con más detalle en el siguiente capítulo que está dedicado a los experimentos.

Capítulo 5
Experimentos

Una vez descrito el diseño y la implementación del sistema perceptivo usando reconstrucción 3D con movimiento de cámaras, vamos a poner a prueba el esquema y comprobaremos si cumple o no las expectativas requeridas. Las pruebas descritas en este capítulo han sido seleccionadas para probar los aspectos más críticos.

Los experimentos y las pruebas nos sirven para validar experimentalmente la solución programada y sus diferentes bloques. También para analizar con cierto detalle el funcionamiento del sistema o de alguna de sus partes. Además hemos incluido experimentos probando diferentes alternativas para ayudar a ajustar lo mejor posible el algoritmo final, como por ejemplo la vergencia con holgura.

5.1. Modelo de movimiento

Los primeros experimentos realizados los hemos centrado en probar y validar el modelo de movimiento creado. El modelo de movimiento es una pieza clave del sistema, ya que es él que nos permite mover las cámaras libremente ajustando los parámetros extrínsecos y con ello relacionar la información en las imágenes con el espacio tridimensional en todo momento.

Para este experimento hemos preparado un escenario donde conocemos la posición exacta de varios puntos respecto al sistema de referencia del robot. A través de la interfaz gráfica del esquema, le ordenamos a las cámaras que miren a esos puntos. Con este experimento queremos comprobar que el modelo de movimiento es coherente y que las cámaras centran en sus imágenes los puntos ordenados. En la figura 5.1 se pueden ver el escenario y los puntos conocidos que son las pelotas rosas y los vértices de la caja de cartón. Las coordenadas de los mismos aparecen en la imagen.

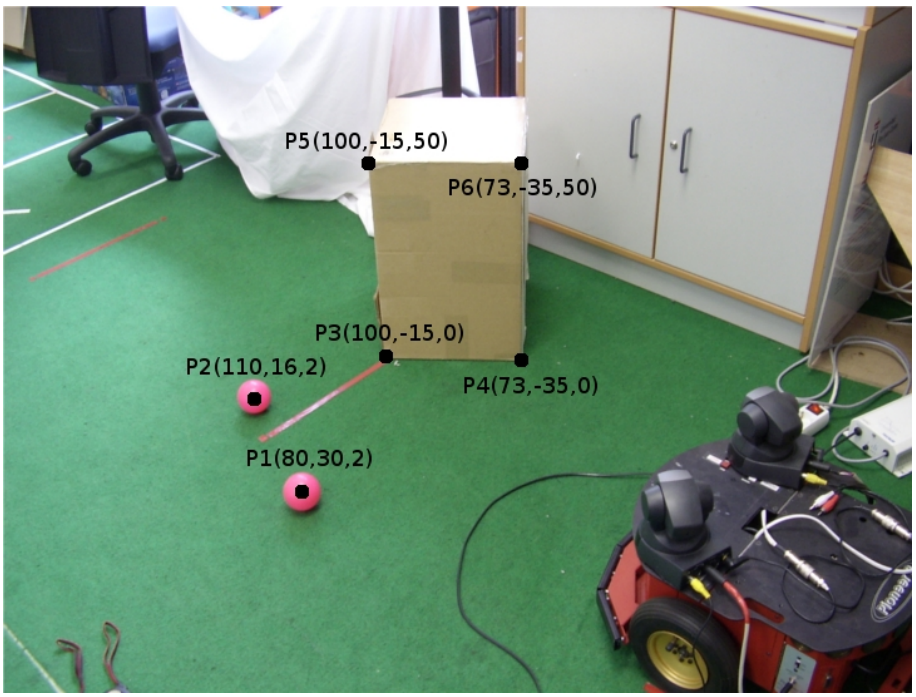


Figura 5.1: Escenario del experimento del modelo de movimiento.

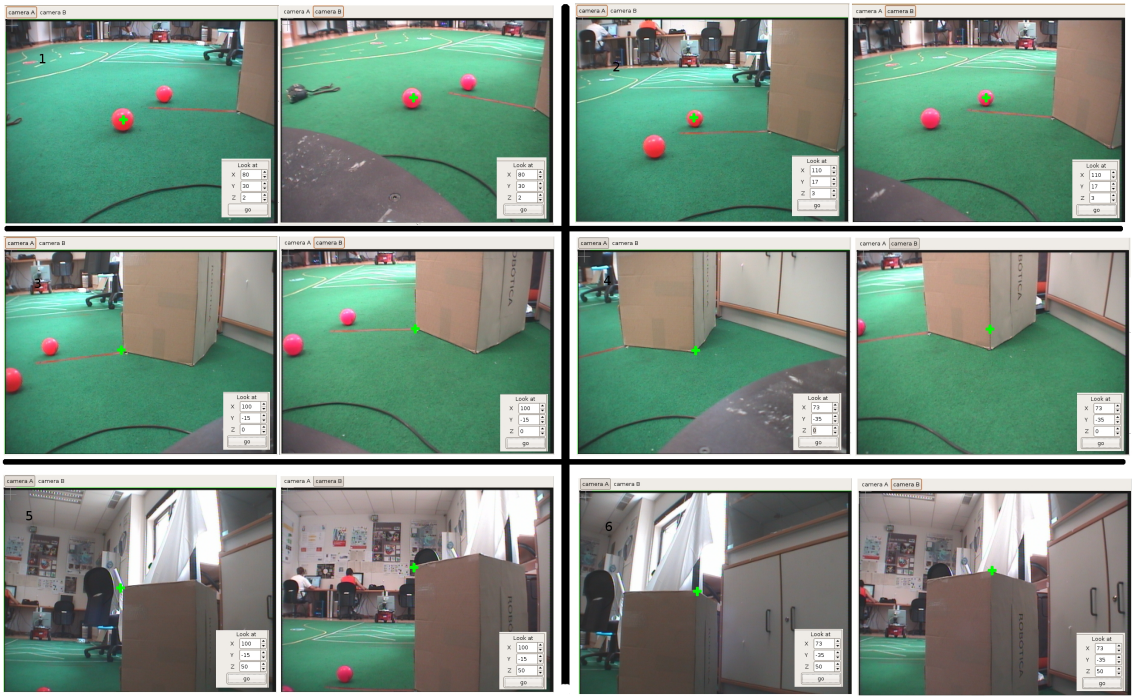


Figura 5.2: Experimento del modelo de movimiento.

En las imágenes de las pruebas el foco de atención esta pintado en verde y en forma de cruz tanto en la cámara izquierda como en la derecha. Si el foco coincide con el punto al que queremos mirar significa que la precisión del sistema es buena. Se puede apreciar en la figura 5.2 que los resultados son bastante precisos. La precisión con que el sistema mira los puntos es bastante buena. Solamente hay un punto en el que la cámara de la derecha no ha podido mirar correctamente. Esto es debido a que se encuentra fuera del rango de visión de la cámara. El cuello mecánico de estas cámaras tiene un rango de amplitud vertical de -25º hasta 25º. Para poder mirar este punto, el cuello tendría que sobrepasar ese límite, cosa que no es posible. Con el movimiento de las cámaras se abre más la escena que el campo visual de la cámara estática, pero no es completo, no abarca los 360º posibles.

Para medir la precisión del modelo de movimiento cuando reconstruimos puntos en 3D, hemos hecho otro experimento utilizando el mismo escenario. Esta otra prueba consiste en reconstruir los puntos interesantes en 3D usando el algoritmo de reconstrucción 3D con movimiento de cámaras. Hemos comparado las coordenadas devueltas por el algoritmo con las coordenadas reales y calculado el error medio.

Para esta prueba hemos utilizado la funcionalidad de reconstrucción 3D manual, que consiste en picar en el punto interesante de la imagen de la cámara principal. El sistema se encarga de buscar el píxel homólogo, mediante la técnica de reconstrucción 3D con movimiento de cámaras, y devolver las coordenadas 3D calculadas del punto. Es importante mencionar que hemos utilizado la técnica de vergencia pura, que ya hemos explicado el apartado 4.3.2.

Punto	Coordenadas originales	Coordenadas calculadas	Error(cm)
P1	(80,30,2)	(76'9,28'1,3'8)	4,06
P2	(110,16,2)	(105,17'3,5)	5,38
P3	(100,-15,0)	(98'4,-17'2,1'2)	2,97
P4	(73,-35,0)	-	-
P5	(100,-15,50)	(97'5,-16'8,49'2)	3,18
P6	(73,-35,50)	(70'9,-35'4,49'6)	2,17
		Error medio	3,55

Cuadro 5.1: Tiempos generales de representación 3D.

En la tabla 5.1 se pueden ver los resultados obtenidos. que son satisfactorios. Con un error medio de 3,55 centímetros, la reconstrucción de escenarios es bastante fiel a la realidad. Con estos experimentos damos por validado el modelo de movimiento con

un resultado satisfactorio para objetos cercanos a las cámaras.

También hemos realizado una prueba reconstrucción de un punto lejano. Este lo hemos situado a una distancia de tres metros y medio. Los resultados concretos obtenidos han sido:

Punto	Coordenadas originales	Coordenadas calculadas	Error(cm)
P	(350,0,27)	(290,5,-8,1,30,4)	60,02

Se puede apreciar que el error es demasiado grande. Por lo que pondremos una limitación de dos metros para asegurar la correcta reconstrucción de los puntos. La calibración y el modelo geométrico no son los suficientemente precisos como para abordar la representación 3D de objetos distantes a las cámaras por encima de ese umbral.

5.2. Novedades y olvido en la memoria 3D

En esta sección vamos realizar una prueba de ejecución típica del esquema. En este experimento vamos ver cómo se comporta el sistema cuando hace un seguimiento de varios puntos, cómo se actualiza la memoria cuando entran puntos nuevos y cómo se olvidan (se borran de la memoria) si después de un periodo de tiempo no hemos vuelto a encontrar dicho punto en las imágenes. Hemos configurado el sistema de atención 2D para que nos llamen la atención los objetos de color rojo.

Tal y como se ve en la figura 5.3 el algoritmo comienza a reconstruir e insertar puntos en la memoria 3D. Estos tres primeros puntos tienen la característica de que son tres de las cuatro esquinas de un cuadrado. Cuando reconstruye las tres, el sistema se da cuenta de que cumplen los requisitos de un posible cuadrado. Mediante cálculos cognitivos reconstruye el cuarto punto y sólo tiene que ir a comprobar la hipótesis calculada. Esta característica de cálculo de hipótesis la detallamos más adelante en otra sección de este capítulo.

La parte que más nos interesa para este experimento comienza una vez que ya hemos ingresado los cuatro puntos del cuadrado en la memoria 3D. Fijémonos en la figura 5.4. En la primera imagen se puede ver que ya hay reconstruido un cuadrado en

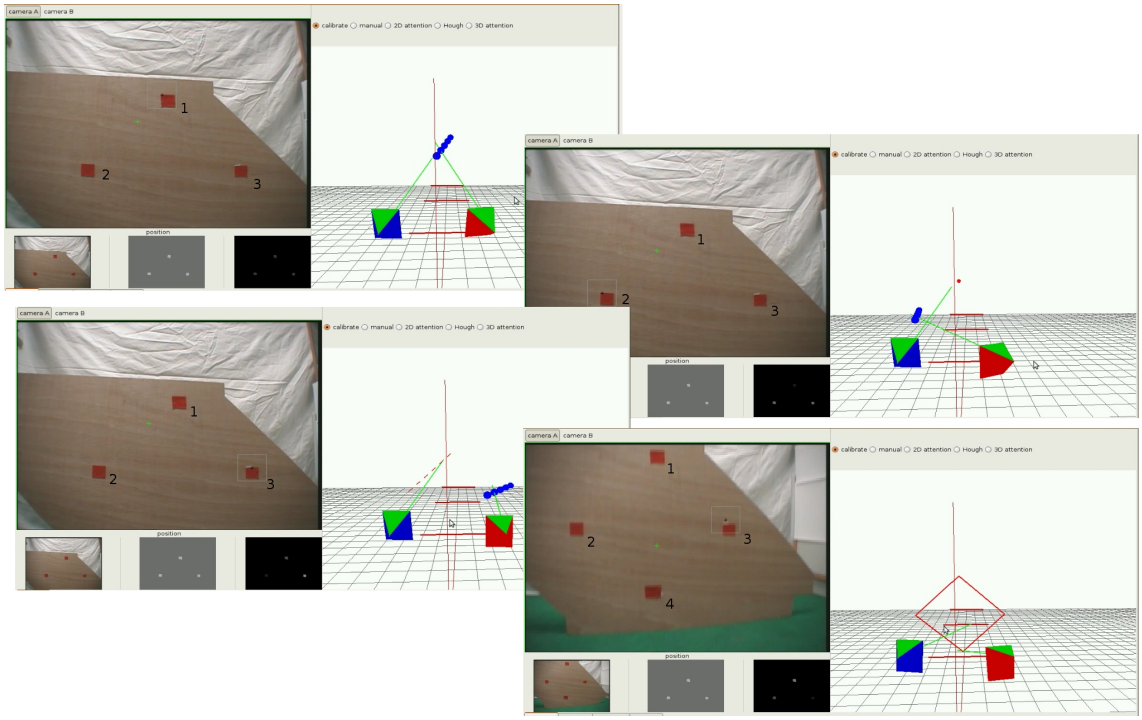


Figura 5.3: Comienzo de la ejecución típica de seguimiento de puntos.

tres dimensiones. En la segunda imagen se puede ver que hemos introducido un quinto punto en el escenario real. El sistema se ha dado cuenta y lo está reconstruyendo. Acto seguido introducimos un sexto punto. De igual manera, el sistema lo reconoce como nuevo y lo sitúa en 3D. Después de varias iteraciones, donde el sistema ha estado refrescando los puntos ya conocidos, hemos quitado el quinto punto que habíamos introducido anteriormente. En la cuarta imagen se puede ver que, aunque ya no se encuentra el punto donde estaba antes, el sistema visita ese lugar para comprobar si aparece de nuevo ese punto de interés. En la última imagen se puede ver como, después de varias iteraciones, el sistema termina por olvidar ese punto de interés y lo saca de la memoria 3D.

De este experimento podemos sacar varias conclusiones. La primera es que el sistema atentivo 3D hace un reparto de la mirada eficiente. Va cambiando contínuamente de punto de interés gracias a la dinámica de saliencia con que gestionamos la memoria de puntos 3D. La segunda conclusión que sacamos es que la gestión de puntos de interés funciona correctamente. Los puntos, una vez visitados, si siguen en su lugar, son refrescados en la memoria. En cambio, cuando no encontramos un objeto donde estaba antes, seguimos visitando ese lugar durante un tiempo por si vuelve a aparecer. Pasado ese tiempo lo olvidamos puesto que ya ha desaparecido del escenario.

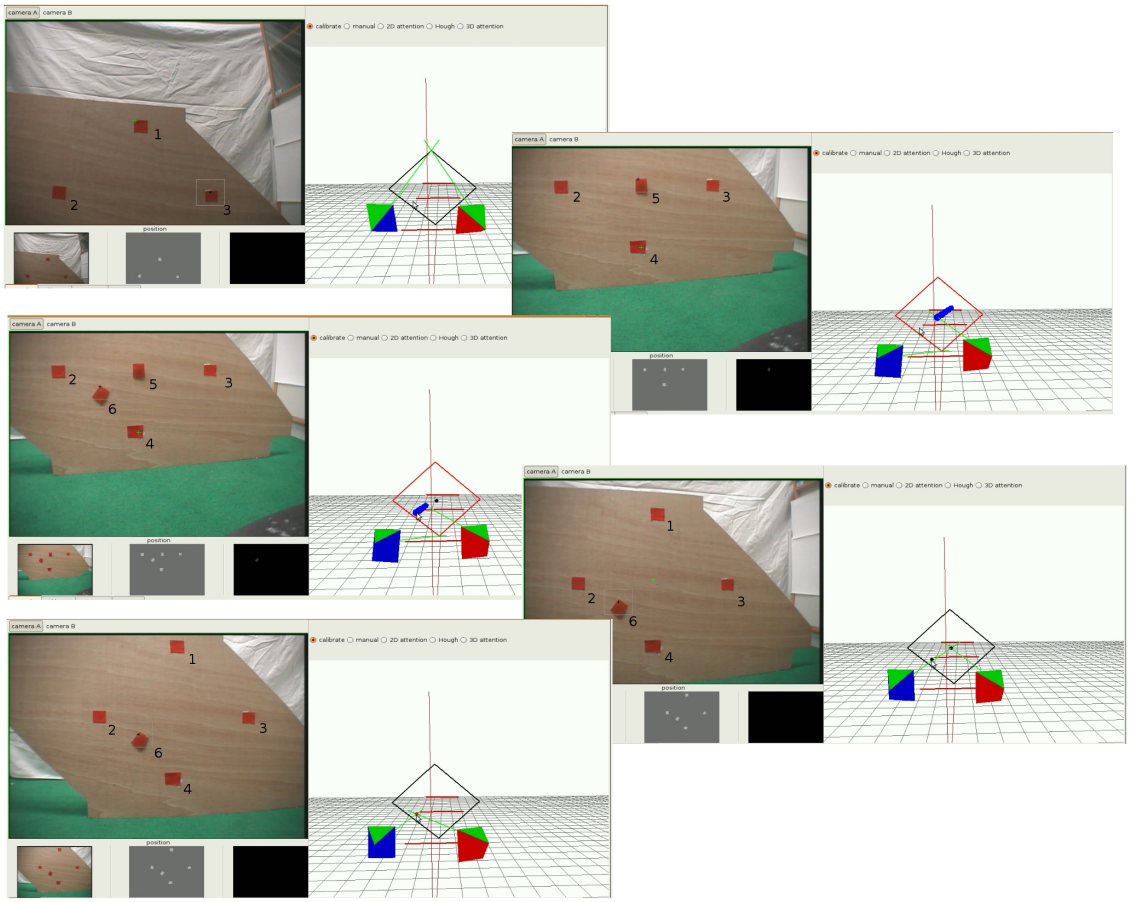


Figura 5.4: Atención 3D para el seguimiento de puntos interesantes del escenario.

5.3. Reconstrucción de segmentos rectos

Otra ejecución significativa de nuestra aplicación es la reconstrucción de un escenario usando segmentos. Este método acelera bastante la reconstrucción del escenario, ya que con solamente dos puntos, tenemos un segmento. El tiempo de cómputo no es el mismo para reconstruir un objeto punto a punto que utilizando segmentos. El sistema atentivo 2D es el que nos devuelve los extremos del segmento. Como hemos explicado en capítulos anteriores, nos ayudamos de la transformada de *Hough* para esta tarea.

Para este experimento hemos dispuesto un escenario compuesto en su mayoría por segmentos rectos. Para reconstruir un segmento, primero reconstruimos sus extremos en 3D, y luego para cerciorarnos de que es realmente un segmento, comprobamos dos

puntos más pertenecientes al segmento. En la figura 5.5 pueden verse cuatro cruces de color azul clarito en dos de las imágenes. Las cruces de los laterales son los extremos del segmento y las del centro son los puntos que hemos calculado para verificar que el segmento está bien reconstruido.

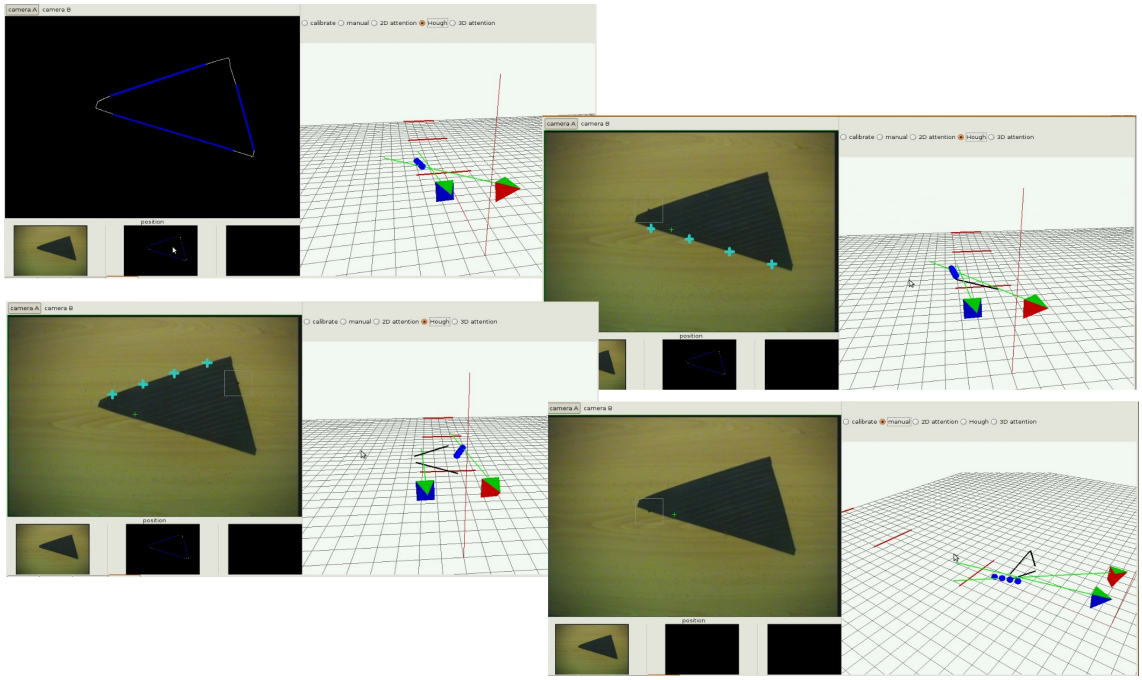


Figura 5.5: Reconstrucción 3D de un escenario con segmentos.

Los resultados que hemos obtenido han sido los esperados. Los segmentos detectados en el escenario aparecen remarcados en azul en la primera imagen de la figura 5.5. Con esta técnica hemos acelerado notablemente la reconstrucción del escenario, ya que queda representado con tres segmentos (seis puntos) en vez de los 20 o 30 puntos que necesitaríamos para su reconstrucción punto a punto.

5.4. Atención cognitiva

En esta sección vamos a hablar del experimento realizado utilizando un razonamiento cognitivo para incluir puntos en nuestra memoria 3D. En este experimento vamos a mostrar la potencia de la arquitectura. Al igual que la reconstrucción del escenario con segmentos, esta otra característica acelera la percepción 3D del entorno.

El objetivo es mostrar al sistema una figura incompleta y que sea capaz de reconstruir las partes que faltan. Como ejemplo concreto vamos a mostrarle tres puntos que forman parte de un cuadrado. El sistema, una vez que reconstruya en 3D e introduzca en su memoria las tres esquinas, calculará la posición de la cuarta esquina y comprobará en las imágenes que efectivamente se encuentra en el punto calculado.

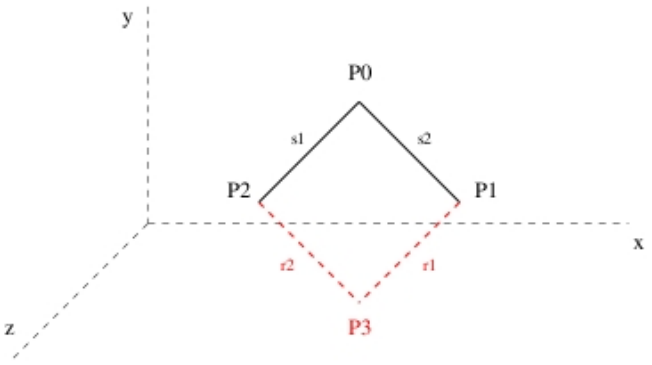


Figura 5.6: Representación del problema del cuadrado.

$$r_1 = \begin{cases} x_3 = x_1 + (x_0 - x_2) * t \\ y_3 = y_1 + (y_0 - y_2) * t \\ z_3 = z_1 + (z_0 - z_2) * t \end{cases} \tag{5.1}$$

$$r_2 = \begin{cases} x_3 = x_2 + (x_0 - x_1) * t \\ y_3 = y_2 + (y_0 - y_1) * t \\ z_3 = z_2 + (z_0 - z_1) * t \end{cases} \tag{5.2}$$

$$t = \frac{x_1 - x_2}{(x_0 - x_1) - (x_0 - x_2)} \tag{5.3}$$

Para obtener la cuarta esquina del cuadrado es necesario hacer cálculos geométricos. Para esto vamos a analizar la figura 5.6. Dicha solución se basa en obtener las rectas *r1* y *r2*, para calcular su intersección y así hallar el punto buscado. La recta *r1* debe ser paralela a la recta *s1* y pasar por el punto *P1*, justo lo que refleja la ecuación 5.1. Del mismo modo, la recta *r2* tiene que ser paralela a la recta *s2* y pasar por el punto *P2*, que matemáticamente queda representado en la ecuación 5.2. Por último, en la ecuación 5.3 se resuelve la incógnita t, que la sustituiremos en cualquiera de las dos ecuaciones de las rectas del punto P3.

En la figura 5.7 se puede ver el desarrollo de cómo se calcula la hipótesis del cuarto punto de un cuadrado. En las tres primeras imágenes se ve cómo se reconstruyen las tres esquinas del cuadrado visibles. Con estos tres puntos realizamos los cálculos necesarios para hipotetizar dónde se encuentra la cuarta esquina, e inmediatamente después procedemos a visitarla para verificar que nuestra hipótesis perceptiva (de existencia de un cuadrado) es correcta. En este ejemplo se ve muy bien la característica de atención abierta (*overt*) del sistema. Este es capaz de prestar atención a un punto que no se encuentra actualmente en el campo visual de ninguna cámara.

De este experimento podemos sacar varias conclusiones. La primera es que hemos creado un sistema que es capaz de construir una hipótesis de situaciones que no se ven en la imagen, ofreciendo así un matiz inteligente en el comportamiento del algoritmo. Otra conclusión interesante es que hemos dotado al sistema la capacidad de decisión utilizando los datos en 3D, y no sobre las imágenes que obtiene la cámara.

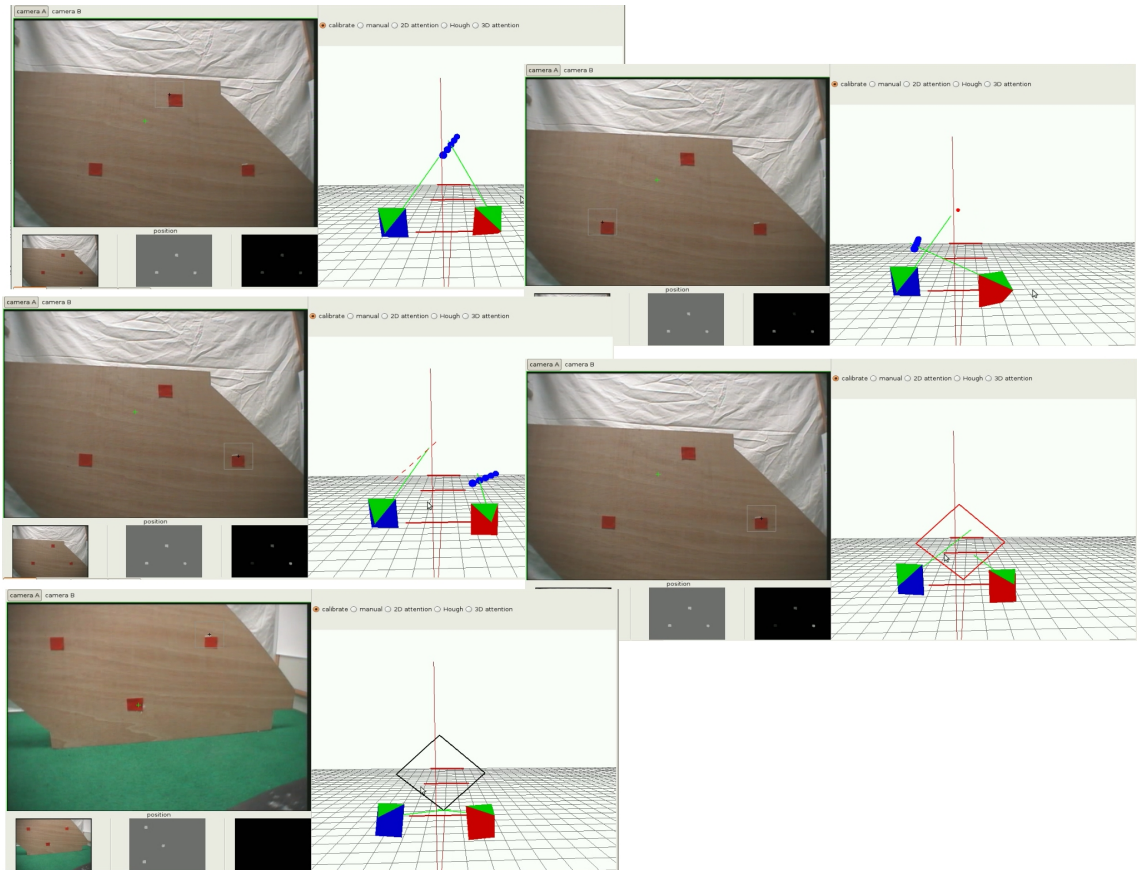


Figura 5.7: Secuencia de hipótesis de la cuarta esquina de un cuadrado.

5.5. Inhibición cognitiva

En este experimento vamos a analizar una misma escena activando la característica de inhibición cognitiva y desactivándola para apreciar mejor las ventajas que ofrece. Esta característica nos ayuda a sincronizar la memoria 3D (lo que ya conocemos), con las imágenes que analiza el sistema de atención 2D (lo que estamos viendo en ese momento) para evitar reconstruir puntos que ya conocemos, tal y como se describió en la sección 4.2.3.

En la figura 5.8 podemos ver el comportamiento que sigue el esquema con la inhibición cognitiva desactivada. En la imagen de la cámara principal detecta tres puntos interesantes y los reconstruye en 3D uno a uno. Las tres primeras imágenes muestran este procedimiento. Una vez que hemos reconstruido estos tres puntos le pedimos al sistema de atención 2D que nos devuelva un nuevo píxel interesante. Como el sistema no tiene alimentación de la memoria de puntos 3D, no sabe si ya ha reconstruido un punto o no, por lo que nos devolverá el píxel con mayor saliencia, que es el punto *1* y pasará a reconstruirlo de nuevo. Justo en ese momento introducimos un cuarto punto en el escenario (imagen 4). Cuando termina de volver a construir el punto *1* pasa a reconstruir el punto con más saliencia, esta vez es el *2* (imagen 5). De igual manera reconstruirá el punto *3* (imagen 6). Y, por fin, en la séptima imagen se ve cómo reconstruye el punto que habíamos introducido nuevo en la escena. En la última imagen vuelve a comenzar el ciclo de reconstrucción 3D con el píxel más saliente, el correspondiente al punto *1*.

Para optimizar la reconstrucción y refresco de puntos 3D interesantes, hemos implementado la inhibición cognitiva. Funciona marcando como visitados los puntos que ya tenemos en la memoria 3D para evitar que el sistema de atención 2D los tome como nuevos. En la figura 5.9 se puede ver el comportamiento del esquema cuando la inhibición cognitiva está activada. Utilizamos el mismo escenario que en la prueba anterior. El comienzo de la prueba es exactamente igual. El sistema reconstruye en 3D los tres puntos uno por uno. En la cuarta imagen se puede ver cómo ha actuado la inhibición cognitiva y en vez de tomar, como en el experimento anterior, el punto *1* como nuevo, simplemente lo mira para refrescarlo, acción mucho más rápida que reconstruirlo nuevamente. Justo en ese instante introducimos, al igual que antes, un punto nuevo (imagen 4). Esta vez, gracias a la inhibición cognitiva el sistema se da

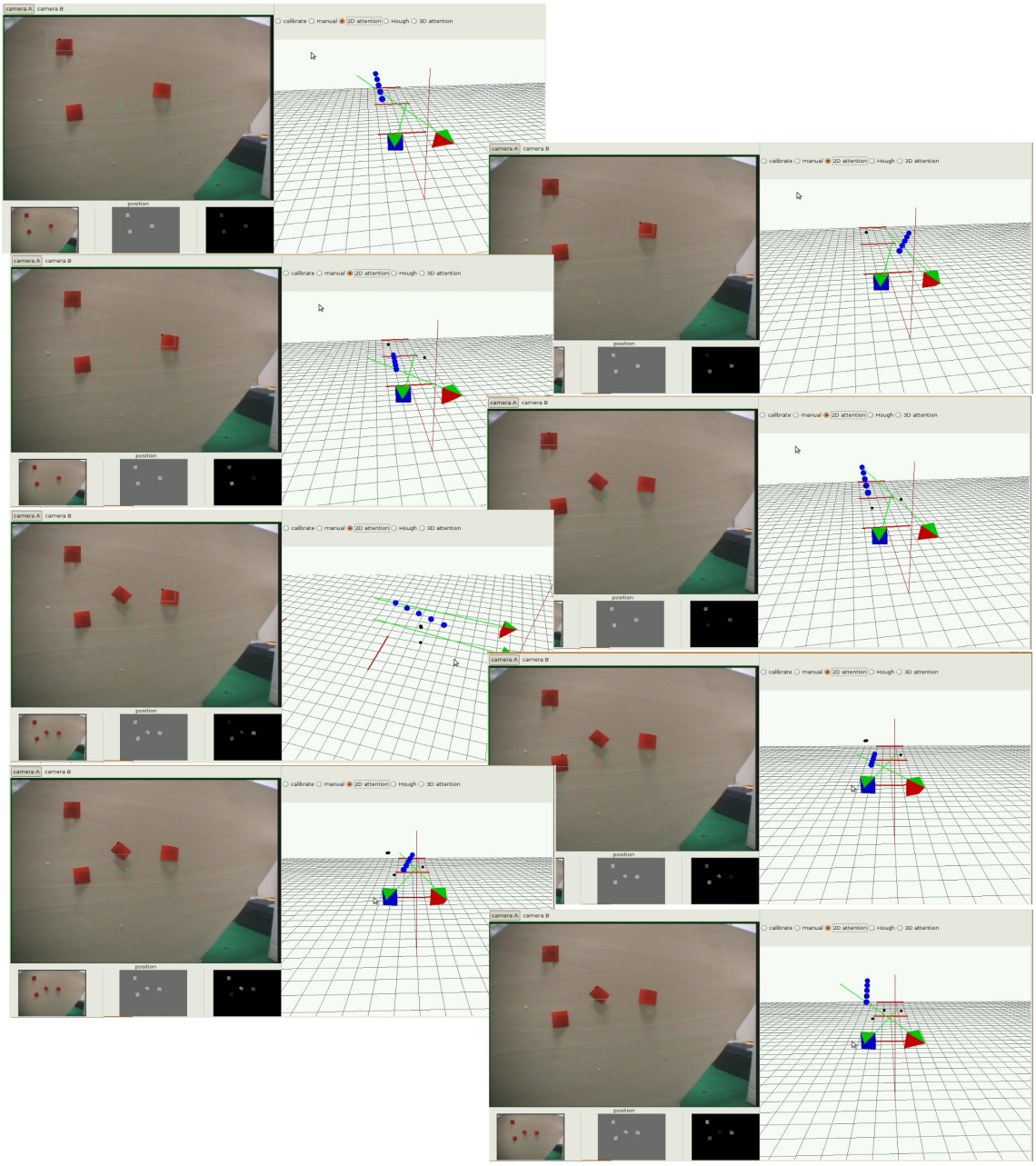


Figura 5.8: Inhibición cognitiva desactivada.

cuenta inmediatamente de que hemos hay un punto nuevo y pasa a reconstruirlo en 3D (imagen 5). En la última imagen se ve cómo el sistema, al no tener más puntos nuevos que reconstruir, continúa con el refresco de los puntos conocidos.

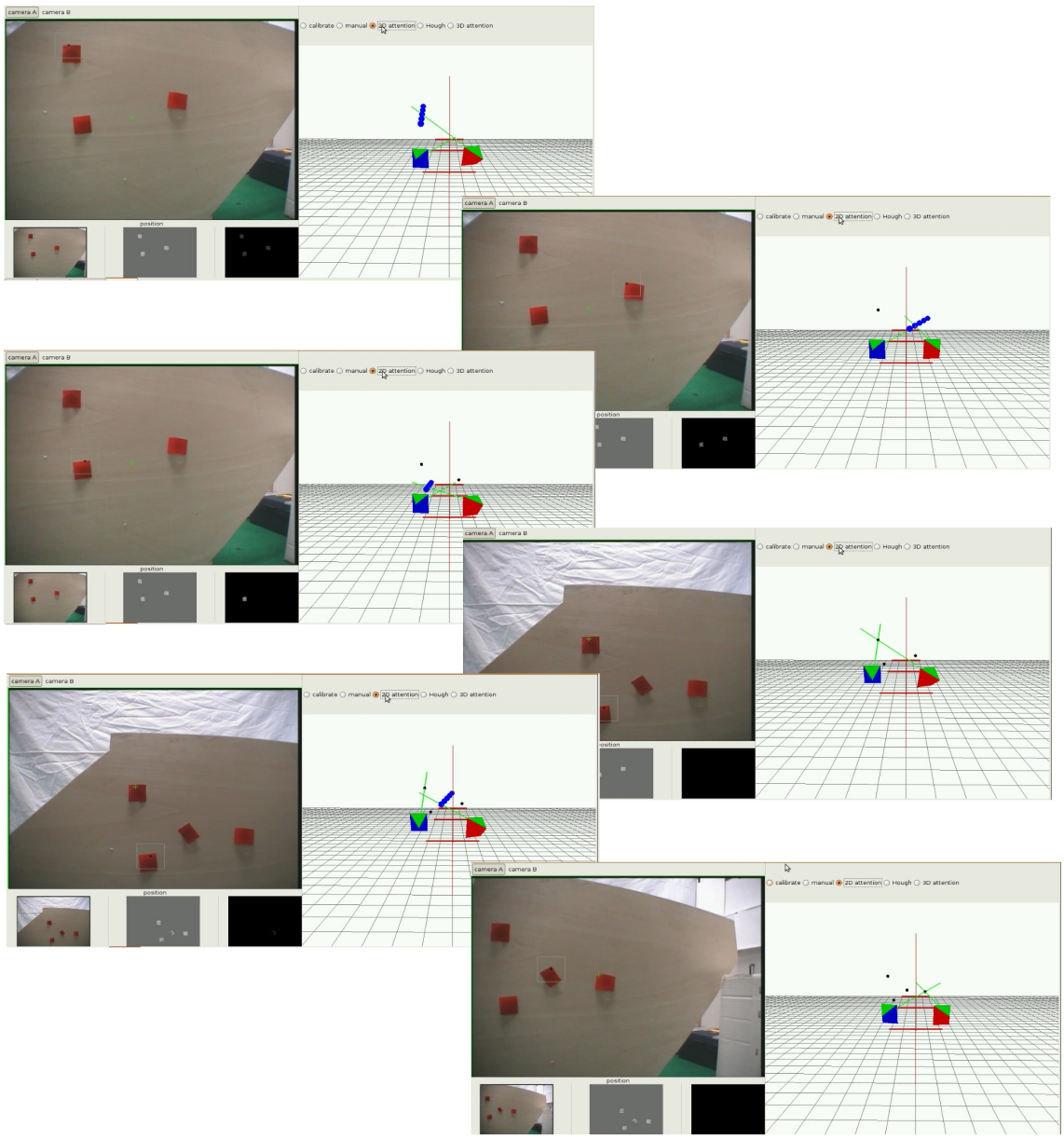


Figura 5.9: Inhibición cognitiva activada.

Como conclusión de este experimento podemos decir que la inhibición cognitiva le da mucha potencia a este proyecto. Gracias a ella el sistema no reconstruye contínuamente puntos, sino que también refresca los ya conocidos. Esto acelera de manera notable el seguimiento de varios puntos en un escenario. También es muy interesante el aspecto

de detectar los puntos nuevos e inmediatamente reconstruirlos. Hace que el sistema sea muy rápido incorporando novedades.

5.6. Tiempos y precisión

El último experimento lo hemos dedicado a recuperar estadísticas de nuestro algoritmo de reconstrucción 3D. Este algoritmo está basado en el movimiento de las cámaras.

Comparándolo con el algoritmo tradicional de cálculo de epipolar y triangulación, ya *a priori* sabíamos que los resultados iban a ser menos precisos simplemente por el hecho de que el algoritmo está basado en un cuello mecánico. El cuello al ser mecánico, no llega a alcanzar la precisión del algoritmo clásico, pero hemos conseguido unos resultados aceptables.

Para estas pruebas hemos utilizado el mismo escenario que usamos para probar el modelo de movimiento, sólo que esta vez, hemos picado manualmente en los puntos con posiciones conocidas tal y como aparecen en la imagen de la cámara dominante y hemos emparejado utilizando la técnica de vergencia con holgura.

Esta técnica ya la hemos explicado en el capítulo 4. La vergencia con holgura pretende amortiguar el error producido por las pequeñas imperfecciones que pueda tener el cuello mecánico. Los resultados han sido los siguientes:

Punto	Coordenadas originales	Vergencia con holgura	Error (cm)
P1	(80,30,2)	(78'6,28'3,3'1)	2,46
P2	(110,16,2)	(106'8,20,4'5)	5,7
P3	(100,-15,0)	(99,-17,1'1)	2,49
P4	(73,-35,0)	-	-
P5	(100,-15,50)	(97'3,-16'82,50'15)	3,26
P6	(73,-35,50)	(70'6,-35'3,49'1)	2,58
		Error medio	3,29

Cuadro 5.2: Tiempos generales de representación 3D.

En general hemos obtenido mejores resultados aplicando vergencia con holgura. La media de error está en 3,29 cm frente a 3,55 que presenta de media de error la vergencia pura. En cuanto a precisión no hemos ganado mucho. Pero donde mejor se ven las cualidades de la vergencia con holgura es cuando espaciamos la distancia entre

los puntos candidatos. Cuanto más separados están unos de otros, más imprecisa es la reconstrucción, pero con vergencia con holgura esta imprecisión es menor.

Para reducir los tiempos de reconstrucción 3D se puede discretizar el rayo de retroproyección en un menor número de puntos. Gracias a la vergencia con holgura no se pierde apenas precisión, pero se gana bastante tiempo.

Capítulo 6
Conclusiones y trabajos futuros

En los capítulos anteriores hemos descrito el problema que abordamos en el proyecto, los objetivos iniciales y la solución propuesta junto con una serie de experimentos que la validan y caracterizan. En este capítulo expondremos las conclusiones obtenidas y las posibles líneas por las que se puede continuar la investigación.

6.1. Conclusiones

Los objetivos marcados inicialmente para este proyecto fin de carrera implicaban que la aplicación fuera capaz de dotar al robot de un sistema perceptivo con atención visual mediante el uso de un par estéreo de cámaras móviles. 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.

La conclusión general es que hemos conseguido desarrollar un sistema perceptivo para el robot muy completo. El sistema es capaz de reconstruir en tres dimensiones los objetos de interés de un escenario y, posteriormente, realizar un seguimiento de ellos refrescándolos u olvidándolos. Los objetivos y subobjetivos que nos habíamos propuesto en un principio, comentados en el segundo capítulo, han sido cubierto con éxito.

1. **Sistema de atención 2D**: En este subobjetivo se proponía la realización de un sistema de atención que trabajase en el espacio 2D y que prestara interés a las características relevantes de la imagen. El sistema debía ser configurable para poder adaptarse a distintos escenarios y, por último, debía hacer un reparto de la mirada eficiente para evitar devolver siempre los mismos puntos interesantes.

   El sistema de atención desarrollado funciona correctamente tal como se puede ver en los experimentos del capítulo 5. Para desarrollar este bloque hemos aprovechado la investigación realizada por Roberto Calvo [Calvo Palomino, 2008]

en su proyecto de fin de carrera. El sistema de atención 2D cuenta con una dinámica de saliencia que nos permite realizar un reparto de la mirada eficiente. El sistema es configurable pudiendo elegir qué características le llaman la atención a partir del análisis directo de las imágenes. La saliencias implementadas son: bordes, color y segmentos. También tenemos la opción de hacer una combinación de estas.

De las tres saliencias implementadas, la más interesante es la de segmentos. Esta saliencia ha sido un aporte genuino del proyecto. Nos permite reconstruir mucho más rápido un escenario.

Otro aporte novedoso de nuestro proyecto ha sido la inhibición cognitiva. Tal y como contábamos en capitulo cuarto, la inhibición cognitiva nos relaciona la memoria 3D de puntos con el mapa de saliencia actual. Con esto evitamos reconstruir varias veces un mismo punto y aceleramos el proceso de refresco de los puntos 3D. La inhibición cognitiva aporta al sistema de atención 2D una capa de conocimiento de más alto nivel.

2. **Reconstrucción 3D**: En este otro subobjetivo se pedía un sistema de reconstrucción 3D que se alejase del mecanismo clásico de cálculo de epipolar y triangulación. Además, se obligaba a que trabajase en el espacio tridimensional evitando, en la medida de lo posible, operar en el plano imagen. También se pedía que el algoritmo estuviese basado en el movimiento de las cámaras.

   Para resolver este subobjetivo hemos creado un algoritmo de reconstrucción 3D que trabaja en el espacio tridimensional, como nos habían pedido. Para poder relacionar los movimientos del cuello mecánico con el espacio 3D, hemos desarrollado un modelo de movimiento que nos permite mover libremente las cámaras. Como se puede ver en el experimento 5.1, el modelo creado es robusto, fiable y preciso. El modelo de movimiento ha sido una pieza clave en nuestro proyecto, ya que uno de los requisitos fundamentales era desarrollar un sistema perceptivo con cámaras móviles.

   Mediante movimientos de vergencia hemos optimizado la búsqueda del píxel interesante de la imagen principal en la cámara secundaria. Esta técnica limita la búsqueda del píxel homólogo a una serie de puntos 3D pertenecientes a la recta retro-proyectada del píxel interesante. También hemos probado una mejora respecto al algoritmo de vergencia pura, la vergencia con holgura. Con esta alternativa intentamos superar las pequeñas imprecisiones de calibración que puedan tener las cámaras.

Esta técnica es innovadora y hace un uso intensivo del movimiento de las cámaras, tal y como nos habíamos exigido en el capítulo dos.

3. **Atención 3D**:El último de los subobjetivos era el de crear un sistema de atención 3D que gestionase la memoria de puntos 3D. El sistema de atención tridimensional que hemos desarrollado es capaz de repartir la mirada entre los distintos puntos interesantes. Estos puntos pueden no estar en el campo visual de las cámaras en ese momento, ya que se trata de sistema de atención *overt*.

   El sistema realiza un seguimiento de los puntos interesantes de la memoria 3D. Para ello hace un reparto de la mirada entre los puntos de la memoria para refrescarlos u olvidarlos. Este sistema hace uso de dos dinámicas que trabajan concurrentemente: la dinámica de saliencia y la dinámica de vida.

   Para obtener más información de la escena hemos diseñado la memoria para que sea capaz de almacenar primitivas en vez de puntos sólamente. La tres primitivas con las que trabajamos son: punto, segmento y cuadrado. Por último, hemos diseñado un algoritmo de atención cognitiva. Este algoritmo analiza los puntos de la memoria 3D y realiza hipótesis perceptivas que presenta puntos 3D a donde conviene mirar para confirmar o desechar esa hipótesis. Este cálculo lo hace en el espacio tridimensional. En concreto hemos implementado el concepto perceptivo "cuadrado". A partir de tres esquinas conocidas de un cuadrado somos capaces de calcular y mirar el punto donde creemos que está el punto. Con este aporte se le añade una rasgo inteligente al sistema. Este aporte es genuino del proyecto y es una característica muy interesante.

El sistema ha sido validado y experimentado con cámaras e imágenes reales. Los diferentes experimentos nos han sido de gran ayuda para ir comprobando cómo se van cumpliendo cada uno los objetivos.

Además de los objetivos, se han cumplido los requisitos funcionales exigidos. Los hemos resuelto de la siguiente manera:

1. El diseño se ha concebido siguiendo la arquitectura de *JDE*. De este modo se ha implementado la aplicación en forma de esquema. El esquema está escrito en lenguaje C y algunas partes en C++.

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

3. Hemos utilizado imágenes de tamaño 640x480 píxeles.

4. Se ha utilizado el modelo *EVI D-100P* de *Sony* como cámaras móviles.

5. El algoritmo de reconstrucción 3D se ha basado en la técnica de vergencia usando el movimiento de las cámaras. Lo hemos explicado detalladamente en el capítulo 4.

Este proyecto de ingeniería tiene un marcado carácter de investigación. Al principio se tenía el problema de un contexto bastante abierto del que no se sabía hasta dónde íbamos a llegar, o cuan positivos iban a ser los resultados obtenidos. 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.

Hemos seguido una línea de investigación incremental con varios antecedentes de los que hemos aprovechado y, a veces reutilizado, su *software* e ideas. A grandes rasgos este proyecto fin de carrera ha supuesto un paso adelante significativo sobre los proyectos de [Calvo Palomino, 2008], [León Cadahía, 2006] y [Martínez de la Casa Puebla, 2005]. Todos ellos desarrollados sobre la misma temática de atención visual.

Son aportes genuinos de este proyecto:

1. La inhibición cognitiva que relaciona el sistema de atención 2D con la memoria de puntos 3D.

2. La reconstrucción de una escena a partir de segmentos. Esto aporta velocidad mucha velocidad a la reconstrucción 3D.

3. El modelo de movimiento que nos permite mover libremente las cámaras sin perder su caracterización. Relaciona las posiciones de *pan* y *tilt* del cuello mecánico con el espacio tridimensional.

4. El algoritmo de reconstrucción 3D basado en el movimiento de las cámaras.

5. El almacenaje de puntos en una memoria 3D para su posterior gestión a través del sistema de atención 3D.

6. La carga cognitiva del sistema, dándole inteligencia a este mediante el algoritmo de atención cognitiva. Se ve muy bien su aplicación en el experimento del cuadrado.

7. Utilización de un tamaño de 640x480 para el análisis de imágenes, que dan una mayor nitidez y precisión al sistema.

8. Los aportes a la infraestructura de JDE como son el *driver vídeo4linux2* y las mejoras realizadas en el calibrador de cámaras.

Los resultados de este proyecto,documentación, código, vídeos y demás material está disponible en la página web de JDE[1].

6.2. Trabajos futuros

Uno de los caminos más interesantes por los que podríamos continuar este trabajo es intentar hacer una reconstrucción más rápida de la escena. Ahora mismo en este proyecto se trabaja con escenas estáticas donde apenas hay cambios. Una de las ideas para conseguir este objetivo es la implementación de un algoritmo híbrido que implemente un sistema de atención *overt* y un algoritmo de reconstrucción más rápido que el nuestro.

También sería muy interesante ampliar las posibilidades de representación de la información. En este momento sólo podemos almacenar puntos, segmentos y cuadrados, pero cabría la posibilidad de representar la información mediante primitivas más abstractas y potentes como cubos, personas, etc.

Podríamos incrementar el número de objetos sobre los cuales el sistema es capaz de hacer una hipótesis. En este proyecto hemos realizado el experimento sobre cómo averiguar la cuarta esquina de un cuadrado. Del mismo modo, podríamos realizar hipótesis perceptivas sobre una cara humana; presentando atención a la cara, podemos hipotetizar donde estarían los ojos y saltar a esa zona de una manera muy rápida.

[1] http://jde.gsyc.es/index.php/Gdago

Bibliografía

[Calvo Palomino, 2008] Roberto Calvo Palomino. Reconstrucción 3d mediante un sistema de atención visual. *Proyecto fin de carrera - 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.

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

[Kachach, 2008] Redouane Kachach. Calibración automática de cámaras en la plataforma jdec. *Proyecto fin de carrera - 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 - Universidad Rey Juan Carlos*, 2006.

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

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

[Pacios y Cañas Plaza, 2007] Esteban Pacios y Jose María Cañas Plaza. Reconstrucción 3d visual con triangulación. *Proyecto fin de carrera - Universidad Rey Juan Carlos*, 2007.

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

[Shreiner *et al.*, 2007] Dave Shreiner, Mason Woo, Jackie Neider, y 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 - Universidad Rey Juan Carlos*, 2008.