INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2006-2007

Proyecto Fin de Carrera

Reconstrucción 3D visual con algoritmos evolutivos

Tutor: José María Cañas Plaza.

Autor: Iván García Martínez.

Para todas las personas con las que comparto grandes momentos de mi vida.

ivan.gm.85@gmail.com
jmplaza@gsyc.escet.urjc.es

Agradecimientos.

Son muchas las personas que debería agregar en estas líneas por todo su apoyo personal, valor humano, simpatia, interes y bondad.

En primer lugar quiero dedicarle este proyecto a mi familia y especialmente a mi primo David Jiménez por haber sido una bellisima persona y todo un ejemplo a seguir. Nunca te olvidaremos.

Quiero dar las gracias a José María Cañas por su ayuda durante la realización de este proyecto, su gran asesoramiento y conocimientos aportados y su envidiable dedicación hacia su trabajo.

Mi más sincero agradecimiento a mis compañeros de batalla José María Esteban, Ricardo Palacios, Miguel Ángel Tomé y Ángel Cortes por su interes y cercanía.

Deseo mencionar también a mis grandes amigos Raul Díaz, Esther Peloche, Eduardo Parada, Alberto Gallardo, Ana Isabel Jiménez, Ana Jara, Eva Jara, Emilio Criado, Inma, Iván Robles, Isidoro Hidalgo y a todos mis compañeros de clase por ayudarme día a día en todos los aspectos.

También me gustaría mencionar a mis compañeros de trabajo y alumnos ya que me ayudan a pasar muy buenos momentos en su compañia, especialmente a Alberto Huerta y Borja por su amistad, apoyo e interés.

Finalmente quiero expresar mi aprecio y cariño a mis padres y mi hermana por darme todo lo que más necesita una persona en todos los momentos de su vida.

Enhorabuena a todos.

Resumen.

El uso de cámaras en cualquier sistema informático cotidiano, como un teléfono o una videoconsola, está ayudando a disminuir los costes de estos sensores y a aumentar su uso por parte de programadores. Extrayendo la información aportada por las cámaras e interpretándola podemos crear aplicaciones que incorporen comportamientos inteligentes que pueden ser útiles para realizar tareas rutinarias. Uno de los problemas fundamentales con el que se encuentra la visión artificial es *interpretar la información obtenida de las cámaras.*

La finalidad de este proyecto es crear una aplicación capaz de *reconstruir objetos de nuestro entorno* estimando la posición tridimensional de sus bordes a partir de un par estéreo de cámaras.

Se han desarrollado tres *algoritmos genéticos* como método de búsqueda en el espacio de posibles interpretaciones de la escena observada a través de las cámaras. Estos algoritmos hacen evolucionar a una población de individuos utilizando *operadores genéticos* y asocian a cada individuo una *salud* dependiendo de factores que indican si un individuo es mejor, o no, que otro. La primera de las técnicas utiliza como individuos de la población puntos tridimensionales basandose en el conocido algoritmo de moscas. La segunda utiliza segmentos 3D de longitud fija como individuos de la población para aportar a la reconstrucción 3D una mayor cantidad de información y operadores adaptados a esta primitiva. La tercera utiliza segmentos 3D de longitud variable que incorpora nuevos operadores destinados al aumento y reducción de longitud, movimiento, cambio de orientación y fusión de las primitivas segmento, y una nueva reestructuración de la población que permite reconstruir múltiples objetos.

La aplicación ha sido desarrollada apoyándose en la *plataforma software jde.c* e implementada en forma de hebras iterativas o *esquemas*. Se han realizado numerosos experimentos con imágenes sintéticas hasta conseguir que este sistema sea lo suficientemente *vivaz* como para obtener la posición de un objeto, ofrezca una *precisión centimétrica* y funcione con *hardware convencional.*

Índice general

Índice de figuras

Capítulo 1
Introducción

En este primer capítulo se va a dar una visión global del contexto del presente proyecto. Para un robot sería una gran fuente de información poseer una vista y un cerebro, tan adaptado al tratamiento e interpretación visual, como el de los seres humanos. Una de las tendencias de la *robótica* es conseguir procesar imágenes de manera similar a como lo hace un ser humano. La *visión artificial* estudia esta fuente de inspiración y desarrolla técnicas computacionales que también pueden ser utilizadas para la *localización y reconstrucción* en 3D.

1.1. La robótica

Los primeros robots creados en toda la historia de la humanidad tenían por finalidad entretener a sus dueños. Estos inventores se interesaban solamente en conceder los deseos de entretener a quien les pedía construir el robot o autómata. Sin embargo, estos inventores comenzaron a darse cuenta de que los robots podían imitar movimientos humanos o de alguna criatura viva. Jacques de Vauncansos construyó varios músicos de tamaño humano que esencialmente eran autómatas mecánicos. Henri Maillardert creó una muñeca que era capaz de hacer dibujos. Estos movimientos pudieron ser mecanizados, y de esta manera, se podía automatizar y mecanizar algunas de las labores más sencillas de aquellos tiempos. El origen del desarrollo de la robótica, se basa en el empeño por automatizar la mayoría de las operaciones que puede realizar un ser humano.

Durante la revolución industrial, en el siglo XVIII, se diseñaron nuevos instrumentos mecanizados dirigidos al sector de la producción textil; entre ellos destacan la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785) y el telar de Jacquard (1801) que se programaba con tarjetas perforadas.

El desarrollo en la tecnología de los años posteriores facilitó la inclusión de poderosas *computadoras* electrónicas, de *actuadores* de control retroalimentados, de trasmisión de potencia a través de engranajes y de *sensores* que han contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria. En la figura 1.1 se puede ver un ejemplo claro de cadena de montaje de vehículos, modelo clásico de representación de la industrialización de principios del siglo XX.

Figura 1.1: Cadenas de montaje del siglo XXI, izquierda; y del siglo XX, derecha.

Una obra Checoslovaca publicada en 1917 por Karel Kapek, denominada Rossum's Universal Robots, dio lugar al término *robot*. La palabra checa 'Robota' significa servidumbre o trabajador forzado y cuando se tradujo al inglés se convirtió en el término robot. El término robótica procede de la palabra robot. La robótica es la rama de la ciencia que se ocupa del estudio, desarrollo y aplicaciones de los robots.

Al oír la palabra robot, a menudo se produce en nuestra mente la imagen de una máquina con forma humana, con cabeza y extremidades. Esta asociación es fruto de la influencia de la televisión o del cine, cuyos anuncios o películas muestran máquinas con forma humana llamadas androides o humanoides, que generalmente son pura ficción. Los robots son usados, hoy en día, para realizar tareas sucias, peligrosas, difíciles o repetitivas para los humanos. Tareas de este tipo son, por ejemplo: la limpieza de centrales nucleares, donde se utilizan robots teleoperados; misiones espaciales a otros planetas; u operaciones quirúrgicas que pueden realizarse con el médico y el paciente

separados por grandes distancias. En la figura 1.2 podemos observar una máquina que, a pesar de no tener forma humana, sí que es un robot. Se trata de la aspiradora Roomba, capaz de aspirar una habitación completa sin intervención humana.

Figura 1.2: Aspiradora Roomba.

En la actualidad, los avances tecnológicos y científicos no han permitido todavía construir un robot realmente inteligente. En esta línea existen grandes firmas japonesas que están investigando y estudiando ciertos comportamientos humanos con la finalidad de crear asistentes personales robóticos. Por ejemplo el robot *Asimo* de la empresa japonesa Honda (figura 1.3), así como *Qrio* de la empresa Sony.

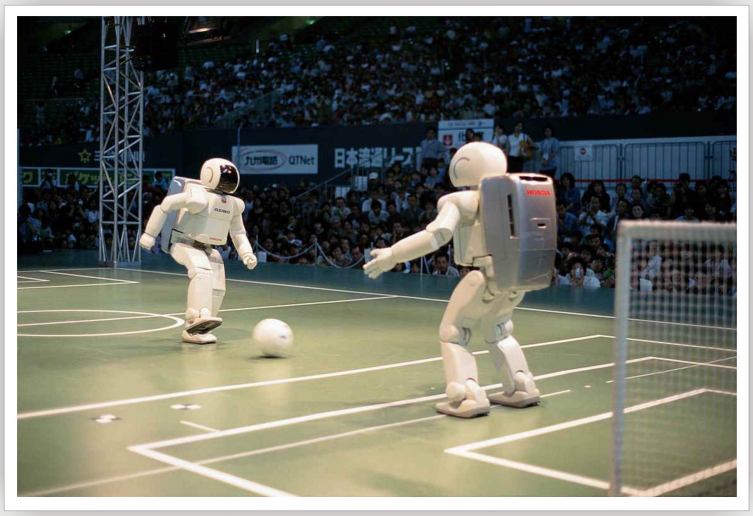

Figura 1.3: Robots Asimo jugando a fútbol.

En términos generales, los robots son dispositivos compuestos de *sensores*, *actuadores* y *procesadores*. Los sensores se encargan de recibir información o, lo que es lo mismo, datos de entrada que están conectados a uno o varios *procesadores*. El procesador al recibir la información de entrada, la interpreta y la organiza y posteriormente ordena a los actuadores que efectúen una determinada acción asociada al evento. El comportamiento autónomo del robot lo determina el software. Por eso la robótica es un campo interesante desde el punto de vista de la informática.

Actualmente las nuevas tendencias hacen que uno de los sensores principales de los robots sean las cámaras. Esto nos abre las puertas a un indudable área común entre la robótica y la visión artificial.

1.2. Visión artificial 3D

La visión artificial, también llamada visión computacional, en un término que ha sido muy utilizado en los últimos años y que tiende a ser confundido con otros conceptos. La visión artificial es el área de la inteligencia artificial que se encarga del análisis de imágenes por medio de ordenadores para conseguir extraer información con el fin de comprender y asimilar lo que está sucediendo en ellas.

Aunque a día de hoy el uso de la visión artificial está muy extendido, sus comienzos se remontan a mediados del siglo XX. Ya entonces se desarrollaban pequeños programas capaces de reconocer tanques o armas y obstáculos que pudiesen impedir el acceso a alguna zona.

Quizá el auge de la visión artificial surgió tras la llegada de ordenadores con velocidad de cómputo suficiente para procesar imágenes de forma ágil y con poca demora. A partir de entonces comenzó a emplearse para múltiples tareas y su desarrollo fue concentrándose en el estudio de técnicas para tratar de resolver algunos problemas con el tratamiento de las imágenes.

Algunas de estas técnicas son:

- *Filtrados*. Mediante el uso de esta técnica conseguimos obtener una nueva imagen en la cual sólo mostramos la información que nos es útil de la imagen original.

Ejemplos clásicos pueden ser el filtro de color, los suavizados y los realces de calidad.

- *Detección de bordes.* Consiste en reconocer cuales son los límites o bordes de los objetos de la imagen. Se suelen utilizar imágenes en escala de grises para determinar las áreas que tienen brillo uniforme, considerando como borde la frontera que separa estas áreas.

- *Segmentación.* Esta técnica se usa para aislar los elementos que nos interesan de la imagen agrupando los píxeles que cumplen unas determinadas características. Esta técnica nos facilita un posterior análisis y reconocimiento.

- *Localización.* Consiste en calcular, por medio de las cámaras de las que disponemos, la posición bidimensional o tridimensional de un objeto concreto en un entorno.

- *Reconocimiento de formas.* Esta idea propone que, tras haber analizado cierta información de un objeto determinado, seamos capaces de comparar dicho objeto con patrones conocidos con el fin de comprender o aproximar con qué tipo de objeto nos encontramos.

En la actualidad, los avances en geometría proyectiva, pares estéreo y autocalibración han abierto la puerta a la extracción de *información tridimensional* orientando las nuevas perspectivas hacia la *visión artificial 3D*. El uso de estos nuevos datos permite conocer mejor y con mayor precisión el entorno en el que se encuentran las cámaras.

Gracias al impulso de estas últimas técnicas hoy día es posible el reconocimiento automático de caras o del iris (figura 1.4), la localización de objetos tridimensionales que pueden ser utilizados en distintos ámbitos, la representación de piezas industriales, el reconocimiento de personas mediante el estudio biométrico en áreas de seguridad, el control de calidad, el diseño infográfico para la industria del cine, la programación de nuevos videojuegos más interactivos, la búsqueda de imágenes por su contenido, etc.

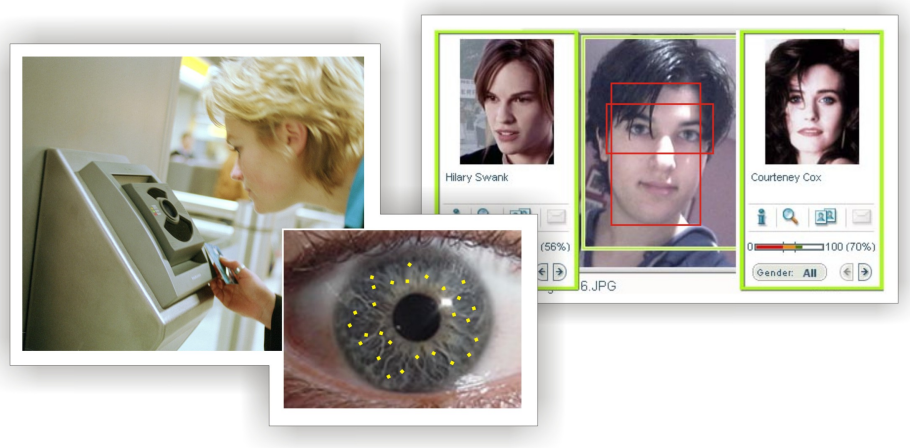

Figura 1.4: A la izquierda, reconocimiento de iris; a la derecha, reconocimiento facial.

Para un robot resulta muy interesante poder reconocer en 3D el mundo que le rodea para tomar buenas decisiones de movimiento, este es el motivo por el que las cámaras se han convertido en su sensor principal. La visión artificial 3D es muy importante en la robótica ya que nos ayuda a localizar objetos tridimensionalmente.

1.3. Localización y reconstrucción 3D

Dentro de la visión artificial, en robótica hay dos problemas clásicos la localización de objetos y la reconstrucción de los mismos, que realizamos posteriormente con mayor o menor detalle dependiendo de la cantidad de objetos localizados.

La localización en el mundo de la robótica consiste en obtener la posición real o aproximada de un objeto cualquiera en un mundo real o simulado. Por un lado podemos realizar una localización densa, en la cual la finalidad es obtener información de todos los puntos de la escena hasta conseguir identificar todos los detalles del entorno. Este método resulta computacionalmente muy costoso y por ello es una técnica difícil de llevar a cabo con las computadoras actuales. Por otro lado, se puede utilizar la localización de objetos, con la que prestamos especial atención a los objetos del entorno que sí son relevantes para nuestra solución final. Esta solución es mucho más efectiva que la localización densa ya que requiere mucha menos capacidad de cómputo. En muchas ocasiones podemos diferenciar cuales objetos nos interesan de la escena y cuales no son necesarios.

Un ejemplo clásico de localización orientada a objetos utilizando únicamente una cámara es el robot aibo (figura 1.5). Este robot es utilizado en el campeonato internacional *Robocup*[1] que consiste en utilizar diferentes robots para que jueguen a fútbol entre ellos. A los robots Aibo les interesa esencialmente detectar la posición de la pelota, la portería y el resto de jugadores.

Figura 1.5: Robot Sony Aibo.

Con los ejemplos presentados hasta el momento sólo atendemos a la localización bidimensional de los objetos en el entorno. Sin embargo, en numerosas ocasiones necesitamos no sólo saber la altura y la anchura, sino también la distancia a la que los objetos se encuentran. Esta es una de las labores más complicadas y de ello se encarga la *localización 3D*. La localización tridimensional consiste en evaluar la posición concreta del objeto en coordenadas (x,y,z), referidas a un origen o centro de referencia previamente establecido. Para lograr la localización 3D es necesario el uso de como mínimo dos cámaras debido a la ambigüedad de profundidad con una sola cámara.

Como contexto próximo a este PFC, un ejemplo de uso de localización 3D usando varias cámaras, dentro del grupo de robótica de la URJC[2], es el proyecto de *Localización 3D* [Pineda, 2006] que es una aplicación para detectar personas dentro de una habitación mediante cámaras estáticas.

Existen numerosas técnicas implementadas para obtener información de distancia utilizando un par estéreo. En la figura 1.6 podemos observar la técnica *Disparity*

[1] http://www.robocup.org
[2] http://gsyc.escet.urjc.es/robotica

Map que consiste en obtener, a partir de dos imágenes tomadas desde dos cámaras ligeramente separadas, una nueva imagen filtrada en la que mostramos una imagen, similar a las anteriores, en escala de grises. En esta imagen aparecen los objetos representados con un único color que representa, según la iluminación del mismo si se encuentra más lejos o cerca de la posición de las cámaras.

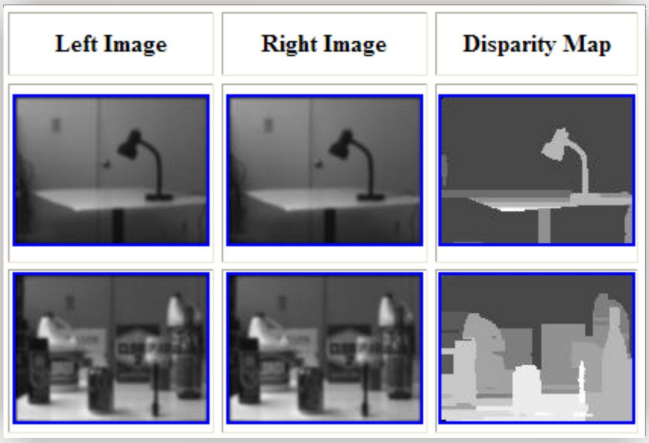

Figura 1.6: Disparity Map.

Con la técnica llamada *Space Carving* (figura 1.7) enfocamos un mismo objeto desde diferentes ángulos para después, utilizando algoritmos de localización, poder capturar y calcular todos los detalles del mismo. Una vez que tenemos toda la información podemos reconstruir el objeto con toda calidad de detalles.

Figura 1.7: Space Carving.

Entre las aplicaciones más conocidas para la localización 3D aparecen los *sistemas Vicon*[3] de captura de movimientos (figura 1.8) utilizados especialmente por industrias como la cinematográfica y de videojuegos para la captura de movimientos de personas. Consiste en grabar con múltiples cámaras a una persona vestida con un traje que lleva adheridos objetos en las extremidades capaces de reflejar fácilmente la luz infrarroja emitida por gran cantidad de diodos situados en cada cámara. Con la información que reciben dichas cámaras y con el uso de un potente computador podemos reconstruir los puntos del traje en movimiento.

Figura 1.8: Reconstrucción 3D utilizando sistemas Vicon.

Tras localizar cada uno de los objetos interesantes de la escena podemos obtener representaciones del entorno utilizando dicha información. En muchas ocasiones resulta especialmente importante realizar una reconstrucción e incluso memorización temporal de la realidad observada por las cámaras. Por este motivo la localización y la reconstrucción están íntimamente ligadas.

Una vez planteado el entorno general de la robótica, la visión artificial, la localización y reconstrucción 3D, la finalidad de este proyecto será conseguir reconstruir en 3D los objetos interesantes del entorno. Conseguiremos una aplicación capaz de obtener las imágenes de dos cámaras, extraer la información que proporcionan, localizar los objetos interesantes y reconstruirlos en una interfaz amigable.

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

Capítulo 2
Objetivos

Tras haber presentado el contexto general y particular sobre el que se asienta este proyecto fin de carrera, vamos a mostrar en este capítulo los objetivos concretos que se han propuesto y los requisitos que han condicionado el desarrollo del mismo.

2.1. Descripción del problema

El objetivo general del proyecto es *reconstruir en 3D los bordes de los objetos del entorno* desde las imágenes de un par estéreo de cámaras utilizando algoritmos evolutivos. Dicho objetivo lo hemos articulado en cuatro subobjetivos más específicos:

1. *Simulador de imágenes.* Los algoritmos genéticos *obtendrán las imágenes de entrada* observadas por un par estéreo de cámaras simuladas sobre un mundo virtual. Es necesario crear un mundo virtual que reconstruirá el laboratorio de robótica situando dentro de él al robot Pioneer junto con su par estéreo de cámaras. Se desarrollará un simulador que será capaz de reconstruir las imágenes que estén visualizando el par estéreo de cámaras de dicho robot en ese mundo virtual. Este simulador deberá utilizar la tarjeta gráfica, en lugar del procesador, para generar la escena utilizando la biblioteca OpenGL.

2. *Implementación del algoritmo genético de moscas* para interpretar lo que aparece en las imágenes obtenidas de las cámaras. Este trabajo pretende dar un paso más en la misma línea del algoritmo genético de Louchet [Louchet, 2000] para interpretar imágenes en tiempo real. Este algoritmo utilizará individuos puntuales y el nuevo operador de abducción.

3. *Diseño e implementación del algoritmo genético de segmentos* cuya población serán segmentos, se utilizarán nuevos operadores genéticos adaptados para tratar esta nueva primitiva y se desarrollarán nuevos criterios de cálculo de salud.

4. *Experimentos.* Con la intención de asegurar y cumplir lo más fielmente posible los requisitos del proyecto se realizarán gran cantidad de pruebas. También se realizarán comparativas entre los distintos algoritmos para obtener una valoración rigurosa del comportamiento de los mismos.

2.2. Requisitos

Los requisitos de partida para la aplicación que presenta este proyecto fin de carrera son:

1. Utilización del *lenguaje C* para programar la aplicación, apoyándose en el sistema operativo Linux.
2. Implementación del sistema dentro de la *plataforma software jde.c*[1] que proporciona imágenes de las cámaras reales y facilita la reutilización de lo desarrollado en este PFC.
3. Funcionamiento de la aplicación utilizando *computador y periféricos convencionales.*
4. Seguimiento de los objetos en movimiento.
5. Procesamiento de datos con *elevada vivacidad.*
6. Representación de objetos con *precisión centimétrica.*

2.3. Metodología y plan de trabajo

El plan de trabajo trazado para la realización de este proyecto sigue el *modelo de desarrollo en espiral basado en prototipos* (figura 2.1). Usando este modelo se aporta cierta flexibilidad en cuanto a posibles cambios de requisitos. Este modelo de desarrollo se caracteriza por la estructuración de las subtareas en un número determinado de ciclos. En cada uno de estos ciclos existen cuatro etapas: *Análisis de requisitos, Diseño e implementación, Pruebas y Planificación del próximo ciclo de desarrollo.*

[1] http://gsyc.escet.urjc.es/ jmplaza/software.html

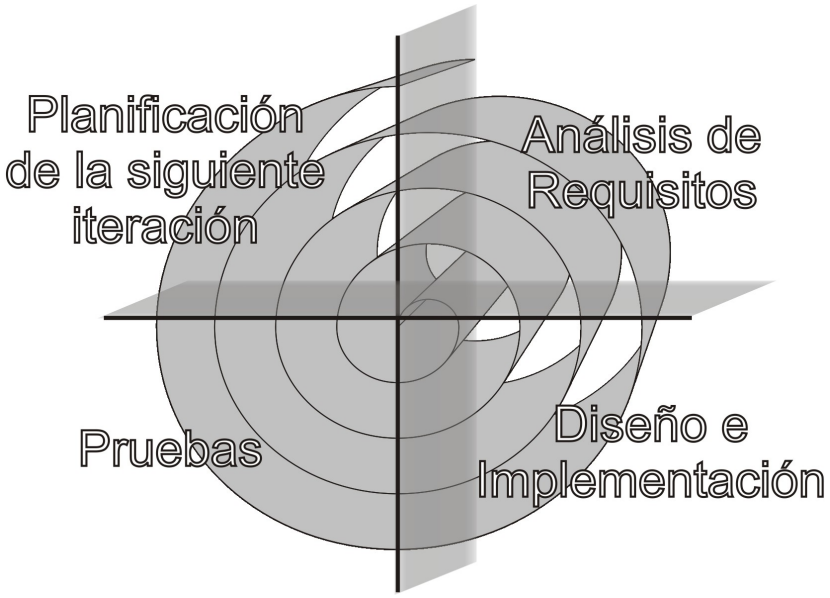

Figura 2.1: Modelo en espiral

Para el desarrollo de este proyecto se han completado los siguientes ciclos:

1. *Formación y familiarización con la infraestructura software.* En este ciclo inicial se pretende adquirir conocimientos sobre el *entorno linux y sus herramientas básicas* para instalación de paquetes y bibliotecas en el sistema Ubuntu; familiarización con la *plataforma software jde.c* [Plaza, 2004] y estudio a fondo de la estructura de la misma. Estudio y pruebas con el *lenguaje C* y la biblioteca gráfica *xforms*.

2. *Estudio de técnicas de visión artificial* tales como filtros de color y movimiento, conocimientos de espacios de color, soportes de vídeo, calibración y funcionamiento de controladores de cámaras USB y Firewire.

3. *Simulación con OpenGL.* Este ciclo de la espiral consistirá en la creación del simulador 3D utilizando la biblioteca OpenGL tras el aprendizaje de la misma. Se integrarán también las técnicas de visión implementadas previamente.

4. *Estudio del algoritmo genético de moscas puntuales.* En este ciclo se realizará un estudio de los algoritmos genéticos y del filtro de moscas adaptado a resolver el problema de la reconstrucción 3D. Se efectuarán una serie de mejoras sobre sus operadores y el cálculo de salud.

5. *Diseño e implementación del algoritmo genético de segmentos fijos.* Se implementará una nueva primitiva segmento en lugar de individuo puntual,

operadores genéticos para tratar estas nuevas primitivas y mejoras en el cálculo de salud.

6. *Diseño e implementación del algoritmo genético de segmentos variables.* El último ciclo de la espiral consistirá en desarrollar nuevos operadores genéticos para variar la longitud y orientación de las primitivas segmento, nuevos cálculos de salud adaptados a estas últimas mejoras y un nueva estructuración de la población para dar solución a la localización de múltiples objetos.

Durante todo el desarrollo del proyecto se han mantenido reuniones semanales con el tutor para establecer y afinar los puntos que se han llevado a cabo en cada etapa, y para comentar los resultados de etapas anteriores con la intención de conseguir realimentación continua.

Capítulo 3
Entorno y plataforma de desarrollo

En este capítulo se describe la infraestructura software sobre la que se apoya este proyecto fin de carrera. A grandes rasgos, la plataforma utilizada para este proyecto se resume en la figura 3.1. Los elementos Software más relevantes son el entorno jde.c 4.2 y las bibliotecas, xForms, OpenGL y Progeo.

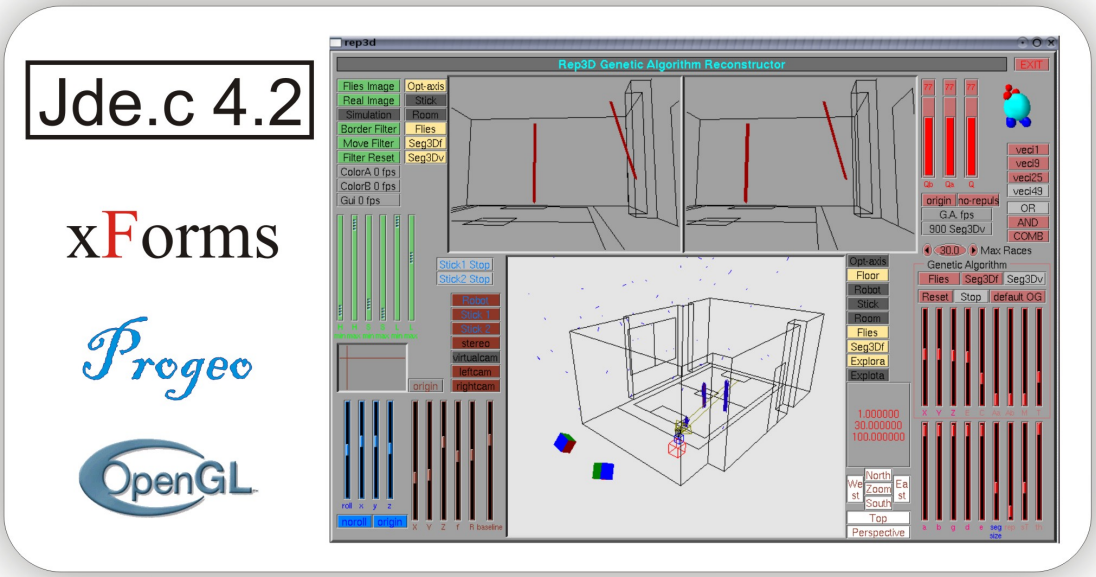

Figura 3.1: Plataforma utilizada.

3.1. Plataforma robótica jde.c

La aplicación se apoya en la plataforma robótica jde.c 4.2[1][2] (*Jerarquía Dinámica de Esquemas Versión 4.2*) que ha sido desarrollada en la Universidad Rey Juan Carlos. Esta plataforma permite desarrollar aplicaciones robóticas con sensores y actuadores, que tomen decisiones inteligentes de manera autónoma.

[1] http://gsyc.escet.urjc.es/ jmplaza/research-arch.html
[2] https://trac.robotica-urjc.es/jde/

La plataforma plantea las aplicaciones como un conjunto de *esquemas* que se ejecutan simultáneamente en hilos distintos y que realizan tareas sencillas y concretas. Existen esquemas de distintos tipos: *perceptivos* que se encargan de producir y almacenar información sensorial o elaborada por otros esquemas, y de *actuación* que son los que toman las decisiones para realizar una acción en función de la información sensorial. Este proyecto fin de carrera está compuesto por un conjunto de esquemas.

Las medidas sensoriales aparecen como variables que se leen y las órdenes a los actuadores se dan escribiendo en otras variables. Si bien la arquitectura software está principalmente orientada a la programación de comportamientos en robots, ha sido adaptada al caso concreto de este proyecto para *aprovechar de ella todos los elementos dedicados a la visión artificial*. Principalmente se utiliza el acceso a las imágenes de las cámaras, que mediante la interfaz proporcionada se actualizan en las variables perceptivas *colorA y colorB*.

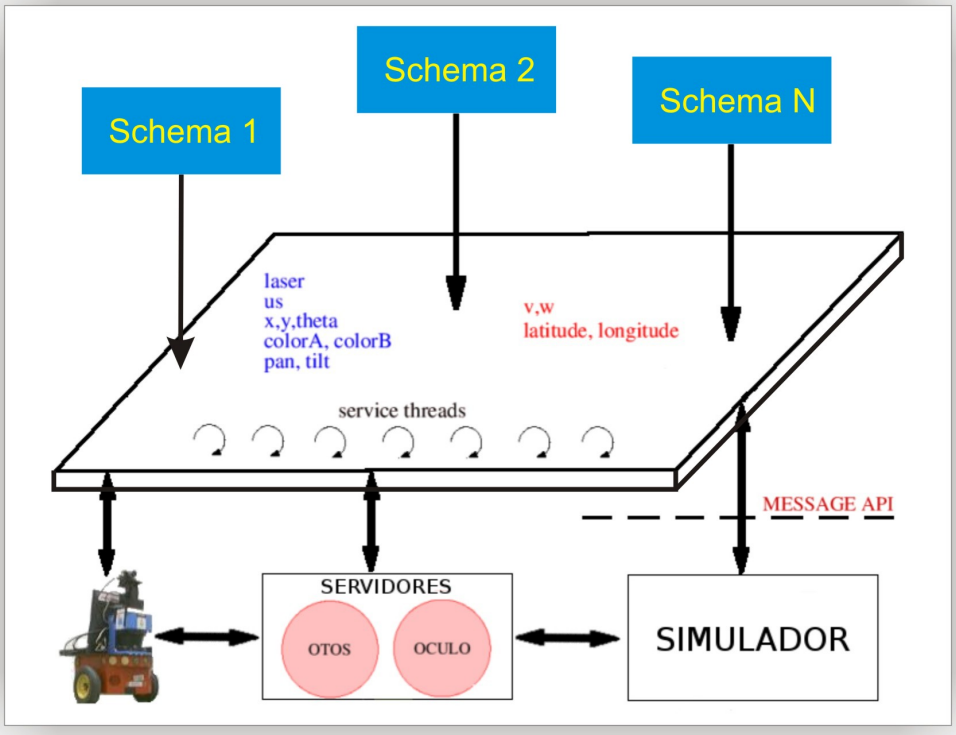

Figura 3.2: jde.c básico

3.2. Biblioteca gráfica XForms

La aplicación utiliza el sistema de interfaces gráficas proporcionado por la plataforma software jde.c 4.2 que se apoya directamente en la biblioteca de interfaces gráficas Xforms [T. C. Zhao, 2000]. En jde.c cada esquema posee su propia interfaz.

Con XForms es posible crear ventanas directamente sobre el sistema X Window de todo sistema operativo Linux. No se apoya en ninguna otra biblioteca o motor gráfico intermedio. Además proporciona una herramienta con la que poder crear visualmente la interfaz, con botones, barras de desplazamiento, menús o cuadros de imágenes llamada *fdesign*; y acceder a esos objetos desde el programa de la aplicación.

3.3. Biblioteca de gráficos 3D OpenGL

OpenGL[3] es una especificación estándar que define una API multi-lenguaje multi-plataforma para escribir aplicaciones que producen gráficos 3D, desarrollada originalmente por Silicon Graphics Incorporated (SGI). OpenGL significa Open Graphics Library [Mark Segal, 2001].

Entre sus características podemos destacar que es multiplataforma (puede encontrarse en Linux, Unix, MacOS, Windows, etc.), y que realiza la generación de gráficos 2D y 3D por hardware, ofreciendo al programador una API sencilla, estable y compacta. Además, su escalabilidad ha permitido que no se haya estancado su desarrollo, permitiendo la creación de extensiones (Como GLU o GLUT), que son una serie de añadidos sobre las funcionalidades básicas, en aras de aprovechar las crecientes evoluciones tecnológicas.

La biblioteca unida al objeto GLcanvas de xForms nos permite representar tridimensionalmente todo tipo de formas en nuestra interfaz gráfica. Esto aporta a este proyecto un factor muy importante para su desarrollo ya que, gracias a OpenGL, liberamos trabajo a la CPU en casi un 100 % en la tarea de representación o reconstrucción de resultados.

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

3.4. Biblioteca Progeo

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

- *Proyectar*. Esta función permite realizar la proyección geométrica de un punto 3D del espacio en el plano imagen de una de las cámaras, obteniendo así un punto 2D perteneciente a ese plano. Con ese punto 2D es posible obtener el pixel de la imagen correspondiente.
- *Retroproyectar*. Permite obtener para cada píxel 2D de una imagen, la recta 3D de todos los puntos del espacio que proyectan sobre dicho píxel.

Progeo se basa en el *modelo Pinhole de cámara* [Hartley y Zisserman, 2004] para realizar estas transformaciones, cuya figura descriptiva se puede observar en la figura 3.3.

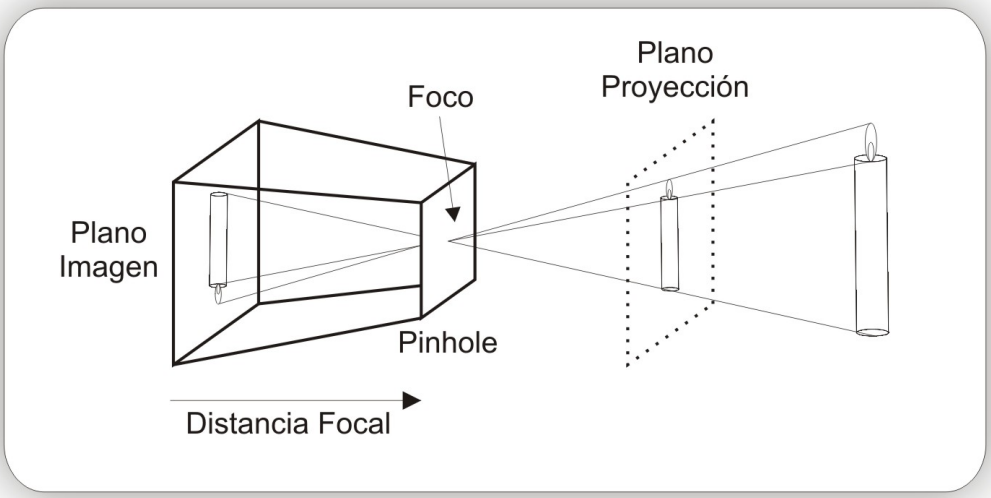

Figura 3.3: Modelo de cámara Pinhole

En este modelo se asume que cualquier punto $P(x, y, z)$ se proyecta en el plano de imagen a través de otro único punto llamado *Foco*. La recta que une el punto P y el centro óptico se denomina *Línea de proyección* e intersecta al plano imagen justo en el pixel $p(x', y')$, que es la proyección de $P(x, y, z)$. El centro óptico está situado a la *Distancia focal* del plano imagen. Este modelo lo completan el *Eje óptico*, que es

una línea perpendicular al plano imagen y que atraviesa al centro óptico, y también el *Plano focal*, que es el plano perpendicular al eje óptico cuyos puntos no se proyectan en el plano imagen, incluyendo al centro óptico.

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

Para la correcta simulación del entorno virtualizado es necesario que las cámaras se encuentre calibradas. El proceso de calibración consiste en obtener los *parámetros intrínsecos* y *parámetros extrínsecos* de las mismas. En el caso de los parámetros intrínsecos es necesario conocer la distancia focal y el píxel central del plano imagen. Para los parámetros extrínsecos se necesita la posición 3D de la cámara y su orientación.

Capítulo 4
Descripción informática

Tras haber presentado en capítulos anteriores los requisitos y herramientas usados en este proyecto, en las siguientes líneas se detalla el software desarrollado. Se comienza describiendo los fundamentos teóricos de los algoritmos evolutivos, se presenta el diseño global del programa, y se describen las implementaciones realizadas para cada una de las partes que lo forman.

4.1. Fundamentos teóricos de los algoritmos evolutivos

Los llamados *algoritmos genéticos o evolutivos* buscan una solución al problema planteado combinando las propiedades de los miembros de una población para obtener nuevas generaciones más robustas. Estos algoritmos se utilizan a menudo en tareas de optimización, diseño de controladores borrosos o sistemas de aprendizaje automático. Nosotros usaremos estos filtros para reconstrucción 3D desde visión. Para ello, contamos con un conjunto o población de N individuos donde cada n_i tiene asociado un determinado peso o salud h_i que se actualiza en cada iteración del proceso refinando iterativamente la población. Cada uno de los individuos es un punto en el espacio de soluciones. Dependiendo del propósito del algoritmo, un determinado individuo es una solución total (enfoque *Michigan*) o parcial (enfoque *Pittsburgh*).

Estos algoritmos se dividen fundamentalmente en dos fases que se ejecutan en cada iteración:

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

2. *Computación de la salud* para cada individuo. Se calcula cuán bueno o malo es el individuo como solución al problema en función de las observaciones realizadas.

La nueva población se genera gracias a los llamados *operadores genéticos*. Se encargan de generar la población en el siguiente instante utilizando en el proceso los individuos con mejor dotación cromosómica. Estas propiedades dependerán del propósito del algoritmo, y los operadores genéticos pueden ser diseñados expresamente para captar estos rasgos de manera más eficiente. Existen varios *operadores genéticos clásicos* que se suelen utilizar siendo adaptados al contexto específico de cada problema. Por ejemplo el *elitismo*, la *mutación aleatoria*, el *ruido térmico*, el *cruce* o la *repulsión*. El *elitismo* permite mantener intactos a los individuos de la población anterior que tengan una buena salud. La *mutación aleatoria* genera individuos en cualquier posición del espacio de soluciones. El *ruido térmico* genera individuos alrededor de otro de la población anterior. El *cruce* permite combinar las características de dos individuos para generar uno nuevo. La *repulsión* evita que dos individuos sean generados en posiciones demasiado próximas.

El *cálculo de salud* consiste en evaluar a cada uno de los individuos n_i a partir de unos criterios dependientes del problema concreto. Cada uno de estos criterios se corresponde con características que determinan la calidad del individuo. Cuantos más criterios satisface un individuo mayor será su valor de salud h_i que está definido por una *función de salud* variable según la importancia que posea cada uno de los criterios con respecto de los otros.

4.2. Diseño general con esquemas

Para dar solución a la reconstrucción 3D de los objetos interesantes de la escena vamos a usar algoritmos genéticos con tres tipos de individuos que son *moscas puntuales*, *segmentos 3D de longitud fija parametrizable* y *segmentos 3D de longitud variable*.

Estos tres algoritmos se han implementado independientemente en diferentes esquemas que funcionan en iteraciones correspondiendo a las iteraciones del algoritmo genético. En cada iteración necesitan información sensorial para calcular la posición 3D del objeto. Estas observaciones serán las *imágenes 2D* de un par estéreo de cámaras, que se obtendrán gracias a la plataforma software jde.c sobre la que se apoya la aplicación, y/o serán generadas por el propio esquema gracias al simulador 3D realizado.

La población de individuos es una representación 3D del entorno alrededor de las

cámaras que se va perfeccionando tras cada iteración de los algoritmos genéticos.

Desde un punto de vista de *caja negra* el diseño general de la aplicación se resume en la figura 4.1.

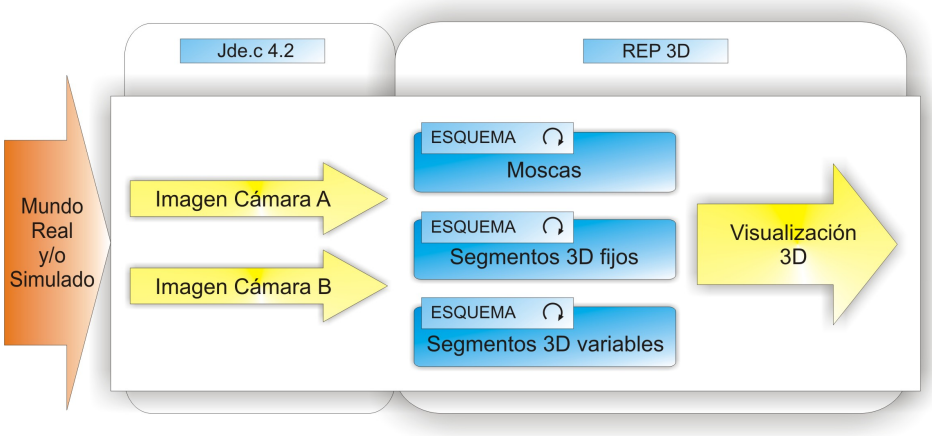

Figura 4.1: Representación del diseño software general de la aplicación.

Para visualizar los individuos y la reconstrucción la aplicación incorpora en la interfaz gráfica varias herramientas de visualización, de control y de configuración de los algoritmos.

4.3. Imágenes de entrada

Estos algoritmos utilizan dos imágenes correspondientes a un par estéreo de cámaras. Estas imágenes serán visualizadas siempre que se desee y pueden ser reales o simuladas con un tamaño de 320x240 píxeles.

En ocasiones es preferible utilizar un mundo simulado para probar los algoritmos con imágenes sin ruido ni distorsiones y desde cámaras perfectamente conocidas. Por ello se ha creado un mundo virtual correspondiente al laboratorio de robótica de la universidad. El robot y sus cámaras se encuentran en ese mundo. Dependiendo del lugar en el que se encuentren se generará una representación diferente en las imágenes del par estéreo que mostrarán el mundo virtual que estén observando las cámaras simuladas. La generación del mundo simulado se realiza utilizando la biblioteca OpenGL para liberar de tiempo al procesador en el tratamiento de estas imágenes.

OpenGL resulta muy útil para desarrollar imágenes virtuales debido a su compatibilidad con modelos de cámara entre OpenGL y Progeo y a su sencillez para representar objetos complejos. El siguiente código muestra un ejemplo sencillo de como representar una pared de color rojo utilizando las funciones de OpenGL.

```
glColor3f( 1.0, 0.0, 0.0 );
glBegin(GL_POLYGON);
v3f( 1.0, 1.0, 1.0 ); v3f( 1.0, -1.0, 1.0 );
v3f( 1.0, -1.0, -1.0 ); v3f( 1.0, 1.0, -1.0 );
glEnd();
```

4.3.1. Filtrado de color

Una vez que hemos capturado las imágenes reales y/o generado las imágenes simuladas del par estéreo se somete a ambas imágenes a un filtrado de color para obtener únicamente la información relativa al objeto u objetos que deseamos reconstruir.

En las imágenes obtenidas en los pasos anteriores el color de cada pixel viene representado en el espacio de color RGB (rojo, verde y azul). Sin embargo, un filtro de color para el espacio RGB no es un filtro robusto ante los posibles cambios de iluminación. Por ello, en este proyecto se ha optado por un espacio de color más robusto frente a cambios de intensidad luminosa: el *espacio de color HSI*. Un color en este espacio se representa mediante tres componentes: el *Tinte* (Hue) que es el color puro propiamente dicho; la *Saturación* (Saturation) que muestra la viveza de un color determinado; y la *Intensidad* (Intensity) que es la iluminación del color.

El espacio HSI trabaja directamente con una magnitud relacionada con la iluminación del color, que se puede ignorar en los filtros trabajando únicamente con las componentes H y S. La relación existente entre RGB y HSI se puede observar en las siguientes ecuaciones:

$$H = cos^{-1}\left(\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \tag{4.1}$$

$$S = 1-\frac{3}{(R+G+B)}min\,(R,G,B) \tag{4.2}$$

$$I = \frac{1}{3}(R+G+B) \tag{4.3}$$

Para *caracterizar el color interesante* se establecen unos umbrales máximos y mínimos por cada componente (Hmin, Hmax, Smin, Smax). Estos umbrales serán definidos a partir de un píxel elegido en el interfaz gráfico como muestra y una tolerancia establecida (*tolerancias de color*). Aquellos píxeles con componentes dentro de estos umbrales se consideran del color objetivo y computan en el cálculo de salud. Los píxeles que no superen alguno de los umbrales serán rechazados y visualizados por la interfaz en escala de grises (figura 4.2).

Figura 4.2: Par estéreo con filtro de color rojo HSI.

4.4. Algoritmo de moscas

En esta sección se describe una adaptación del algoritmo genético al problema de la reconstrucción 3D de objetos tomando moscas puntuales como individuos de la población[Louchet, 2000][Louchet, 2001][Jean Louchet y Boumaza, 2002]. En este diseño cada uno de los individuos contribuye parcialmente en la reconstrucción de la solución (enfoque *Michigan*) y está definido por un *punto 3D* de coordenadas (x,y,z) y por una salud h_i que indicará cómo de buena es esa solución teniendo en cuenta las observaciones en las imágenes del par estéreo de cámaras.

El esquema de la aplicación que materializa el diseño del algoritmo de moscas se denomina *flies*. En la figura 4.3 se puede observar un diagrama de flujo del algoritmo de moscas. Inicialmente, este esquema *distribuye las moscas uniformemente* sobre el espacio 3D; después entra en una sección iterativa para la *generación de la siguiente población*, sometiendo a las moscas a una serie de operadores genéticos; continúa con el *Cálculo de la salud de cada mosca* generada; y finalmente realiza un *ordenamiento de la población* de moscas según su salud.

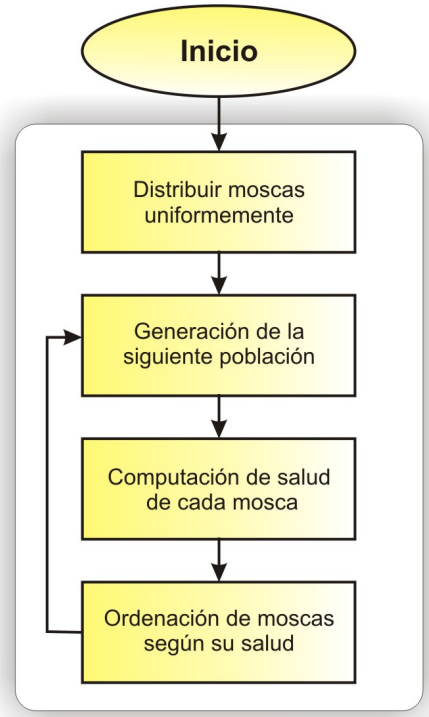

Figura 4.3: Diagrama de flujo del esquema flies.

4.4.1. Generación de la siguiente población

La *generación de una nueva población* se efectúa aplicando los *operadores genéticos*. Estos operadores se encargan de captar las mejores propiedades de las moscas de la población anterior y de observar las imágenes del par estéreo de forma que se consiga una nueva población que permita una localización más exacta de cada uno de los objetos:

- *El operador elitismo* elige las moscas con mayor salud de la población anterior y permite que permanezcan en la siguiente sin que sufran ningún cambio. La idea del operador consiste en no olvidar aquellas posiciones que ya eran muy buenas en la población anterior, dando estabilidad a la evolución.

- *El operador cruce* toma al azar dos individuos de la población anterior con muy alta salud (moscas elitistas), calcula la recta que los une y obtiene un punto aleatorio contenido en dicha recta. Este último punto del espacio 3D será el nuevo individuo de la población. Este operador permite explorar la zona existente entre dos individuos buenos, basándose en la idea de que es posible que ambos se encuentren en el mismo objeto (figura 4.4).

Figura 4.4: Esquema del operador de cruce para moscas.

- *El operador ruido térmico* se encarga de situar una nueva mosca en una posición aleatoria cercana a alguna otra mosca padre, entre los mejores, de la iteración anterior. Permite *explotar una zona* del espacio de soluciones distribuyéndose como ruido gaussiano o uniforme en las cercanías (figura 4.5).

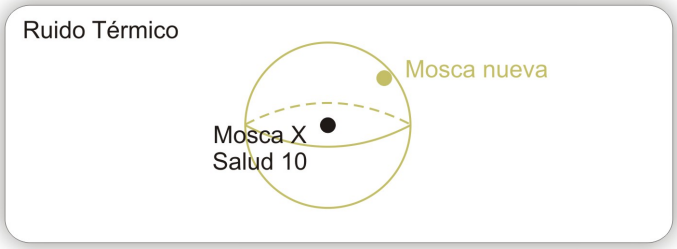

Figura 4.5: Esquema del operador de ruido térmico para moscas.

- *El operador mutación aleatoria* selecciona al azar una posición en el espacio 3D y sitúa allí una mosca. Este operador permite *explorar uniformemente el espacio* de la habitación. Sus probabilidades de éxito son bajas pero permite evitar los máximos locales en la búsqueda.

- *El operador repulsión* compara cada individuo creado tras usar otro operador genético con el resto de individuos de la población. Si resulta muy similar a otro individuo por estar demasiado cercano a él, se desecha y se vuelve a generar otro individuo distinto. Evita el innecesario solape entre individuos

- *El operador abducción.* Dada la implicación $a \rightarrow b$ podemos afirmar que *de a deducimos b* pero también que *de b abducimos a.* Aplicado a nuestro problema, la abducción nos permitirá suponer que si en una imagen de una cámara visualizamos un objeto que es compatible con el patrón de color buscado, el objeto en sí se encontrará en algún punto de la línea de proyección que une el centro óptico de la cámara con el punto donde intersecta al plano imagen. De

este modo, es razonable pensar que generando nuevas moscas a lo largo de esa línea de proyección, algunas puedan estar cerca del objeto (ver figura 4.6).

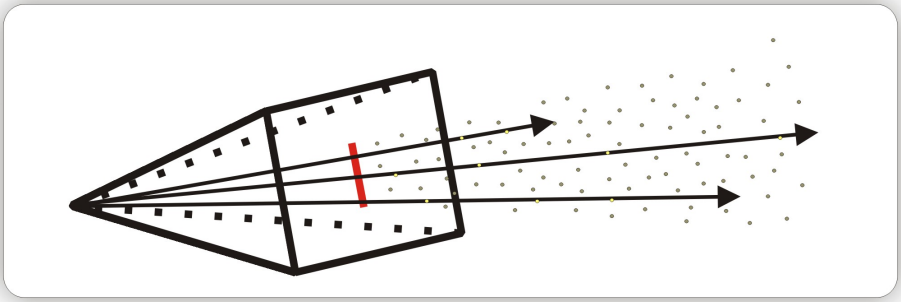

Figura 4.6: Moscas generadas a lo largo de líneas de proyección.

Basándose en este concepto, el *operador genético de abducción* genera nuevas moscas posicionándolas a lo largo de la línea de proyección de cada una de las cámaras con el objeto detectado. En cada iteración, siempre que se detecte algún objeto compatible con el color, podrá visualizarse un haz de moscas distribuidas a lo largo de las líneas de proyección de cada cámara. De este modo, el algoritmo busca en zonas del espacio de soluciones compatibles con las observaciones sensoriales. Para que el operador de abducción pueda funcionar correctamente y se pueda aprovechar al máximo sus posibilidades, es necesario contar con una *lista de píxeles* de cada imagen que sean compatibles con el patrón del color buscado. Esto se consigue *filtrando por completo las imágenes*. La inclusión de este operador en el algoritmo es completamente genuina, y supone una gran mejora con respecto a la versión que plantea Jean Louchet [Louchet, 2000]

La aplicación de algunos de estos operadores exige tener ordenada por salud la población. Además, la medida para utilizar un operador u otro para generar las nuevas moscas en la población se fija mediante *porcentajes*. Se establece un porcentaje a cada operador (exceptuando la repulsión) de manera que en cada iteración se asigna a cada operador el número de moscas que debe generar, del total de la población. Estos porcentajes han sido ajustados tras muchos experimentos hasta conseguir una configuración óptima que se detallará en la sección 5.1.

4.4.2. Computación de salud

Una mosca gozará de buena salud si proyecta dentro de las imágenes de las cámaras, si proyecta bien sobre el borde de algún objeto y si sus vecindades en ambas imágenes son similares. Este proceso se lleva a cabo para cada uno de los individuos-mosca de la población que obtendrán su valor de salud final según una serie de criterios. Inicialmente se *proyectará el punto 3D* que representa la mosca m_i sobre el plano imagen de la cámara c_m utilizando para ello la biblioteca Progeo (sección 3.4). De esta forma conseguimos las coordenadas (x,y) del plano imagen donde se visualiza la mosca (píxeles p_i y p_d). Una vez obtenidos los puntos-proyección 2D de la mosca m_i sobre las cámaras derecha e izquierda se procede a calcular los valores de varios criterios:

- *Criterio 1, proyecta en camI*: Si las coordenadas de proyección de la mosca m_i están dentro de la imagen de la cámara izquierda se continuará con el cálculo de salud, en caso contrario su salud será cero.

- *Criterio 2, proyecta en camD*: Este criterio es análogo al anterior pero utilizando para ello la cámara derecha. Con estos dos criterios se penaliza a las moscas que no proyectan dentro de las imágenes de ambas cámaras.

- *Criterio 3, vecindad en camI*: En este campo se analiza la ventana de vecindad en torno al píxel p_i que indica las coordenadas de proyección en el plano de la imagen de la cámara izquierda. Para ello se observará la zona de vecindad de dicho pixel, mirando en las imágenes filtradas por color previamente generadas. El valor máximo de este criterio es el número de píxeles de la zona de vecindad que expliquen la imagen filtrada por color, es decir, el número de píxeles que proyecten encima del objeto.

- *Criterio 4, vecindad en camD*: Es igual que el anterior pero utilizando el píxel p_d y la imagen de la cámara derecha. Con estos dos criterios se premia que la mosca proyecte sobre un objeto relevante compatible con el color filtrado en ambas cámaras.

- *Criterio 5, parecido de vecindad*: Mide el número de píxeles de la vecindad de la cámara izquierda que sean iguales que los de la vecindad de la cámara derecha. Con este criterio se consigue que el individuo tenga mejor salud cuanto más se aproxime en 3D al objeto sobre el cual se debe apoyar.

En la figura 4.7 se puede observar cómo se calculan los valores de vecindad definidos anteriormente para los criterios 3, 4 y 5 de la salud. En la figura se muestra un ejemplo

con vecindad 5x5 aunque en la aplicación se pueden utilizar vecindades de 1x1, 3x3, 5x5 y 7x7 seleccionables desde el GUI.

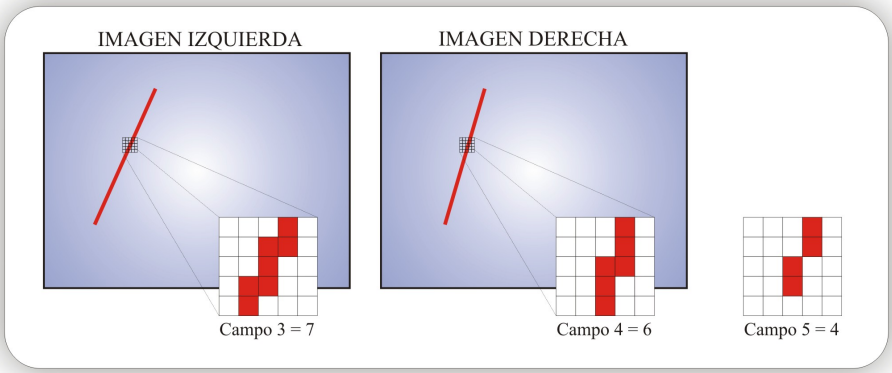

Figura 4.7: Cálculo de los campos 3, 4 y 5 utilizando vecindad 5x5.

Una vez calculados los valores de cada uno de los criterios se procede a su combinación lineal para calcular la salud h_i con la ecuación de comportamiento OR cuyos pesos se han ajustado experimentalmente:

$$h_i = \alpha * Criterio_3 + \beta * Criterio_4 + \gamma * Criterio_5$$

4.5. Algoritmo de segmentos 3D fijos

El segundo algoritmo genético desarrollado en este proyecto usa como individuos *segmentos 3D de longitud fija* que nos aportan mayor cantidad de información. En este diseño cada individuo es una parte parcial de la solución (enfoque *Michigan*). Cada segmento se representa de dos formas diferentes:

- *Coordenadas Absolutas*. Esta representación consta de los dos *puntos 3D* extremos del segmento de coordenadas P_1(x,y,z) y P_2(x,y,z).
- *Coordenadas Relativas*. Esta representación se compone de un *punto 3D* de coordenadas P_C(x,y,z) que es el centro del segmento, un *vector director 3D* v(x,y,z) de la recta que pasa por los puntos P_1 y P_2, y la *longitud* del segmento que es fija pero parametrizable. En la figura 4.8 se pueden comprobar estas partes del segmento.

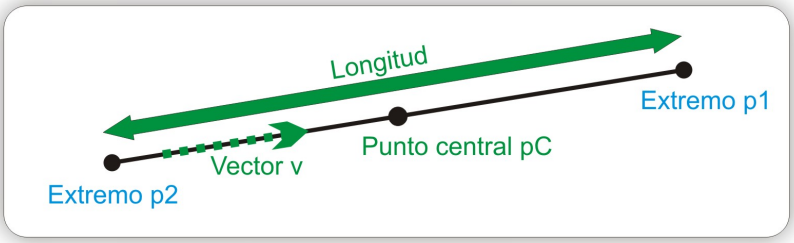

Figura 4.8: Representación de un segmento 3D.

Para este algoritmo genético todos los segmentos tendrán asignada una longitud fija que no variará durante la evolución de las poblaciones. Esta longitud se puede parametrizar utilizando la interfaz de la aplicación entre 0 y 300 mm.

El esquema de la aplicación que materializa el diseño del algoritmo de segmentos 3D de longitud fija se denomina *seg3df* siguiendo el diagrama de flujo de la figura 4.9 que es homólogo al diagrama de moscas (figura 4.3).

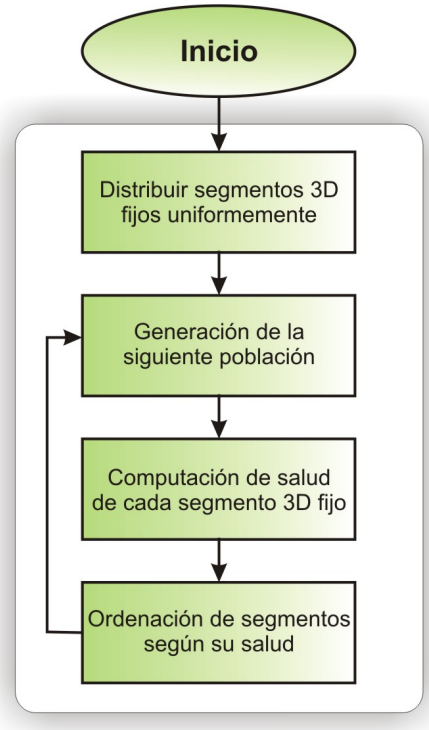

Figura 4.9: Diagrama de flujo del esquema seg3df.

4.5.1. Generación de la siguiente población

Este diseño incorpora los mismos operadores genéticos del algoritmo de moscas pero modificados para tratar segmentos 3D en lugar de individuos puntuales. Estos son:

- *El operador Elitismo* elige los segmentos con mayor salud de la población anterior y permite que permanezcan en la nueva población sin ningún cambio.
- *El operador Cruce* toma al azar dos individuos de la población anterior con muy alta salud (segmentos padre), calcula la recta que une los puntos centrales de ambos segmentos y obtiene un punto aleatorio contenido en dicha recta. Este último punto del espacio 3D será el centro del nuevo individuo que además tomará el vector director y la longitud del segmento padre que mayor salud tenga (figura 4.10).

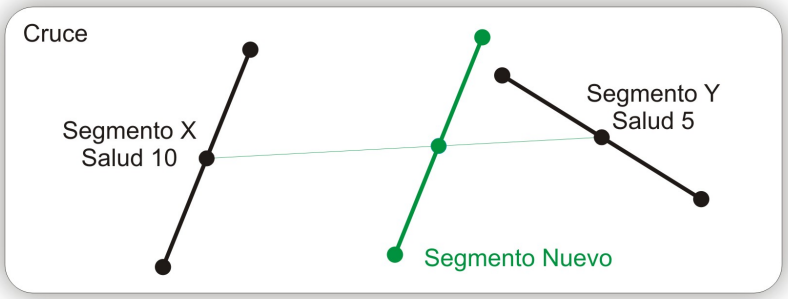

Figura 4.10: Esquema del operador de cruce para segmentos fijos.

- *El operador Abducción* genera nuevos segmentos posicionando sus puntos centrales a lo largo de la recta de proyección que se forma entre el foco de cada una de las cámaras y el píxel del plano imagen en el que proyecta el objeto. Una vez posicionado el punto central P_C del segmento, este toma la longitud fija correspondiente y una orientación. Esta orientación se consigue obteniendo un segundo punto 3D P_O, que se sitúa a lo largo de la línea de proyección que forme otro píxel del plano imagen que proyecte sobre el objeto, y calculando el vector director de la recta que une ambos puntos P_C y P_O. En cada iteración se podrá visualizar un haz de segmentos 3D distribuidos a lo largo de las líneas de proyección de cada cámara (figura 4.11).

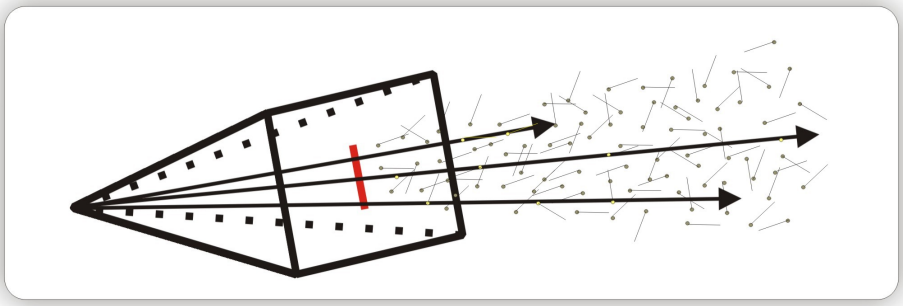

Figura 4.11: Segmentos fijos generados a lo largo de líneas de proyección.

- *El operador Ruido térmico* se encarga de situar un nuevo segmento en una posición aleatoria cercana al punto central de algún otro segmento de la población anterior. Permite *explotar una zona* del espacio de soluciones distribuyéndose como ruido térmico en torno a la misma (figura 4.12).

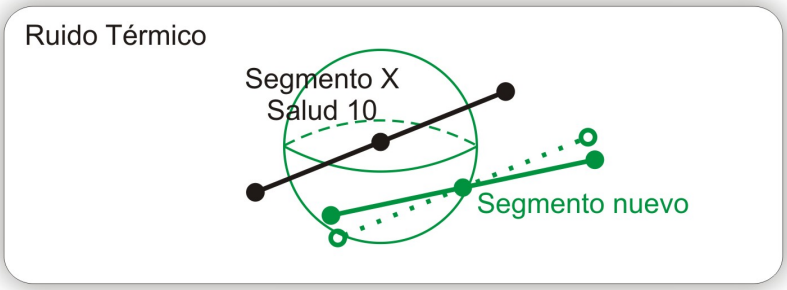

Figura 4.12: Esquema del operador de ruido térmico para segmentos fijos.

- *El operador Mutación aleatoria* selecciona aleatoriamente una posición en el espacio 3D y sitúa allí un segmento nuevo. Este operador permite *explorar uniformemente el espacio* de la habitación.

- *El operador Repulsión* se encarga de comparar al individuo creado tras usar otro operador genético con el resto de individuos de la población. Esta comprobación consiste en calcular la distancia a la que se encuentran los extremos P_1 y P_2 del segmento s_i con respecto al resto de segmentos de la población. Si en la comparación el nuevo individuo resulta ser muy similar a otro se desecha y se genera otro individuo nuevo en su lugar.

La aplicación de algunos de estos operadores exige tener ordenada por salud la población. Además, la medida para utilizar un operador u otro para generar los nuevos

segmentos en la población se fija mediante *porcentajes*. Se establece un porcentaje a cada operador (exceptuando la repulsión) de manera que en cada iteración se asigna a cada operador el número de segmentos que debe generar, del total de la población. Estos porcentajes han sido ajustados tras muchos experimentos hasta conseguir una configuración óptima que se detallará en la sección 5.1.

4.5.2. Computación de salud

Este proceso se lleva a cabo para cada uno de los individuos-segmento de la población que obtendrán su valor de salud final según una serie de criterios.

Inicialmente se *proyectan los puntos 3D* de los extremos y del centro del segmento s_i sobre el plano imagen de la cámara c_m utilizando la biblioteca Progeo (sección 3.4). De esta forma conseguimos las coordenadas (x,y) en el plano imagen de los tres puntos más significativos que forman el segmento. Posteriormente se procede a calcular los valores de cada criterio según la siguiente especificación:

- *Criterio 1, dentro del frustum*[1]: Este primer campo no da valor a la salud si no que es un requisito indispensable para que el individuo sea interesante. Por un lado se comprueba si las coordenadas del segmento s_i están dentro del frustum de las cámaras, es decir, que el segmento se encuentre a una distancia cercana a las cámaras. Por otro lado se comprueba si las tres coordenadas de proyección del segmento s_i están dentro de la imagen de la cámara izquierda y de la derecha.

- *Criterio 2, porcentaje de píxeles interesantes cubiertos en camI*: Permite conocer el porcentaje de píxeles que ocupa el segmento sobre la imagen contabilizando cuántos son y cuántos de ellos tienen el color buscado. Para ello se obtienen las coordenadas (x,y) de proyección sobre la imagen de los dos extremos del individuo-segmento a tratar, después, basándose en la ecuación de la recta 2D $y = p*x+n$, se calculan los valores de la pendiente *(p)* y del valor de intersección con el eje y *(n)* para finalmente recorrer uno a uno los píxeles entre ambos puntos-extremos del segmento (figura 4.13) .

[1]Frustum: Volumen observado por una cámara con forma de pirámide 3D con la punta recortada.

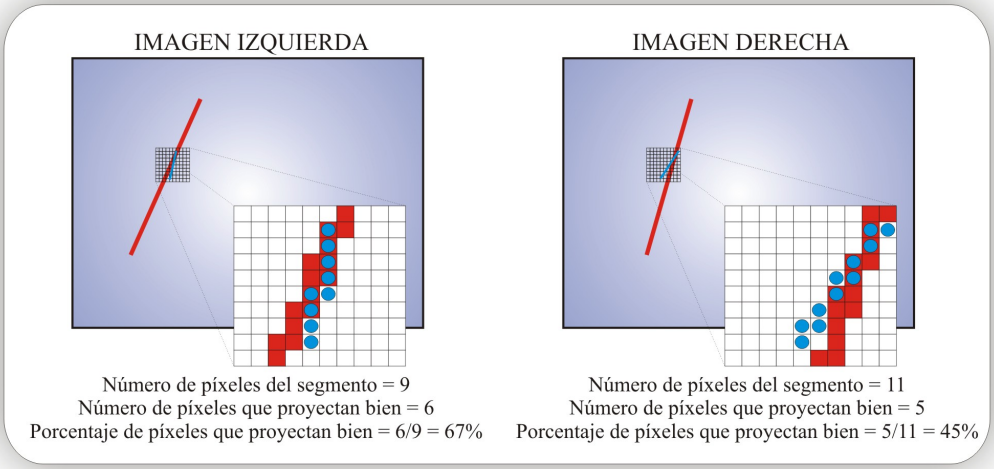

Figura 4.13: Cálculo del porcentaje de píxeles sobre los que proyecta un segmento en imagen.

- *Criterio 3, porcentaje de píxeles interesantes cubiertos en camD*: Igual que el criterio 2 pero utilizando como imagen de origen la ofrecida por la cámara derecha. Este criterio y el anterior se basan en la idea de que los individuos que cubren más píxeles del color relevante están mejor colocados sobre el objeto.

- *Criterio 4, parecido de vecindad en el extremo* P_1: Tal y como se explicó en la sección 4.4.2, este campo analiza para el extremo del segmento P_1 los parecidos entre la ventana de vecindad en torno al píxel de las coordenadas de proyección entre la imagen de la cámara izquierda y la de la derecha. Para ello se observa la zona de vecindad del pixel, mirando en las imágenes filtradas por color previamente generadas. La salud máxima que toma el individuo en este campo es el número de píxeles de color relevante que son iguales en la zona de vecindad de ambas cámaras.

- *Criterio 5, parecido de vecindad en el extremo* P_2: Igual que el criterio 4 pero utilizando el otro extremo del segmento P_2. Estos criterios se basan en la idea de que si las vecindades son similares sobre ambos objetos, el individuo está más próximo al objeto en 3D.

Una vez calculados los valores de cada uno de los campos se procede al cálculo de la salud h_i utilizando la función de salud del comportamiento OR similar a la de las moscas:

$$h_i = \alpha * Criterio_2 + \beta * Criterio_3 + \gamma * Criterio_4 + \delta * Criterio_5$$

4.6. Algoritmo de segmentos 3D variables

El tercer algoritmo genético desarrollado recibe el nombre de *Algoritmo Genético de Segmentos 3D de Longitud Variable* donde cada individuo-segmento varía su longitud creciendo o decreciendo. En este diseño cada segmento es una solución en sí misma en cuanto a cada arista 3D real (enfoque *Pittsburgh*). Al igual que ocurre con segmentos de longitud fija, estos se representan por coordenadas absolutas o relativas, cada una es más adecuada para determinados cómputos.

En esta implementación cada segmento tiene una longitud diferente que podrá variar durante la evolución de las poblaciones. La longitud mínima de estos segmentos se puede parametrizar utilizando la interfaz de la aplicación entre 0 y 300 mm.

Al modificar la longitud de los individuos de la población, estos crecen hasta explicar por sí solos una arista completa de algún objeto 3D. Esta es la principal diferencia entre este algoritmo y los anteriores ya que ahora un único individuo representa una recta 3D a diferencia de los anteriores que para representarla necesitaban varios individuos. Con esta idea surge el *problema multiobjeto* porque los algoritmos genéticos tienden a buscar un único representante. Por este motivo ha sido necesario realizar una implementación más compleja del algoritmo para tratar los problemas multiobjeto (Sección 5.5)

La población de individuos esta dividida en dos subconjuntos:

- *Población de exploradores.* Esta población está compuesta de 1000 individuos-segmento cuya labor es explorar el espacio de aristas 3D con la finalidad de encontrar individuos prometedores, es decir, segmentos cuya salud sea mayor que un umbral dado que se sitúan correctamente sobre algún objeto del espacio.
- *Población de explotadores.* Esta población está compuesta por un número indeterminado de razas que inicialmente será cero. Cada raza es un conjunto de segmentos explotadores, y esta compuesta por 100 individuos cuya labor es explotar únicamente la región concreta del espacio 3D que se haya encontrado compatible con las imágenes. Las nuevas razas de explotadores se crean durante la ejecución del algoritmo; cada vez que aparece un segmento explorador prometedor se crea una nueva raza de explotadores compuesta por individuos iguales a ese explorador prometedor.

El esquema de la aplicación que materializa el diseño del algoritmo de segmentos 3D de longitud variable se denomina *seg3dv* y su diagrama de flujo se muestra en la figura 4.14.

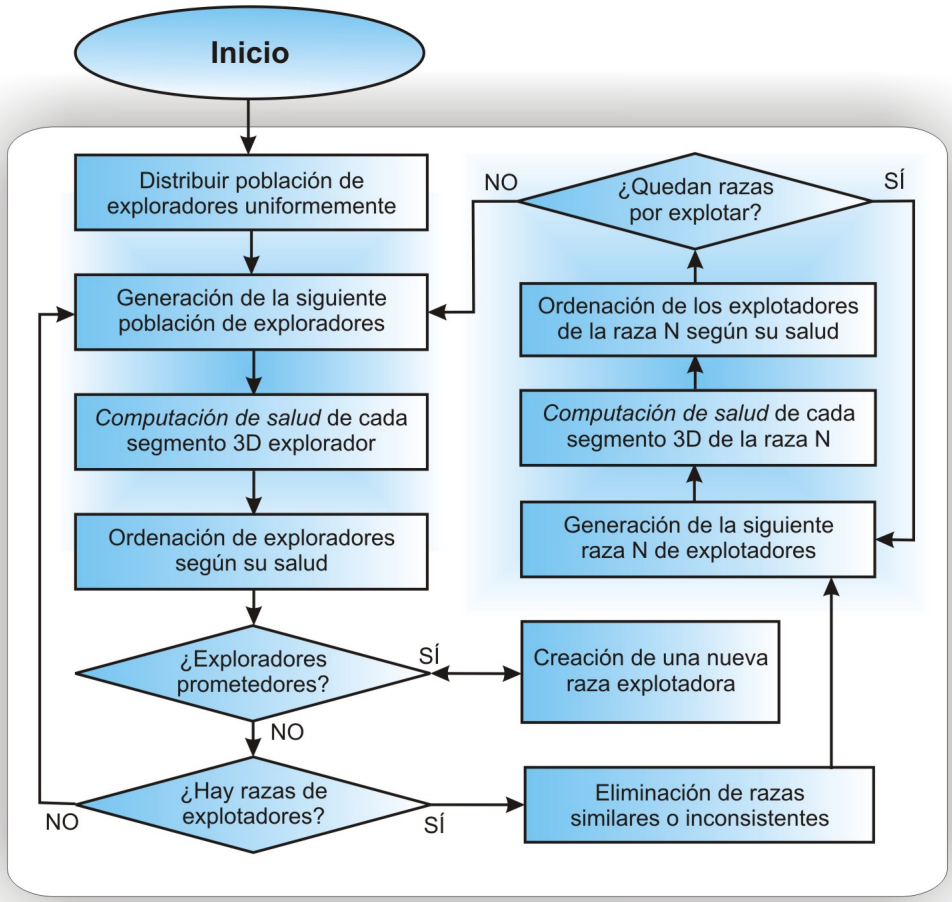

Figura 4.14: Diagrama de flujo del esquema seg3dv.

4.6.1. Generación de la siguiente población de exploradores

La generación de la siguiente población para los *exploradores* se realiza utilizando los operadores genéticos de abducción y mutación.

- *Abducción*. Con este operador se consiguen nuevos segmentos 3D obteniendo dos puntos 3D P_1 y P_2 a lo largo de dos líneas de proyección diferentes y uniendo dichos puntos entre sí. Algunos podrán estar cerca del objeto. En cada iteración, siempre que se detecte algún objeto compatible con el color, se visualiza un haz de segmentos distribuidos a lo largo de diferentes líneas de proyección de cada cámara (figura 4.15). Este operador permite buscar en zonas del espacio de soluciones compatibles con las observaciones sensoriales.

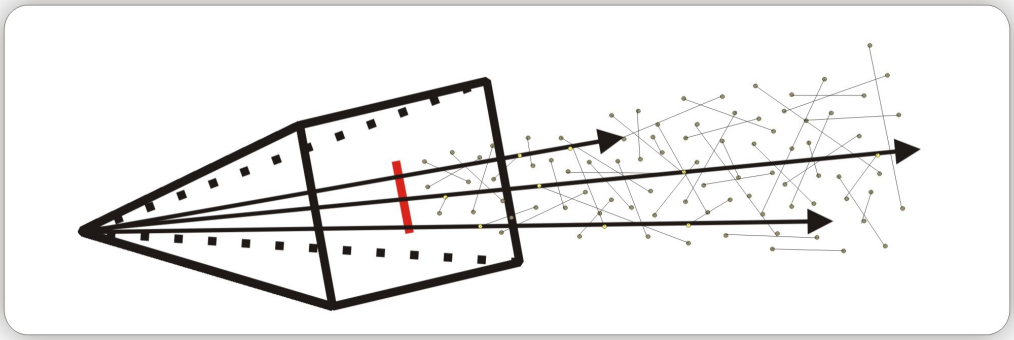

Figura 4.15: Segmentos variables generados a lo largo de líneas de proyección.

- *Mutación aleatoria.* Este operador selecciona aleatoriamente una posición en el espacio 3D y sitúa allí un segmento con una longitud y orientación aleatorias. Este operador permite *explorar uniformemente el espacio* de la habitación.

La mayor o menor influencia de un operador u otro para generar los nuevos segmentos en la población se fija mediante *porcentajes*. Se establece un porcentaje a cada operador (exceptuando la repulsión) de manera que en cada iteración se asigna a cada operador el número de segmentos que debe generar, del total de la población. Estos porcentajes han sido ajustados tras muchos experimentos hasta conseguir una configuración óptima que se detallará en la sección 5.1.

4.6.2. Generación de las siguientes razas de explotadores

La generación de la siguiente población para los *explotadores* se realiza utilizando los operadores genéticos de elitismo, cruce, ruido térmico y repulsión que evolucionan a todos los infividuos de cada raza para conseguir el mejor representante de la misma que es el individuo con mayor salud posible.

- *Elitismo y Repulsión.* Estos operadores funcionan exactamente igual que con segmentos de longitud fija.

- *Cruce.* Este operador toma al azar dos individuos de la población anterior con muy alta salud (segmentos padre) para calcular la recta que une los puntos centrales de ambos segmentos y obtener un punto aleatorio contenido en dicha recta. Este último punto del espacio 3D será el centro del nuevo individuo que además tomará el vector director y la longitud del segmento padre que mayor salud tenga (figura 4.16).

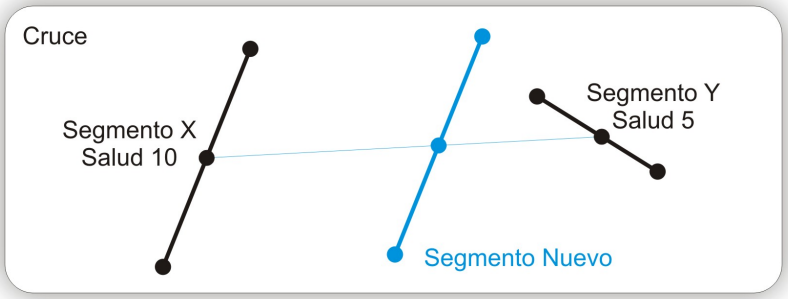

Figura 4.16: Esquema del operador de cruce para segmentos variables.

- *Ruido térmico*. Este operador sitúa un nuevo segmento en una posición aleatoria cercana al punto central de algún otro segmento de la población anterior. Este nuevo segmento toma la misma longitud incrementada o decrementada según un valor aleatorio y una orientación similar a la del segmento original. Permite *explotar una zona* distribuyéndose como un pequeño ruido uniforme en torno a la misma (figura 4.17).

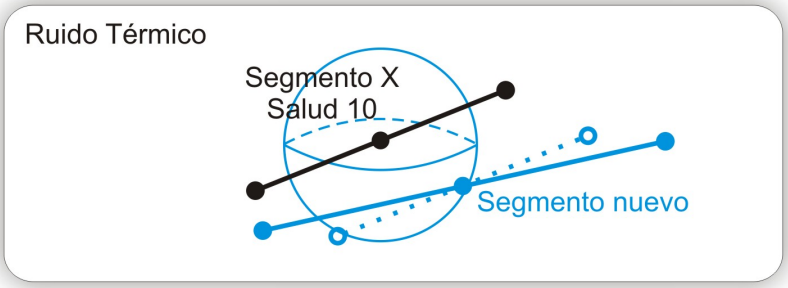

Figura 4.17: Esquema del operador de ruido térmico para segmentos variables.

La aplicación de estos operadores exige tener ordenada por salud la población. La medida en que se utiliza un operador u otro para generar los nuevos segmentos en la población se fija mediante *porcentajes* (exceptuando la repulsión) de manera que en cada iteración se asigna a cada operador el número de segmentos que debe generar, del total de la población.

4.6.3. Computación de salud

Inicialmente se *proyectan los puntos 3D* de los extremos y del centro del segmento s_i sobre el plano imagen de la cámara c_m utilizando la biblioteca Progeo

(sección 3.4). De esta forma conseguimos las coordenadas (x,y) en el plano imagen de los tres puntos más significativos que forman el segmento. Posteriormente se procede a calcular los valores de cada criterio o campo según la siguiente especificación:

- *Criterio 1, dentro del frustum*: Igual que el criterio 1 de segmentos 3D fijos.
- *Criterio 2, porcentaje de píxeles interesantes cubiertos en camI*: Igual que el criterio 2 de segmentos 3D fijos.
- *Criterio 3, porcentaje de píxeles interesantes cubiertos en camD*: Igual que el criterio 3 de segmentos 3D fijos.
- *Criterio 4, número de píxeles interesantes*: Este es el nuevo criterio añadido para este algoritmo en particular, basándose en la idea de que cuanta más longitud tenga un segmento, mejor puntuación o valor de salud debe tener siempre que el segmento proyecte sobre algún objeto relevante. En este sentido, un segmento debe cumplir requisitos de completitud (que abarque todo el objeto) y coherencia (que proyecte realmente sobre un objeto). Este criterio es la suma del número píxeles, de los que cubre el segmento sobre la imagen, que se sitúan sobre un píxel filtrado por color en la cámara izquierda y en la derecha. Con este criterio se satisface el requisito de completitud del individuo, la coherencia la aportan los criterios de porcentaje de píxeles comentados anteriormente.
- *Criterio 5, parecido de vecindad en el extremo* P_1: Igual que el criterio 4 de segmentos 3D fijos.
- *Criterio 6, parecido de vecindad en el extremo* P_2: Igual que el criterio 5 de segmentos 3D fijos.

Una vez calculados los valores de cada uno de los campos se procede al cálculo de la salud h_i utilizando la función de comportamiento OR:

$$h_i = \alpha * Criterio_2 + \beta * Criterio_3 + \gamma * Criterio_4 + \delta * Criterio_5 + \epsilon * Criterio_6$$

4.6.4. Creación de una nueva raza explotadora

Tal y como se expuso en el diagrama de flujo 4.14, tras evolucionar en cada iteración la población de exploradores, alguno de sus individuos puede llegar a tener una

salud que iguala o excede un cierto umbral considerándose entonces que ese individuo es un explorador prometedor. Cuando un individuo es prometedor formará una nueva raza de explotadores o se incorporará en alguna otra raza similar a él.

En concreto, una vez que tenemos el individuo prometedor, se comprueba si existe alguna raza explotadora, si no hay se crea una nueva con este segmento; en caso de que sí haya una o más razas explotadoras comprobamos la similitud entre el explorador prometedor y el individuo representante de cada raza existente de explotadores. La similitud entre dos segmentos $\overline{AB}$ (representante) y $\overline{CD}$ (prometedor) se mide con una función que hemos desarrollado y que combina tres criterios que se deben satisfacer simultáneamente:

- Parecido trasversal. Consiste en comprobar la distancia euclídea a la que se encuentra un segmento de otro. Si esa distancia es muy cercana se puede considerar que los segmentos son similares. Para ello se calcula la distancia que hay entre la recta que forma el segmento $\overline{AB}$ con cada uno de los puntos extremos del otro segmento $\overline{CD}$. Si la *d(AB,C)* y la *d(AB,D)* es inferior a un umbral específico, entonces ambos segmentos se consideran similares (figura 4.18).

$$d(C, \overline{AB}) = \frac{\left|\overline{AB} x \overline{AC}\right|}{d(A, B)} \tag{4.4}$$

- Parecido angular. Consiste en comprobar el ángulo que se forma entre un segmento y otro, en caso de que el ángulo sea muy pequeño los segmentos son similares angularmente. Para calcularlo se obtiene el producto escalar de los vectores directores de los segmentos (figura 4.18).

$$cos(\phi) = \frac{(AB_{xd} * CD_{xd}) + (AB_{yd} * CD_{yd}) + (AB_{zd} * CD_{zd})}{|AB| * |AB|} \tag{4.5}$$

- Parecido longitudinal. Consiste en comprobar si dos segmentos son cercanos longitudinalmente, es decir, si la proyección de uno de los extremos del segmento $\overline{CD}$ proyecta sobre la recta del segmento $\overline{AB}$ dentro del propio segmento $\overline{AB}$. Para ello se calcula el valor de proyección λ de un extremo del segmento $\overline{CD}$ sobre la recta del segmento $\overline{AB}$, obteniendo el valor de intersección entre dicha recta y el plano que pasa por uno de los extremos del segmento $\overline{CD}$ y forma un ángulo de 90 grados con la recta (figura 4.18).

$$\lambda_C = \frac{AB_{xd}(A_x - C_x) + AB_{yd}(A_y - C_y) + AB_{zd}(A_z - C_z)}{AB_{xd}^2 + AB_{yd}^2 + AB_{zd}^2} \tag{4.6}$$

$$\lambda_D = \frac{AB_{xd}\,(A_x - D_x) + AB_{yd}\,(A_y - D_y) + AB_{zd}\,(A_z - D_z)}{AB_{xd}^2 + AB_{yd}^2 + AB_{zd}^2} \tag{4.7}$$

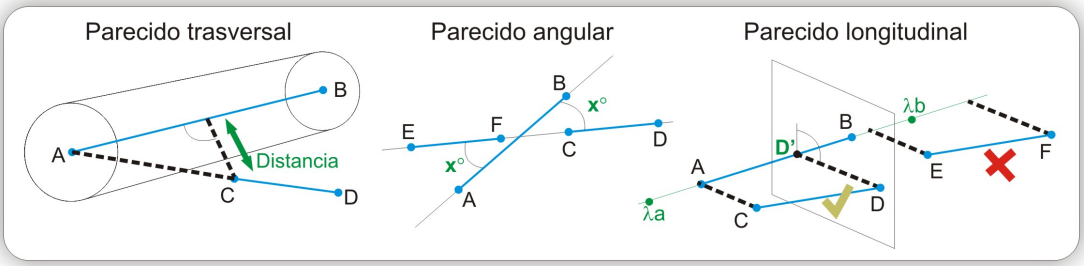

Figura 4.18: Parecido trasversal, angular y longitudinal.

Todos estos cálculos se realizan de manera rápida (sólo involucran sumas, productos y divisiones) para no ralentizar la ejecución del algoritmo evolutivo. Una vez que se ha comprobado la similitud del individuo-segmento prometedor con todos los individuos representantes de cada raza de explotadores, si el segmento es parecido, se añade a la población con la cual tiene parecido en su posición correspondiente según la salud; en caso de que el segmento no se parezca a ninguna raza se crea una nueva. Las razas de explotadores se almacenan en una lista dinámica tipo FIFO (First input, first output).

4.6.5. Eliminación de razas similares o inconsistentes

Esta etapa es la encargada de realizar dos comprobaciones diferentes sobre todas las razas de explotadores.

Por un lado se fusionan las *razas que son similares*. Cuando tenemos más de una raza de segmentos explotadores inicialmente serán diferentes, sin embargo, es posible que trascurrido un tiempo, aparezcan varias razas *similares* entre sí e incluso *iguales*. Por este motivo es necesario comprobar, todas con todas, las razas de explotadores y en caso de encontrar dos similares, eliminar la raza que menor salud tenga. Se considera por salud de una raza, la salud del individuo representante de la misma.

Por otro lado se eliminan las *razas que pierden consistencia*. Se comprueba que la salud de cada raza no baje de un determinado umbral. Es posible que una raza se encuentre explotando un objeto que desaparece instantáneamente. En tal caso, los individuos de esa raza comenzarán a perder salud rápidamente y empezarán a explotar

una zona en la que no hay ningún objeto. Por esto mismo, cuando el representante de una raza de explotadores descienda su salud por debajo de un umbral, la raza es eliminada.

4.7. Interfaz Gráfica de Usuario GUI.

En la figura 4.19 se expone la interfaz de la aplicación y se definen cada uno de sus elementos para modular y parametrizar los algoritmos así como las herramientas de depuración y ajuste de que dispone.

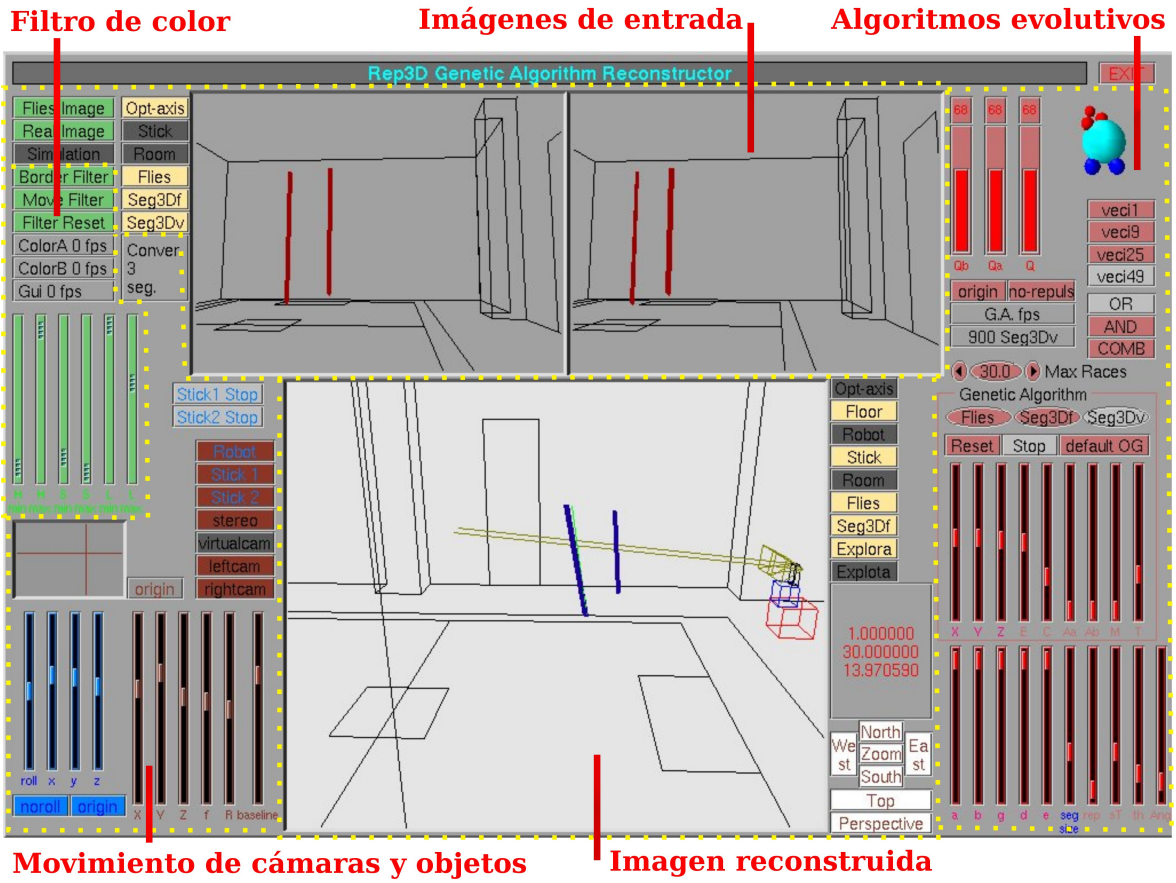

Figura 4.19: Interfaz de usuario de la aplicación reconstructiva Rep3D.

El GUI se divide principalmente en cinco partes bien diferenciadas.

1. *Imágenes de entrada.* Las imágenes de entrada que necesita el algoritmo pueden ser reales, virtuales o reales y virtuales a la vez. En la figura 4.19 se pueden observar los siguientes controles:

- *Cámara derecha.* Visualiza la imagen capturada por la cámara derecha. Si se hace clic sobre algún pixel de esta imagen, se toma el color de ese pixel como valor para realizar un filtrado de color.
- *Cámara izquierda.* Visualiza la imagen capturada por la cámara izquierda. Si se hace clic sobre algún pixel de esta imagen, se toma el color de ese pixel como valor para realizar un filtrado de color.
- *Botones amarillos.* Estos son los botones de visualización, sirven para seleccionar en las cámaras los elementos de la imagen que queremos ver. Estos son: OptAxis (ejes de coordenadas), Stick (objetos palo), Room (habitación virtual), Flies (individuos mosca), Seg3df (individuos segmento fijos) y Seg3dv (individuos segmento variables) pulsando los respectivos botones.
- *Botón Flies Image.* Muestra en las cámaras (derecha e izquierda) la imagen filtrada que están recibiendo y observando los algoritmos evolutivos. En esta imagen solo aparecerá el objeto que se esté tratando sobre un fondo blanco.
- *Botón Real Image.* Muestra en las cámaras (derecha e izquierda) las imágenes reales capturadas por el par estéreo de cámaras.
- *Botón Simulation.* Muestra en las cámaras (derecha e izquierda) las imágenes virtuales generadas por la aplicación.

2. *Imagen reconstruida.* Los objetos reconstruidos por los algoritmos genéticos se podrán visualizar en la aplicación utilizando los siguientes controles (figura 4.19):

   - *Imagen reconstruida.* En esta imagen se pueden visualizar las reconstrucciones conseguidas utilizando los algoritmos genéticos.
   - *Botones amarillos.* Estos son los botones de visualización, sirven para visualizar en la imagen reconstruida los elementos que queremos ver. Los elementos que se pueden activar para ser visualizados son: OptAxis (ejes de coordenadas), Floor (suelo de la habitación), Robot (representación del robot), Stick (objetos palo), Room (habitación virtual), Flies (individuos mosca), Seg3df (individuos segmento fijos), Explora (individuos segmento exploradores) y Explota (individuos segmento explotadores).
   - *Botones blancos.* Estos botones sirven para establecer diferentes puntos de vista predefinidos en la imagen reconstruida. Utilizando estos botones podemos ver la imagen desde diferentes ángulos. Las opciones definidas por defecto son North, South, East, West, Zoom, Top y Perspective.

3. *Filtro de color.* Utilizando los siguientes controles se puede configurar el filtro de color y los filtros opcionales de bordes y movimiento.

   - *Botón Filter Reset.* Para desactivar el filtro de color siempre que haya sido activado previamente haciendo clic en algún pixel de las imágenes de las cámaras derecha o izquierda.
   - *Sliders de configuración de color.* Relevantes para modificar los valores máximos y mínimos de los campos Tinte, Saturación e Intensidad del filtro de color HSI (sección 4.3.1).

4. *Movimiento de cámaras y objetos.* Utilizando los controles de la figura 4.19 se pueden variar la posición y orientación de las cámaras y los objetos de la escena según los siguiente criterios:

   - *Botones Stick1 Stop y Stick2 Stop.* Sirven para activar o desactivar el movimiento automático de los Sticks o palos virtuales visualizados en las cámaras.
   - *Botones marrones.* Sirven para elegir que cámara u objeto se desean mover: el robot, el palo 1, el palo 2, el par estéreo de cámara, la cámara derecha, la cámara izquierda o la cámara virtual.
   - *Sliders marrones.* Permiten modificar los valores de coordenadas de posición X, Y, Z, el foco f, R y la separación de las cámaras Baseline del objeto o cámara que esté seleccionado en los botones marrones.
   - *Sliders azules.* Sirven para variar los valores de orientación X, Y, Z y r del objeto o cámara que este seleccionado en los botones marrones.
   - *Panel de orientación.* Haciendo clic y arrastrando el ratón sobre este panel se puede variar la orientación del objeto o cámara que este seleccionado en los botones marrones.

5. *Algoritmos evolutivos.*

   - *Botones veci1, veci9, veci25 y veci49.* Sirven para seleccionar el tipo de vecindad que se debe aplicar al criterio de vecindad en el cálculo de salud. La vecindad puede ser 1x1, 3x3, 5x5 o 7x7.
   - *Botones OR, AND y COMB.* Estos botones selectores sirven para elegir que modo de combinar criterios queremos utilizar para calcular la salud total de los individuos.

- *Selector Max Races*. Establece un valor máximo de razas de segmentos 3D variables que puede tener la población de explotadores.
- *Botón NoRepuls*. Activa o desactiva el uso del operador genético de repulsión en los algoritmos evolutivos.
- *Barras Quantity*. Estas barras variables muestran en cada momento los porcentajes de píxeles buenos de la imagen A, de la imagen B y de ambas a la vez, que están siendo representados por la población.
- *Botones ovalados*. Sirven para seleccionar el algoritmo genético sobre el cual se quiere hacer modificaciones.
- *Botones Reset, Stop y Default OG*. El botón Reset sirve para iniciar desde el principio la población que esté seleccionada en los botones ovalados. El botón Stop detiene o inicia la evolución del algoritmo seleccionado. El botón Default OG establece los valores por defecto en los sliders genéticos.
- *Sliders genéticos*. Estas barras de desplazamiento sirven para establecer el porcentaje de población que debe someterse a los operadores genéticos E (Elitismo), C (Cruce), Aa (Abducción en a), Ab (Abducción en b), M (Mutación) y T (Ruido térmico). Cuando se varían estos porcentajes se modifican los porcentajes del operador genético del algoritmo evolutivo seleccionado.
- *Sliders para criterios de salud*. Con estas barras de desplazamiento se varían los valores a (alpha), b (beta), g (gamma), d (delta) y e (epsilon) utilizados para calcular la salud total de cada individuo.
- *Sliders segSize, rep, sT, th y Ang*. *segSize* establece el tamaño de los segmentos 3D de longitud fija o el tamaño mínimo de los segmentos 3D de longitud variable. *rep* sirve para establecer la distancia de repulsión entre individuos que se someten a este operador. *st* (sigma thermal noise) establece el radio alrededor del cual se generan los nuevos individuos con este operador. *th* sirve para indicar qué porcentaje o número de individuos se desean representar. *Ang* sirve para elegir el ángulo a partir del cual un segmento es similar a otro.

Capítulo 5
Experimentos

En este capítulo se presentan los experimentos y observaciones realizados a lo largo del proceso de creación de este proyecto incluyendo la ejecución típica y diferentes pruebas con los algoritmos genéticos implementados.

5.1. Ejecución típica

Para comprobar que se cumplen los requisitos impuestos en el capítulo de objetivos (ver 2) se han realizado varios experimentos con cada uno de los algoritmos desarrollados. A continuación se presenta, con los tres algoritmos evolutivos implementados, una prueba utilizando un procesador Pentium 4 mobile 2.8 GHz con HiperThreading, 512 MB de memoria DDR y Tarjeta Gráfica ATI RADEON 9200 64 MB. El escenario virtual utilizado para la prueba muestra dos palos verticales de color rojo sobre ambas imágenes simuladas.

5.1.1. Algoritmo de moscas

En este algoritmo se han realizado varios experimentos variando los porcentajes de los operadores genéticos, los tipos de vecindad y los métodos de cálculo de salud. Los mejores resultados se han conseguido con una población de 2000 moscas, un 30 % de elitismo, un 0 % de cruce, un 60 % de abducción, un 5 % de ruido térmico y un 5 % de mutación; además se utiliza una vecindad de 5x5 y el método OR de cálculo de salud.

En la figura 5.1 podemos observar como las moscas se concentran en las posiciones de los palos verticales. Esta misma prueba se ha realizado sobre palos con diferentes orientaciones y todas ellas han ofrecido resultados muy similares exceptuando el caso del objeto horizontal que comentaremos más adelante.

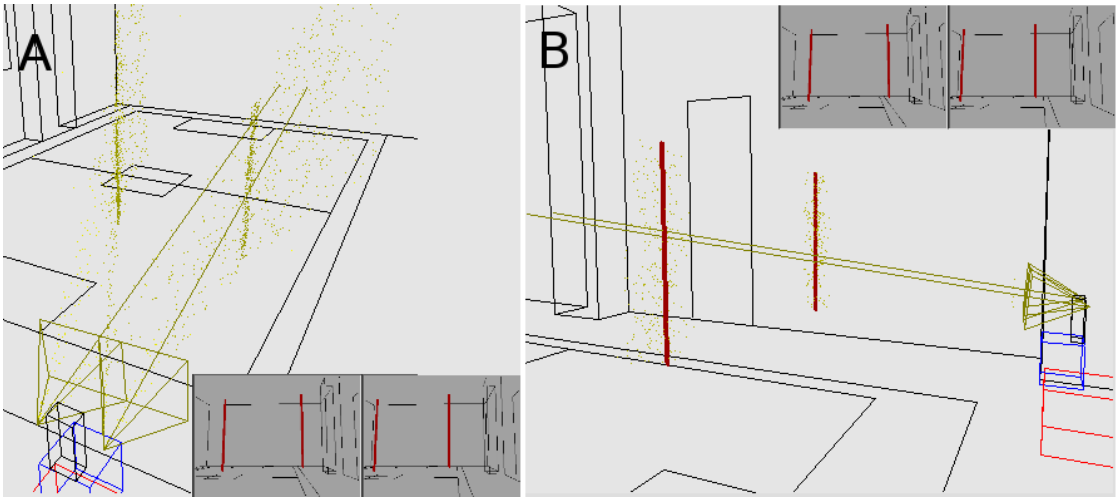

Figura 5.1: Reconstrucción 3D generada con algoritmo de moscas.

La figura 5.1.A muestra el 100 % de la población y podemos observar como el operador de abducción concentra los individuos formando dos haces de moscas para cada objeto. La figura 5.1.B muestra las moscas de mayor salud y la incertidumbre de aproximadamente 5 centímetros en la profundidad del objeto. Esta misma prueba también se ha realizado sobre objetos en movimiento y los resultados cuando se mueven lentamente son muy similares a los que se muestran en la figura pero con un mayor grado de incertidumbre de profundidad.

La velocidad de nuestra implementación en la ejecución típica es de 10 iteraciones por segundo, funcionando con cámaras simuladas ya que se utiliza la tarjeta gráfica (OpenGL) para la reconstrucción. Esta velocidad es suficiente para reconstruir estos objetos en un tiempo de convergencia de 1 segundo. Los individuos que reconstruyen los objetos son un 30 % del total de la población (600 moscas puntuales).

5.1.2. Algoritmo de segmentos 3D fijos

Para este algoritmo se han realizado numerosos experimentos hasta concluir que para obtener resultados similares a los aportados por el algoritmo de moscas se necesita una población de 500 segmentos de longitud fija en lugar de los 2000 puntos; esto se debe a que estas primitivas aportan más información. Los porcentajes de los operadores genéticos son 30 % de elitismo, 0 % de cruce, 60 % de abducción, 5 % de ruido térmico y 5 % de mutación; además de una vecindad de 5x5 y el método OR de cálculo de salud.

En la figura 5.2 podemos observar como los segmentos fijos se concentran en las posiciones de los palos verticales tomando una orientación idéntica a la de los palos reales. Esta misma prueba sobre palos con diferentes orientaciones ofrece resultados muy similares (excepto con objetos horizontales).

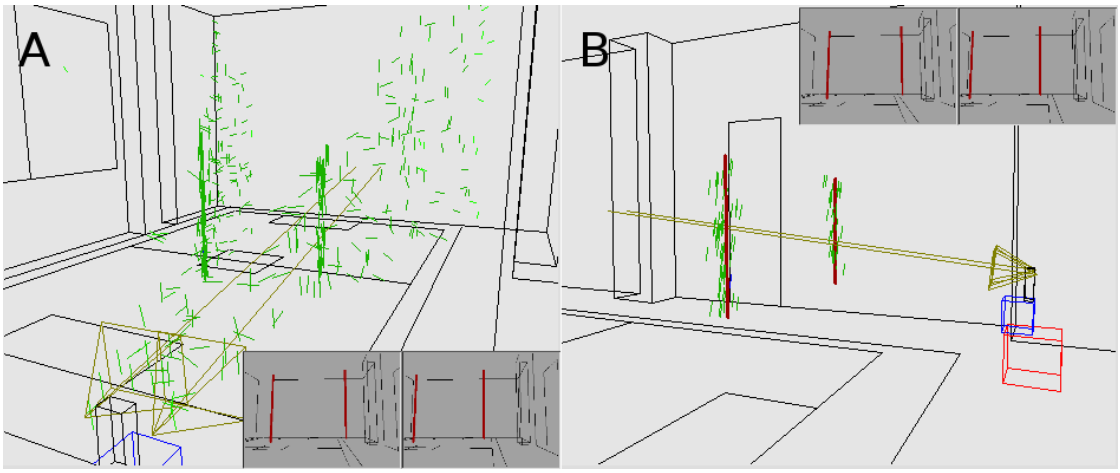

Figura 5.2: Reconstrucción 3D generada con algoritmo de segmentos fijos.

La figura 5.2.A muestra el 100 % de la población y podemos observar como el operador de abducción concentra los individuos formando dos haces de segmentos fijos para cada objeto. La figura 5.2.B muestra los segmentos fijos de mayor salud y la incertidumbre de aproximadamente 5 centímetros en la profundidad del objeto. Sobre objetos en movimiento los resultados, cuando estos se mueven lentamente, son muy similares a los que se muestran en la figura pero con más incertidumbre de profundidad.

La velocidad de nuestra implementación en la ejecución típica es de 12 iteraciones por segundo, funcionando con cámaras simuladas. Esta velocidad es ligeramente inferior a la ofrecida por el algoritmo de moscas, esto se debe a que la primitiva segmento requiere mayor capacidad de cómputo por ser más compleja, sin embargo, como un segmento aporta más cantidad de información no se necesitan tantos individuos como en las moscas para obtener los mismos resultados. Esta velocidad es suficiente para reconstruir estos objetos en un tiempo de convergencia de 1 segundo sin embargo el algoritmo, al igual que en moscas puntuales, no tiene conciencia de cuántos objetos está representando. Los objetos los reconstruyen un 30 % del total de la población (150 individuos).

5.1.3. Algoritmo de segmentos 3D variables

Los experimentos realizados sobre este algoritmo ofrecen los mejores resultados utilizando poblaciones de 1000 segmentos exploradores y 100 segmentos explotadores por cada objeto; como se puede comprobar, el número de individuos inicial es inferior al utilizado por las moscas y bastante superior al de segmentos fijos, esto se debe a que los segmentos variables necesitan más capacidad de cómputo que las otras primitivas pero la calidad de los resultados es bastante superior ya que se consigue representar el objeto palo con un único individuo de la población. Los individuos exploradores poseen un 96 % de abducción y un 4 % de mutación; los individuos explotadores con un 50 % de elitismo, un 24 % de cruce y un 26 % de ruido térmico; además su salud se calcula con vecindad de 5x5 y el método OR de cálculo de salud.

La figura 5.3 muestra como los segmentos variables se concentran en las posiciones de los palos tomando *una orientación y una longitud* idéntica a la de los palos reales con bastante precisión centimétrica. Esta prueba siempre ofrece resultados fiables excepto con objetos horizontales.

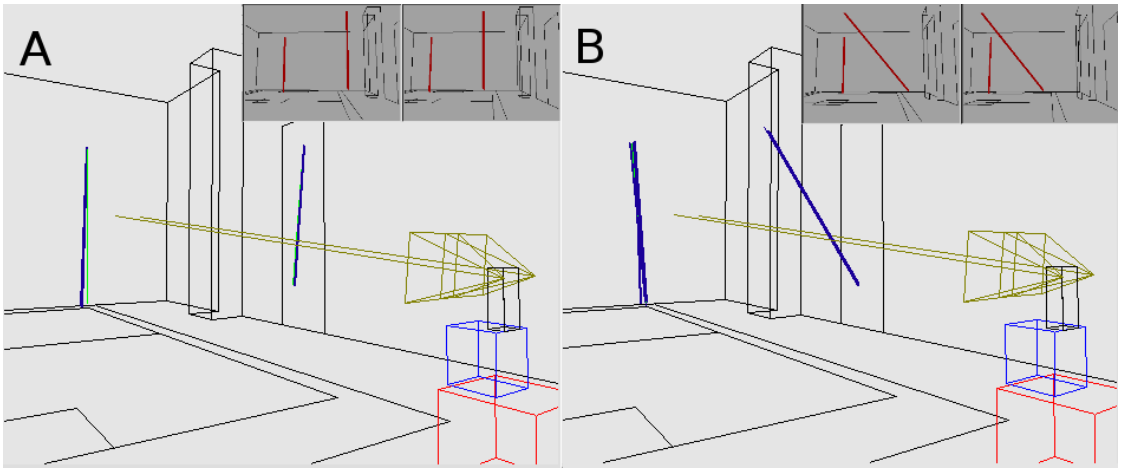

Figura 5.3: Reconstrucción 3D generada con algoritmo de segmentos variables.

La velocidad de la implementación en la ejecución típica es de 8 ips, al igual que con segmentos fijos. Esta velocidad es suficiente para reconstruir estos objetos en un tiempo máximo de convergencia de 5 segundos. Evidentemente el tiempo en esta implementación se dispara bastante en comparación con los otros algoritmos, realmente en 1 segundo, este algoritmo aporta mucha más información que las moscas o segmentos

fijos, es decir, en 1 segundo se obtienen resultados mejores pero en un tiempo estimado de 5 segundos se obtiene el resultado definitivo de localización y reconstrucción del objeto real, hecho que no ocurría nunca con las otras primitivas. Los individuos que reconstruyen el objeto serían tantos como objetos visualicen las cámaras (en este caso 2 individuos segmento, uno por cada objeto).

La tabla 5.4 muestra una comparativa cuantitativa de los resultados obtenidos con cada uno de los algoritmos evolutivos implementados.

	Individuos	Vivacidad	Precisión	Convergencia
Moscas puntuales	2000	10 ips	de 4-5 cm	1 s
Segmentos fijos	500	12 ips	de 4-5 cm	1 s
Segmentos variables	1000 explora 100 x N explota	8 ips	de 1-2 cm	5 s

Figura 5.4: Tabla resumen de algoritmos genéticos.

5.2. Pruebas con operadores genéticos

Se han realizado una serie de pruebas para comprobar la influencia de los operadores genéticos para generar la siguiente población en los algoritmos diseñados. Las pruebas han consistido en probar qué ocurre cuando se varia en mayor o menor medida los porcentajes de los operadores genéticos.

1. La función principal del *elitismo* es mantener a los individuos buenos en su misma posición. Cuando se aplica mucho elitismo gran parte de los individuos detienen su evolución y únicamente algunos continúan variando en las siguientes iteraciones por lo que la convergencia hacia los objetos es muy lenta. Si ponemos una cantidad de elitismo muy baja, no apreciamos evolución en la población ya que, en cada iteración, los individuos de la población anterior simplemente se olvidan y no influyen en la nueva por lo que la convergencia de los individuos se vuelve imposible.

2. La finalidad del *cruce* es obtener individuos en un punto medio entre otros dos representantes de la población anterior. Este operador ofrece resultados diferentes atendiendo al algoritmo en el que se aplica.

   - *Moscas y segmentos fijos.* Con mucho cruce, los individuos tienden a concentrarse en un punto central al azar dentro de algún objeto ya que, si se

cruzan a gran cantidad de individuos, estos terminan atrayéndose entre sí. Cuando el cruce es bajo nos ayuda a situar una mayor cantidad de individuos en posiciones buenas o del objeto. Sin embargo, cuando el cruce es bajo los individuos se mueven aleatoriamente por el interior del objeto y perdemos la percepción correcta del tamaño.

- *Segmentos variables.* La propia definición del operador de cruce nos da la intuición de que este nos ayuda a explotar los segmentos intermedios entre dos individuos. Para este algoritmo es uno de los operadores más relevantes. Con mucho cruce una raza tiende a buscar un punto medio dentro del objeto permitiendo que esta se sitúe más próxima al objeto real. Con poco cruce se pierde la calidad del resultado ya que los individuos de la raza no se atraen y se reparten por los alrededores del objeto.

3. El operador de *abducción* permite explorar el espacio de estados de manera más inteligente que la búsqueda por mutación haciendo más probable estados compatibles con las imágenes. Sirve para situar nuevos individuos en la recta de proyección formada por el foco de la cámara y un píxel característico de la imagen sobre la que se esté abduciendo. Consigue situar individuos de la población en posiciones muy próximas a las cercanías del objeto. Al aumentar la abducción en valores muy altos los individuos se sitúan como un haz sobre las líneas de proyección pero sin situarse de forma correcta sobre el objeto. Cuando se utilizan porcentajes de abducción muy bajos, la población tarda muchísimo en situar individuos sobre los objetos, por eso mismo el operador de abducción es uno de los operadores más importantes para llevar a cabo la reconstrucción de los objetos. Este operador es innovador frente al algoritmo evolutivo propuesto por Louchet [Louchet, 2000].

4. El *ruido térmico* permite explorar las cercanías de un individuo generado en la iteración anterior. Si se aplica mucho ruido térmico, se crean pequeños grupos de individuos, es decir, se organizan por conjuntos diferentes dentro de un objeto. Para el caso concreto de los segmentos de longitud variable, este operador es el que permite que se generen nuevos individuos más largos, más cortos, con diferente orientación y muy cercanos entre sí, esto ayuda a ajustar mejor los individuos al objeto real hasta conseguir el mejor representante para la raza.

5. La utilidad de la *mutación* es situar individuos al azar en el espacio de soluciones 3D para abarcar mayor campo de captura de objetos. Al poner una gran cantidad

de mutación los individuos se expanden en el entorno de la habitación pero no se concentran en ningún objeto ya que la población se genera completamente al azar en cada iteración por no haber un buen porcentaje de elitismo. En el caso de utilizar únicamente elitismo y mutación la convergencia de los individuos sobre los objetos es lentísima pero se conseguiría reconstruir el objeto.

6. Como se ha comentado en anteriores ocasiones la *repulsión* no se aplica sobre un determinado porcentaje de la población sino que se aplica sobre todos los individuos. Su finalidad es comprobar que no haya individuos repetidos en la población. Con una gran distancia de repulsión los individuos se alejan entre sí y resulta difícil concretar la situación real del objeto. Con poca distancia de repulsión la población se llena de individuos iguales o muy similares entre sí dando lugar a información innecesaria.

5.3. Pruebas con combinación de criterios de salud

Los algoritmos creados basan su función de salud en la combinación de criterios como por ejemplo, el individuo proyecta dentro de la imagen o la vecindad proyecta en objeto. Para elegir el método de combinación de criterios que ofrece mejores resultados se han realizado experimentos utilizando diferentes funciones: OR, AND y COMB.

- *Método OR.* Este método se basa en la idea de la disyunción de criterios por lo que estos, multiplicados cada uno por un factor variable desde el GUI, se suman unos con otros para formar el valor de salud del individuo.

$$h_i = \alpha * Criterio_1 + \beta * Criterio_2 + ... + \epsilon * criterio_N$$

  Este método ha resultado ser el que ofrece mejores resultados y ha sido el utilizado por todos los algoritmos. El motivo por el que este método de combinación es el mejor es que permite puntuar a todos los individuos a pesar de que no cumpla alguno de los criterios manteniendo gradualidad en la salud. Además permite multiplicar cada criterio por un factor variable que permitirá dar más valor al criterio que se considere más importante aunque estos valores no se han utilizado en la solución final.

- *Método AND.* Este método se basa en la idea de la conjunción de criterios por lo que estos se multiplican unos con otros para formar el valor de salud del individuo.

$$h_i = Criterio_1 * Criterio_2 * ... * criterio_N$$

Esta combinación de criterios se basa en la idea de que todos los criterios deben puntuar positivamente ya que si alguno de ellos toma el valor cero entonces ningún criterio puntuará y la salud final será 0 igualmente. Experimentalmente se ha comprobado que, para el caso de reconstrucción que se ha implementado en este proyecto, este método ofrece resultados muy similares sobre la reconstrucción final a los resultados conseguidos con el método OR.

- *Método COMB.* Este método se basa en la idea de mezclar el método OR y el AND por lo que algunos criterios se multiplican entre sí, y otros se suman. Además cada uno de los criterios se multiplica por un factor variable desde el GUI al igual que en los métodos anteriores.

$$h_i = Criterio_1 * Criterio_2 * (\alpha * Criterio_3 + ... + \epsilon * Criterio_N)$$

Este método ayuda a que existan algunos criterios que se deben cumplir obligatoriamente (criterios que se multiplican) y otros que se se suman entre sí sin necesidad de tomar valores mayores que 0. Este método ha sido desechado por los mismos motivos del método AND.

5.4. Incertidumbre de profundidad

El uso de dos cámaras como par estéreo da lugar a problemas de incertidumbre de profundidad difíciles de resolver.

Uno de los problemas es la dispersión en profundidad que provoca que los individuos se sitúen en posiciones cercanas al objeto. Para solucionar o aliviar este problema se ha añadido un nuevo criterio de salud llamado *cálculo de vecindad* en sustitución del inicial *cálculo mediante proyección en un punto.*

El cálculo de vecindad (apartado 4.4.2) puede ser 1x1, 3x3, 5x5 y 7x7. Cuanto mayor es el valor de vecindad, menor es la incertidumbre de profundidad y por tanto mejor es el resultado final de la reconstrucción. Con esta técnica podemos comprobar que la incertidumbre de profundidad disminuye considerablemente (figura 5.5).

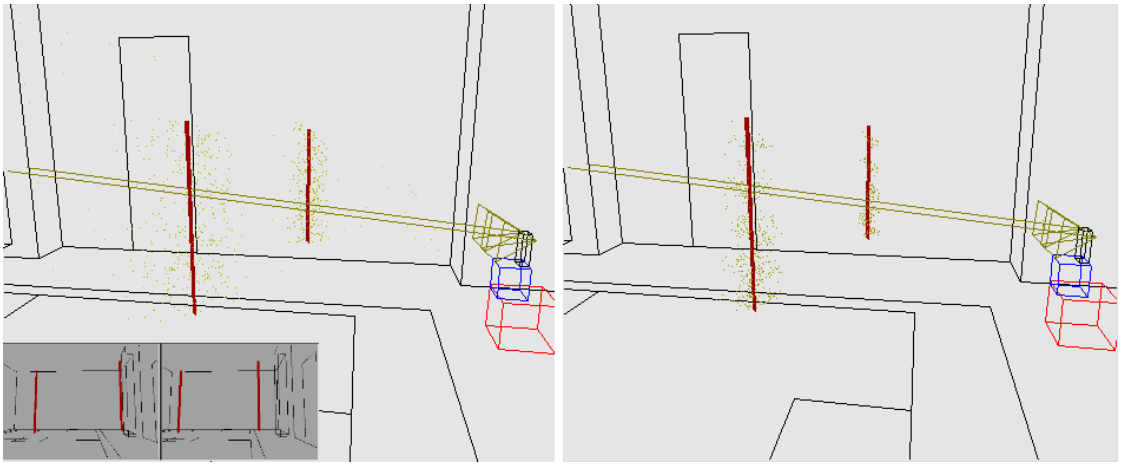

Figura 5.5: Dispersión lateral con vecindad 1x1 (izquierda) y vecindad 7x7 (derecha).

Otra posible solución a este problema consiste en separar las cámaras del par estéreo entre sí. Experimentalmente se llega a la conclusión de que cuanto más separadas están las cámaras, más se concentran los individuos sobre el propio objeto (figura 5.6).

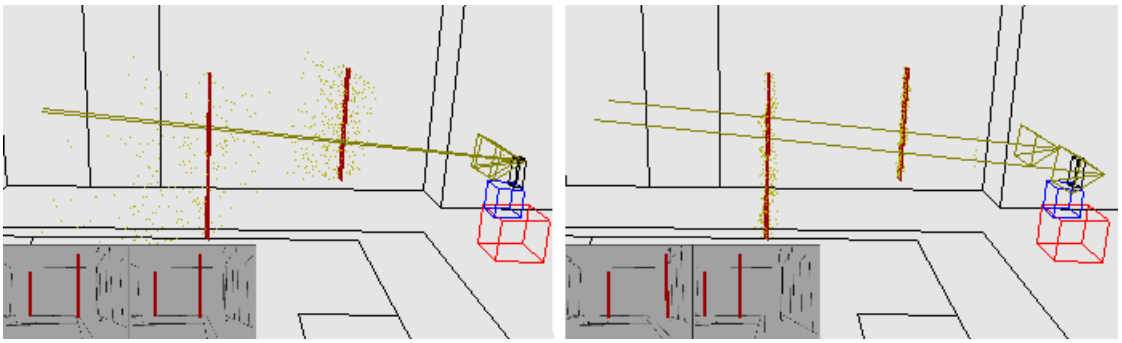

Figura 5.6: Dispersión lateral separando poco (izquierda) o mucho (derecha) las cámaras.

5.4.1. Objetos fantasma

En algunas situaciones nos encontramos con dos objetos similares muy próximos entre sí, como por ejemplo dos palos verticales de tamaño similar uno junto al otro. En estos casos nos aparecen zonas de intriga en las cuales no podemos saber con certeza si hay o no un palo (figura 5.7).

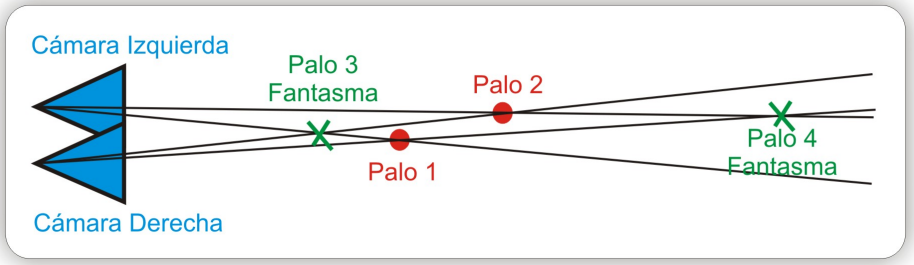

Figura 5.7: Vista cenital del problema de los objetos fantasma.

En la figura 5.8 podemos observar estas zonas delimitadas claramente por las moscas (representadas por puntos amarillos), por los segmentos 3D fijos (representados por pequeños segmentos verdes) y por segmentos variables (representados por segmentos azules). También se puede observar la situación real de los palos de color rojo.

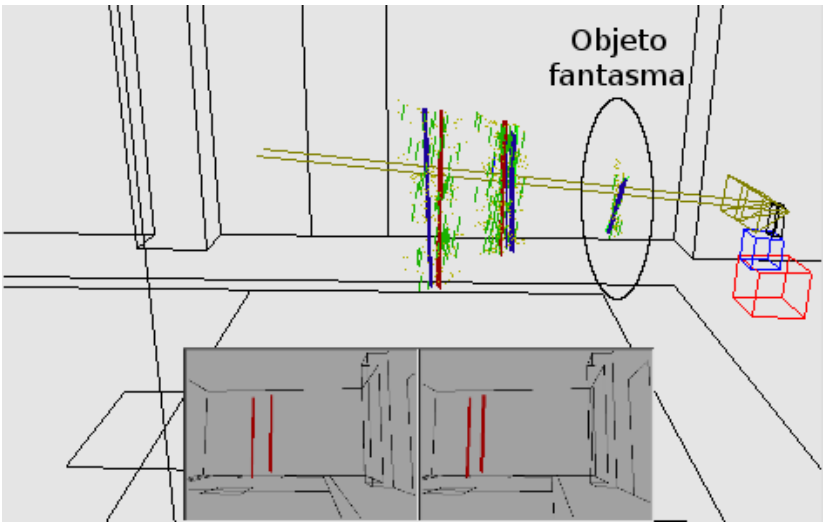

Figura 5.8: Problema de los objetos fantasma.

Este problema es independiente del individuo utilizado por lo que es común a los tres algoritmos implementados. Una posible solución, no implementada, a este problema consiste en mover las cámaras para observar los objetos desde una posición diferente. Otra solución es aplicar otros algoritmos de emparejamiento entre imágenes.

5.4.2. Objetos horizontales

Otro de los problemas surge cuando el objeto es horizontal. Como podemos apreciar en la figura 5.9, el algoritmo no es capaz de deducir cuál es la posición real en la que se encuentra el palo. Esto se debe a que un individuo situado sobre un objeto

horizontal obtiene la misma salud que un individuo que, aunque no esté sobre el objeto, sus coordenadas de proyección sobre la imagen proyectan bien.

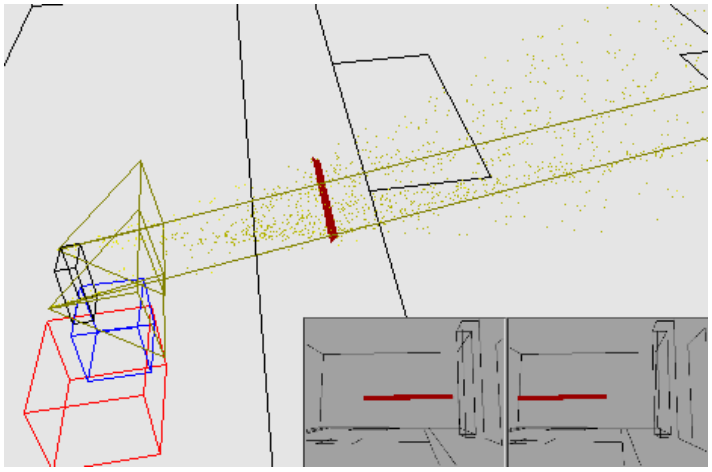

Figura 5.9: Problema del palo horizontal.

Una posible solución a este problema consiste en mover el robot o las cámaras de lugar hasta conseguir que el objeto no se "vea"horizontalmente. Otra posible solución sobre la que solo se han iniciado algunas hipótesis consiste en emparejar los extremos del objeto definiendo un nuevo componente de salud que nos permita saber cual es la posición de cada extremo del palo. Para ello nos ayudaremos de la técnica de detección de bordes laterales (figura 5.10).

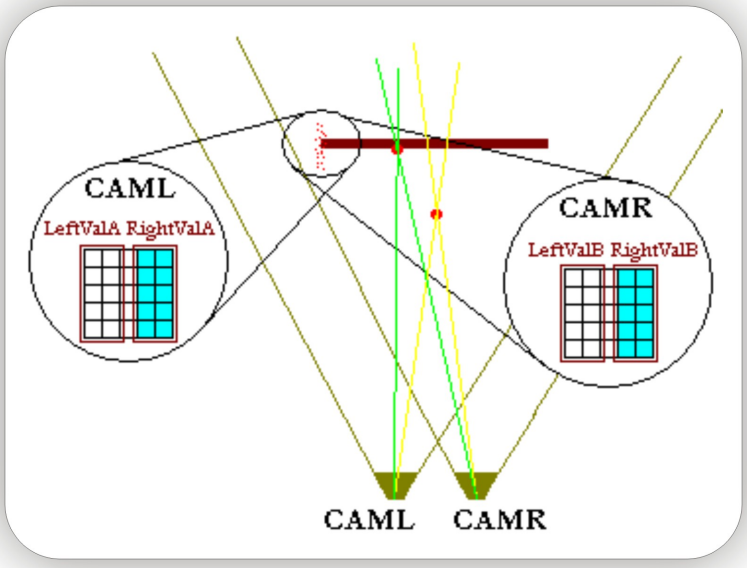

Figura 5.10: Problema del palo horizontal y técnica de bordes laterales.

Esta técnica consiste en detectar puntos sobre el borde derecho y el borde izquierdo del palo, es decir, los dos límites del palo. Para conseguirlo, únicamente hay que comprobar si las proyecciones de vecindad de la mosca se encuentran en la parte derecha o si por el contrario se encuentran en la parte izquierda. Si la mosca está en el borde izquierdo del palo, entonces la vecindad de la mosca tomará valores en su parte derecha y no en la izquierda, y además esto ocurrirá en las dos cámaras. Una vez detectados puntos en los extremos del palo, será necesario generar nuevos segmentos entre un punto de un extremo y otro punto del otro extremo.

5.5. Crecimiento de segmentos y multiobjeto.

Como se descubrió en el capítulo 4, el algoritmo de segmentos de longitud variable es muy diferente de los algoritmos de segmentos de longitud fija y moscas. El algoritmo de segmentos variables incluye nuevas ideas muy diferentes, como por ejemplo, la división de la población en exploradores y explotadores.

Inicialmente se diseñó una estructura con una única población de 2000 individuos que utilizaba los operadores genéticos con los porcentajes de 30 % elitismo, 2 % cruce, 58 % abducción, 5 % ruido térmico y 5 % mutación. El método de cálculo de salud era el OR.

Uno de los experimentos realizados sobre este diseño inicial que muestra el carácter monomodal del algoritmo fue situar en el mundo virtual dos palos verticales de tamaño similar y uno más cercano a la cámara que el otro. Tras lanzar la aplicación se pudo comprobar como la población voluntariamente tendía a centrarse sobre un único objeto según evolucionaba por cada iteración (figura 5.11).

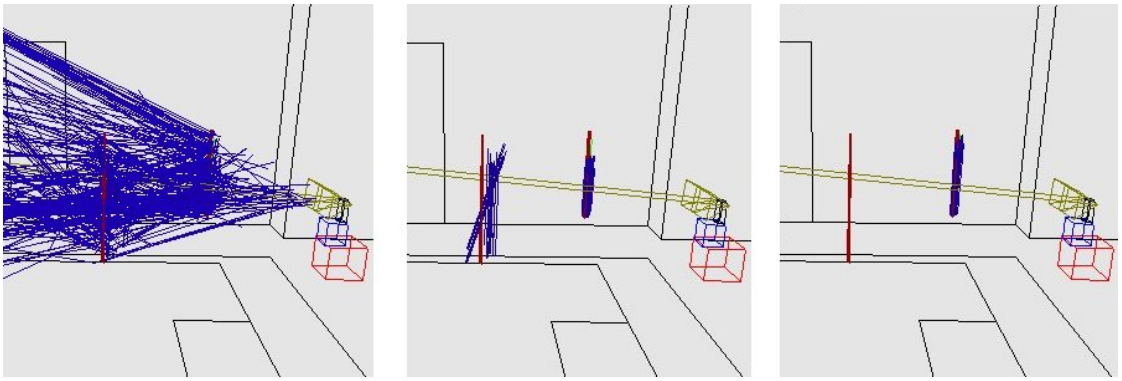

Figura 5.11: Secuencia monomodalidad.

El motivo por el cual sucede este hecho es que el algoritmo genético tiende a buscar el individuo que mejor representa la población. Los operadores de abducción y mutación tienden a buscar múltiples objetos en la imagen, sin embargo, los operadores elitismo y ruido térmico tienden a concentrarse sobre un único objeto aportando al algoritmo un carácter monomodal. El operador de ruido térmico es el encargado de hacer crecer o decrecer a los individuos.

Los criterios de salud permiten puntuar mejor a los segmentos largos que a los pequeños ya que la idea principal de este algoritmo es que el segmento cubra en su totalidad al objeto que se quiere representar. Además, se debe cumplir que toda la superficie del individuo proyecte sobre el objeto, es decir, que no existan puntos intermedios del segmento que proyecten sobre un espacio vacío en el que no hay objeto. Con esta idea se han realizado experimentos con los criterios de *longitud*, *longitud como exponencial*, *porcentaje de píxeles explicados*, *número de píxeles explicados* y *número de píxeles explicados como exponencial*. Se han probado diferentes combinaciones tras las que se han obtenido las siguientes conclusiones:

- *El problema de la longitud.* La longitud es una medida que da lugar a confusión ya que segmentos 3D muy largos que sólo explican parte del palo, puntúan mucho mejor que uno que explique el palo completo pero tenga menor longitud. Experimentalmente esto provoca que los segmentos de gran longitud que son perpendiculares al plano imagen de las cámaras y que pasen por el objeto reciban una salud muy alta a pesar de no ser el individuo adecuado.

- *El problema del número de píxeles.* Un segmento 3D que esté situado sobre un objeto que se represente sobre la cámara con muchos píxeles obtiene mejor salud que un segmento 3D de las mismas características que se encuentre sobre un

objeto representado por menos píxeles en la imagen. Esto se debe a que el número de píxeles que proyectan bien en imagen es mayor para un segmento situado sobre un objeto que en la cámara es más grande, que sobre otros objetos que en las imágenes son más pequeños (no siempre un objeto grande en la cámara tiene que estar obligatoriamente más cerca de la misma, puede estar más lejos pero ser más grande).

Como conclusión final deducimos que ninguna de ellas permite representar más de un objeto en la escena. Además, según los criterios que se utilicen, el resultado final es más o menos preciso. Para obtener unos buenos resultados para representar un solo objeto de la imagen se ha pretendido atender a la completitud y a la coherencia. La *completitud* la ofrece el criterio de *número de píxeles*, ya que premia a los segmentos más largos; la *coherencia* la proporciona el criterio de *porcentaje de píxeles*, ya que este ofrece consistencia sobre el individuo evitando que haya huecos intermedios.

Capítulo 6
Conclusiones y trabajos futuros

En los capítulos anteriores se ha presentado una descripción detallada de la aplicación desarrollada y los experimentos más relevantes realizados con ella. En este último capítulo se resumen las conclusiones obtenidas de estos experimentos y las de todo el proyecto fin de carrera y las posibles vías de continuación para la investigación.

6.1. Conclusiones

El objetivo principal de este proyecto era tratar de *reconstruir en 3D los bordes de los objetos del entorno* desde la información de un par estéreo de cámaras. Para conseguirlo se plantearon diferentes algoritmos evolutivos partiendo del algoritmo genético de *moscas puntuales* ([Louchet, 2000][Pineda, 2006]) que son el algoritmo de *segmentos 3D de longitud fija* y el algoritmo de *segmentos 3D de longitud variable*.

En primer lugar, es necesario destacar que los objetivos planteados se han cumplido con éxito:

1. *Simulador de imágenes*. Este subobjetivo se ha resuelto favorablemente utilizando para la síntesis de imágenes virtuales la biblioteca gráfica Open GL (Sección 4.3). Se ha desarrollado el código que genera las imágenes virtuales de un par estéreo de cámaras que posteriormente utilizan los algoritmos de reconstrucción. En este par de imágenes se puede visualizar el mundo virtual 3D generado en el cual se representan el robot, las cámaras, el laboratorio de robótica, los objetos y los resultados de las reconstrucciones.

2. *Implementación del algoritmo genético de moscas*. Se ha programado un algoritmo evolutivo que utiliza individuos puntuales para llevar a cabo la interpretación del entorno. Este algoritmo se ha inspirado en la idea inicial de Louchet [Louchet, 2000]. Además de su implementación en jde.c hemos añadido

determinadas mejoras propias, como el operador de abducción [Pineda, 2006] que permite obtener mejores individuos de la población y acelerar su convergencia sobre los objetos.

3. *Diseño e implementación del algoritmo genético con segmentos 3D.* Se han diseñado y programado dos variantes, la primera utiliza segmentos 3D de longitud fija como primitivas de la población en lugar de individuos puntuales. Con esta nueva primitiva se consigue que un solo individuo represente más información espacial y así menos individuos en la población pueden reconstruir la misma cantidad de objetos. Además se han mejorado los operadores genéticos y el cálculo de salud para tratar estas nuevas primitivas.
   La segunda variante utiliza segmentos de longitud variable como primitivas que permiten reconstruir un segmento borde del mundo real con un único individuo de la población a diferencia de los algoritmos anteriores. Se han adaptado los operadores genéticos para aumentar o disminuir la longitud. Se ha modificado la estructura inicial de la población separándola en varias subpoblaciones, por un lado los individuos que rastrean todo el espacio visto por las cámaras (exploradores) y por otro lado los individuos que se concentran sobre cada uno de los objetos encontrados (explotadores). Este nuevo algoritmo crea varias razas de explotadores, una por cada objeto de la escena, por lo que ha sido necesario crear listas dinámicas de individuos y comprobaciones de similitud e inconsistencias.

4. *Experimentos.* Los experimentos realizados se comentaron en el capítulo 5 y se han realizado para estudiar el comportamiento, comparar y depurar los algoritmos genéticos desarrollados ante la misma realidad simulada. Estos han permitido afinar la función de salud, comprobar el efecto de los porcentajes de los operadores genéticos, comprobar el comportamiento con objetos dinámicos, detectar algunas situaciones problemáticas y validar experimentalmente la corrección de la reconstrucción y su precisión.

En cuanto al funcionamiento de los algoritmos, las moscas necesitan un gran número de individuos para reconstruir un objeto, sin embargo, su velocidad de convergencia es bastante rápida. El algoritmo de segmentos 3D fijos aporta más información en la reconstrucción de un objeto ya que nos da una idea general de la orientación que puede tomar el mismo, sin embargo, sigue siendo necesario utilizar varios individuos para reconstruir un mismo objeto. Finalmente, el algoritmo de segmentos 3D variables consigue reconstruir fielmente y con gran precisión un objeto con un único individuo de

la población, el problema con el que se encuentra este algoritmo es que es ligeramente más lento. En la tabla comparativa 5.4 vista en el capítulo 5 podemos ver un resumen de los resultados obtenidos con cada uno de los algoritmos.

En resumen, los algoritmos reconstructivos desarrollados en este proyecto dan solución a los requisitos impuestos inicialmente. Estos fueron utilizar el *lenguaje C* sobre la *plataforma software jde.c* utilizando como soporte hardware un *computador convencional*. Todo ello con *elevada vivacidad* y *precisión centimétrica* como se ha visto en el capítulo 5.

6.2. Trabajos futuros

La técnica más potente desarrollada en este proyecto es el algoritmo genético de segmentos 3D variables, en el cual aún se pueden integrar nuevas mejoras. Este proyecto abre las puertas a un gran número de mejoras y aplicaciones ya que la reconstrucción de objetos aporta información muy importante, especialmente para la robótica.

Se pueden incorporar mejoras para resolver las limitaciones del propio algoritmo apreciadas en los experimentos. Uno de los problemas que tiene son los *objetos fantasma*. Una posible solución a este problema podría ser incorporar un algoritmo de emparejamiento entre los objetos de las imágenes izquierda y derecha. Otro problema a destacar del algoritmo es la *intriga horizontal* la cual se puede resolver utilizando cámaras en movimiento o desalineando horizontalmente la posición de ambas cámaras. También se puede explorar el funcionamiento del algoritmo con cámaras reales en lugar de simuladas. Aunque la aplicación permite su uso, no era uno de los requisitos iniciales.

Entre las nuevas aplicaciones que se pueden abordar utilizando el algoritmo de segmentos 3D variables desarrollado en este proyecto fin de carrera, figura la reconstrucción 3D de más objetos adquiriendo la información aportada por el algoritmo para modelizar formas tridimensionales más abstractas que los segmentos, como por ejemplo puertas, sillas, mesas, etc.

Otra posible aplicación podría ser utilizar cámaras en movimiento, cámaras que se puedan mover utilizando cuellos mecánicos, o mundo dinámico, es decir, objetos que se

encuentren en movimiento. Por ejemplo, dado que con este algoritmo somos capaces de reconocer la situación de objetos, se podría desarrollar una aplicación capaz de permitir al robot Pioneer moverse con total libertad sobre el mundo utilizando la información aportada por el algoritmo de segmentos 3D variables.

Bibliografía

[Barrera, 2005] Pablo Barrera. Multicamera 3d tracking using particle filter. *Advanced Tecnology on Imaging and Graphics*, 2005.

[Hartley y Zisserman, 2004] Richard Hartley y Andrew Zisserman. *Multiple View Geometry in Computer Vision.* 2004.

[Jean Louchet y Boumaza, 2002] Marie-Jeanne Lesot Jean Louchet, Maud Guyon y Amine Boumaza. Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern recognition letters*, 2002.

[Louchet, 2000] Jean Louchet. Stereo analysis using individual evolution strategy. 2000.

[Louchet, 2001] Jean Louchet. Using an individual evolution strategy for stereovision. *Genetic Programming and Evolvable Machines*, 2001.

[Mark Segal, 2001] Kurt Akeley Mark Segal. *The OpenGL Graphics System: A specification.* 2001.

[Palacios, 2006] Ricardo Palacios. Representación rica de la escena 3d alrededor de un robot móvil. *Proyecto fin de carrera I.T.I.S., Universidad Rey Juan Carlos*, 2006.

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

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

[Plaza, 2004] José María Cañas Plaza. Manual de programación de robots con jde. *URJC*, 2004.

[T. C. Zhao, 2000] Mark Overmars T. C. Zhao. *xForms Library. A Graphical User Interface Toolkit for X.* 2000.