Sistemas Informáticos 2008 / 2009

Visión estereoscópica

V3D

Desarrollado por:

Conrado Javier García Montiel
Patricia Hernández Llodra
Daniel Merchán García

Dirigido por:

Dr. Gonzalo Pajares Martinsanz
Dpto. Ingeniería del software e inteligencia artificial
Facultad de informática – UCM

Resumen

La visión estereoscópica se basa en la visión binocular (dos ojos) gracias a la cual se produce la sensación de tres dimensiones.

Nuestro proyecto está enfocado en el análisis y obtención de características que poseen las imágenes estéreo. El procesamiento de estas imágenes, permiten que un robot (ó computador) pueda reconstruir el entorno que le rodea tridimensionalmente.

Con dicha reconstrucción del terreno, un robot sería capaz de poder detectar y esquivar los obstáculos que se puede encontrar en su camino, pudiendo así planificar distintas rutas seguras que le lleven hasta su objetivo.

Palabras clave

Visión estéreo, visión por computador, robótica, inteligencia artificial, correspondencia, Mapas Cognitivos Fuzzy, enfriamiento simulado.

Abstract

Stereoscopic vision is based on binocular vision (both eyes) that makes the feeling of three dimensions.

Our project is focused on the analysis and extraction of features that have the stereo images. The processing of these images, allowing a robot (or computer) can reconstruct the environment that surrounds three dimensions.

With the reconstruction of the ground, a robot would be able to detect and dodge obstacles that can be found in its path, being able to plan different secure routes to take to its objective.

Keywords

Stereo vision, computer vision, robotics, artificial intelligence, correspondence, Fuzzy Cognitive Maps, Simulated Annealing.

Autorización

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado: ……. ……. ……..

C. Javier García Patricia Hernández Daniel Merchán

Madrid a 12 de Junio de 2009.

Tabla de contenido

1. Introducción

1.1 Motivación del proyecto

La visión estéreo es un campo muy importante en el mundo de la robótica. Es fundamental para los robots que son enviados a planetas cercanos, como Marte, poder moverse por el terreno de los mismos sin peligro a quedar atrapados entre rocas o cualquier obstáculo que puedan encontrarse. Están muy limitados en cuanto a energía dado que poseen baterías que son recargadas mediante energía solar.

A causa de las limitaciones de los robots, las imágenes estéreo que captan son de poca calidad, pero de ellas hay que extraer la mayor información posible.

Con este aliciente, hemos creado una aplicación JAVA que implementa algoritmos de tratamiento de imágenes estéreo, algunos incluso propuestos por la NASA, con el fin de poder sacar características importantes que ayuden al robot a decidir.

Este trabajo se plantea al amparo de dos proyectos, que el grupo de investigación Ingeniería de Sistemas, Control, Automatización y Robótica (ISCAR) tienen firmados en la empresa TCP Sistemas e Ingeniería S.L., relativos a la dotación de autonomía a los vehículos de exploración planetaria dentro del proyecto ExoMars en el que dicha empresa colabora con EADS Astrium y el CNES francés dentro del programa de la Agencia Espacial Europea.

1.2 Objetivos

El objetivo de este proyecto consiste en el desarrollo de una aplicación, que permite mediante un interfaz amigable la obtención de un mapa de disparidad, como paso previo a la representación tridimensional y de ahí a la planificación de trayectorias que se plantea como reto de futuro.

1.3 Organización de la memoria

Esta memoria se organiza de la siguiente manera. En el capítulo dos se realiza una visión panorámica y análisis de las diferentes perspectivas involucradas en los temas relacionados con la visión estereoscópica, proporcionando los aspectos básicos implementados en el proyecto. En el capítulo tres se presenta la arquitectura de diseño, a la vez que el diseño detallado, sobre el que se basa la aplicación. En el capítulo cuatro se presentan los resultados obtenidos, basados en las imágenes utilizadas. En el capítulo cinco se presentan las conclusiones y trabajo futuro dentro de los proyectos mencionados con la empresa TCP.

2. Análisis

2.1 Algoritmos implementados

En este capítulo, en primer lugar realizamos un recorrido por diversos aspectos visuales relativos a la visión estereoscópica como es la formación de imágenes mediante anaglifos. En segundo lugar, analizamos la geometría del sistema, que permitirá obtener una medida de lo que denominaremos disparidad como paso previo a la obtención de la tercera dimensión. Seguidamente, se estudian los métodos de correspondencia estereoscópica habituales, como son los basados en área y las características. Tras la correspondencia, se obtiene un mapa de disparidad, que es mejorado posteriormente mediante la técnica de optimización basada en el conocido método de enfriamiento simulado. Ambos procesos, correspondencia y enfriamiento simulado constituyen los módulos fundamentales del trabajo que se presenta.

2.1.1 Anaglifo

Las imágenes de anaglifo o anaglifos son imágenes de dos dimensiones capaces de provocar un efecto tridimensional, cuando se ven con lentes especiales (lentes de color diferente para cada ojo). Estas imágenes se componen de dos capas de color, superpuestas pero movidas ligeramente una respecto a la otra para producir el efecto de profundidad.

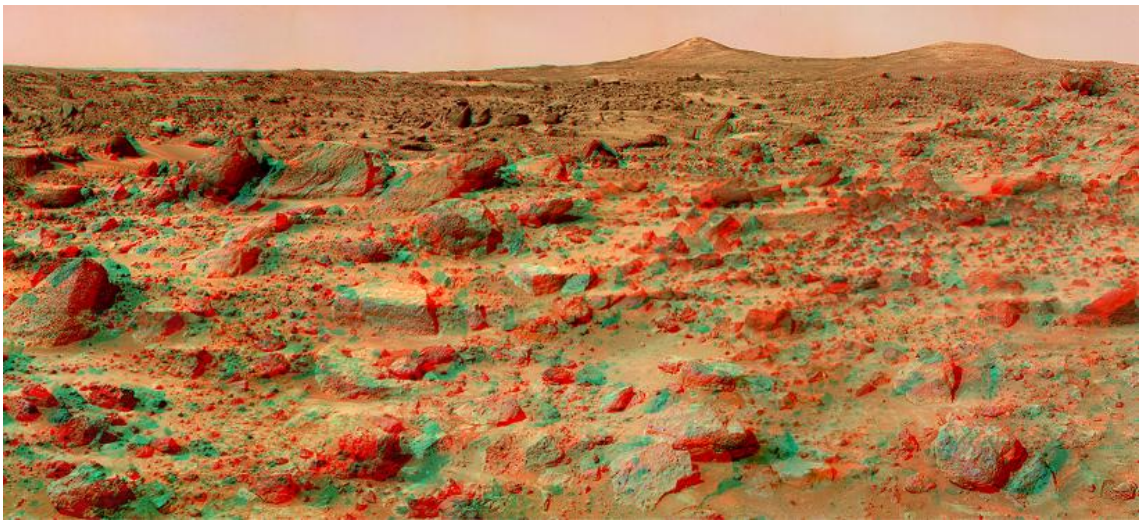

Figura 2.1. Anaglifo de Marte.

Las gafas anaglifo (con filtros de papel de distinto color para cada ojo) son capaces de transmitir la sensación de tridimensionalidad. Los filtros permiten separar el anaglifo en las dos imágenes a partir de las cuales se creó, y hace llegar cada una de ellas a cada ojo. De esta manera conseguimos que cada ojo vea la misma escena pero desde posiciones ligeramente distintas, como lo haría al observar cualquier entorno 3D, dando así sensación de profundidad.

Un anaglifo resulta de la unión del canal rojo de una de las imágenes del par estéreo con los canales verde y azul de la otra. Por lo tanto, si la imagen izquierda es $I_I(R_I, G_I, B_I)$ y la imagen derecha $I_D(R_D, G_D, B_D)$, el anaglifo resultante es $I_A(R_I, G_D, B_D)$, donde R es el canal de rojo, G es el canal de verde y B es el canal de azul.

Para una mejor compresión, en las figuras 2.2 a 2.4 se describe el proceso de generación de las imágenes anaglifo.

Figura 2.2. Par de imágenes estéreo.

Una vez obtenido un par de imágenes estéreo separamos los canales de color, como hemos explicado anteriormente.

Figura 2.3. Separación de los canales.

Por último reconstruimos la imagen para formar el anaglifo.

Figura 2.4. Reconstrucción de la imagen.

Otra forma de conseguir sensación de profundidad comúnmente usada en proyecciones de cine o video tridimensional es el uso de polarizadores sobre una pantalla metálica. La visualización se realiza por medio de gafas dotadas de polarizadores que eliminan la imagen correspondiente al ojo contrario (mediante el mismo principio que los filtros coloreados, anteriormente mencionados). La diferencia con los anaglifos es que estas imágenes son a todo color.

2.1.2 Correspondencia

La correspondencia estéreo es el proceso mediante el cual dado un punto cualquiera de la escena 3D se llega a determinar cuál es su proyección en sendas imágenes del par estereoscópico.

La correspondencia constituye el principal problema dentro del proceso de la visión estereoscópica, siendo éste el aspecto tratado de forma exhaustiva en el presente trabajo.

Para resolver este problema, hay que determinar qué parejas de puntos de ambas imágenes (hablando siempre de visión bifocal) se corresponden con un mismo punto de la escena. De una manera más formal, el problema de la correspondencia consiste en identificar una característica en una imagen, por ejemplo un punto de borde o región y determinar qué característica en la otra imagen es la que corresponde a la misma característica en el espacio tridimensional. Se trata de un problema mal condicionado, pues geométricamente pueden existir infinitas soluciones o que no exista solución (oclusiones), lo que puede dar lugar a ilusiones ópticas (falsas correspondencias).

La dificultad de establecer la correspondencia viene determinada principalmente por el propio sistema, es decir dada la imagen derecha e izquierda de un sistema de visión estéreo hay que tener en cuenta que estas imágenes vienen afectadas por distintos ángulos de vista de la cámara e incluso diferentes iluminaciones.

Para la resolución de éste problema, hemos abordado en nuestro proyecto las dos técnicas que existen a la hora de tratar el problema de la correspondencia: las basadas en el área y las basadas en las características, tal y como se identifican en la literatura especializada.

2.1.2.1 La profundidad como función de la disparidad

El sistema estereoscópico debe poder percibir la distancia que lo separa del objeto en el espacio observado, la medición se produce por medio de triangulación, midiendo el paralelismo entre dos imágenes obtenidas tomando la imagen del objeto desde dos posiciones diferentes, separados una distancia suficiente, conocida como línea base.

A partir de la figura 2.5 considérese que las cámaras están posicionadas sobre el eje X en el sistema de referencia, con los planos de imagen sobre el plano XY. Las cámaras apuntando en dirección del eje Z, de modo tal que sus ejes ópticos sean paralelos y separados una distancia *d*.

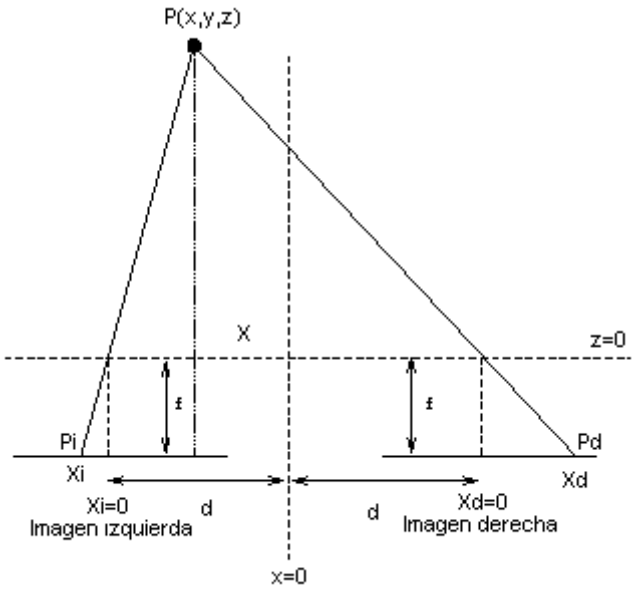

Figura 2.5. Geometría de un sistema de visión estereoscópica.

Mediante triangulación se puede obtener la posición de un punto sobre un objeto (punto observado) determinando el triángulo formado entre el punto observado **P** y los centros ópticos del sistema de observación $\boldsymbol{X_i}$ y $\boldsymbol{X_d}$. La información de profundidad del punto **P** está codificada en la diferencia de posición (disparidad) en los dos planos de imagen. En la ecuación (1) se muestran las expresiones para obtener las mencionadas posiciones y finalmente la coordenada **Z** que se está buscando.

$$X_i = \frac{-(d+x)f}{z} \qquad X_d = \frac{(d-x)f}{z} \qquad z = \frac{2df}{X_d - X_i} \quad (1)$$

2.1.2.2 Correspondencia basada en el área

De lo que se trata es de obtener la diferencia $X_d - X_i$, conocida como disparidad, ya que tanto f como d son conocidas. En efecto, f la proporciona el fabricante de las cámaras y d es la separación de las cámaras impuesta por el usuario. Para calcular tanto X_d como X_i se aplican las técnicas de correspondencia que se describen a continuación.

Para cada píxel de una imagen se calcula la correlación entre la distribución de disparidad de una ventana centrada en dicho píxel y una ventana del mismo tamaño centrada en el píxel a analizar de la otra imagen. El problema consiste en encontrar el punto que se más se parece al primero, es decir el más similar.

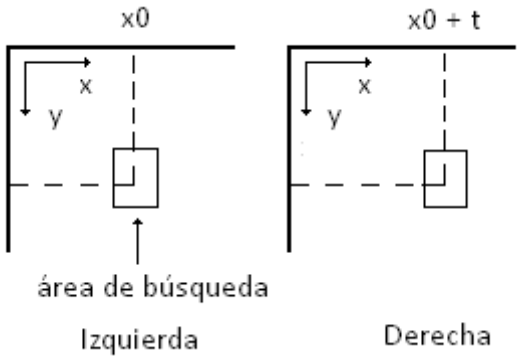

Figura 2.6. Área de búsqueda.

En nuestra implementación nos basamos en las siguientes restricciones:

- La **epipolar**: relacionada directamente con la posición de las cámaras. Si éstas están alineadas, el problema de la búsqueda del punto en la otra imagen puede reducirse a una sola dimensión, en nuestro caso la X puesto que las cámaras están separadas horizontalmente.

- La **semejanza**: los puntos (x, y) de las imágenes estéreo que forman la imagen 3D original deben tener atributos (propiedades) similares.

La restricción epipolar puede ser demasiado restrictiva, por lo que nos centramos principalmente en la de semejanza. Por ello buscamos el punto (x, y) de una de las imágenes en el mismo punto de la otra imagen y sus vecinos. Estos vecinos pueden ser cercanos o lejanos según lo especifiquemos, dado que la cámara puede haber tomado la otra imagen con un ángulo y desplazamiento considerablemente distinto respecto a la primera.

En cada punto (x, y) de la imagen (derecha / izquierda) situados sobre la línea epipolar se define una ventana de tamaño MxN. Finalmente obtenemos el coeficiente de correlación entre estos puntos mediante la siguiente expresión:

$$C = \frac{\sigma_{ID}^2}{\sqrt{\sigma_I^2 \sigma_D^2}} \quad (2)$$

Donde I y D se refieren a las imágenes izquierda y derecha respectivamente, $\sigma_I^2 \sigma_D^2$ representan la varianza de los niveles de intensidad en las correspondientes ventanas y σ_{ID}^2 es la covarianza de los niveles de intensidad entre las ventanas izquierda y derecha.

2.1.2.3 Correspondencia basada en las características

Como contraposición al hecho de establecer correspondencias punto a punto, surge el uso de características. Se entiende por característica alguna estructura significativa de la imagen, como por ejemplo, los puntos de borde.

Es conocido que las técnicas basadas en área son sensitivas a las discontinuidades de profundidad, debido a que la región alrededor de una discontinuidad contiene puntos de más de una profundidad. Estos métodos también son sensibles a las regiones que presentan texturas uniformes. Así, los métodos basados en características buscan sobrepasar estos problemas limitando la región de estudio alrededor de posibles características de una imagen (por ejemplo, bordes, curvas, etc.). Esto también limita la densidad de los puntos para los que la profundidad puede ser estimada.

En nuestro proyecto, hemos implementado la extracción de bordes mediante el uso de operadores de primera derivada como puede ser el gradiente. Por otro lado, para la extracción de esquinas como otra característica de la escena hemos implementado el método de *Harris*.

2.1.2.3.1 Gradiente

El gradiente de una imagen $f(x,y)$ en un punto (x,y) puede definirse del siguiente modo:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} \quad (3)$$

Donde G apunta en la dirección de variación máxima de f en el punto (x,y) por unidad de distancia con la magnitud y dirección dadas por:

$$|G| = \sqrt{G_x^2 + G_y^2} \qquad \phi(x,y) = \tan^{-1} G_y/G_x \quad (4)$$

Tras la aplicación del operador obtenemos una imagen G según la siguiente expresión:

$$g(x,y) = \begin{cases} 1, & G[f(x,y)] > T \\ x, & G[f(x,y)] \leq T \end{cases} \quad (5)$$

Donde T es el valor umbral no negativo que nos indica qué puntos de la imagen consideraremos como borde o no.

En nuestro proyecto hemos utilizado los *Operadores de Sobel* para calcular G_x y G_y. Estos operadores, se materializan según las siguientes máscaras:

$$\text{a)} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{b)} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (6)$$

Donde a) es la máscara utilizada para la obtención de G_x y b) la máscara para la obtención de G_y.

Para aclarar el concepto de obtención de la imagen binaria, supongamos que partimos de G dada por (7).

$$G = \begin{bmatrix} 2 & 1 & 3 & 5 & 2 & 2 \\ 1 & 7 & 8 & 7 & 3 & 1 \\ 0 & 9 & 2 & 7 & 9 & 1 \\ 0 & 9 & 1 & 4 & 8 & 3 \\ 1 & 8 & 2 & 1 & 8 & 0 \\ 1 & 9 & 8 & 7 & 9 & 0 \end{bmatrix} \quad (7)$$

Si utilizamos como umbral $T = 6$, entonces obtenemos la siguiente imagen binarizada con los bordes representados por unos.

$$G_b = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (8)$$

Como podemos observar, la elección del umbral es muy importante a la hora de decidir si un punto (x, y) es considerado borde o no.

2.1.2.3.2 Operador de Harris: extracción de puntos de interés

La extracción de puntos de interés de una imagen es primordial. El operador de Harris extrae de manera muy eficiente las esquinas de los objetos en imágenes, que nos permitirán extraer puntos característicos en las imágenes estéreo.

El método habitual para la extracción de esquinas ha sido el uso de derivadas de segundo orden para medir la razón de cambio de la dirección del gradiente con la magnitud del gradiente. Sin embargo, nuestra aplicación está centrada en la detección de puntos de interés buscando máximas variaciones de curvatura. Por ello nos hemos centrado en la siguiente forma de detección de esquinas:

$$\frac{f_{xx}f_y^2 + f_{yy}f_x^2 - 2f_{xy}f_xf_y}{\left(f_x^2 + f_y^2\right)^{\frac{3}{2}}} \geq U_l \quad (9)$$

Donde U_l es el umbral que sirve para determinar si un punto de la imagen lo consideramos esquina o no. Las funciones f_x , f_y son primeras derivadas en la dirección $x\ e\ y$ respectivamente para la función de intensidad de la imagen $f(x,y)$ en el píxel de la imagen (x,y), mientras que $f_{xx}\ f_{yy} y\ f_{xy}$ son segundas derivadas. Se calculan aplicando las siguientes máscaras:

$$\frac{\partial}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{f}{\partial y} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (10)$$

Por lo tanto podemos definir las derivadas del siguiente modo:

$$f_x \equiv \frac{\partial f}{\partial x}, f_y \equiv \frac{\partial f}{\partial y}, f_{xx} \equiv \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right), f_{yy} \equiv \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right), f_{xy} \equiv \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) \quad (11)$$

Sin embargo, en este trabajo para la detección de esquinas hemos seguido el algoritmo de *Plessey* (*Harris, 1987*). En este algoritmo calculamos las derivadas anteriormente descritas y creamos la siguiente matriz:

$$A = \begin{bmatrix} < f_x^2 > & < f_x f_y > \\ < f_x f_y > & < f_y^2 > \end{bmatrix} = \begin{bmatrix} \sum_{nxn} f_x^2 & \sum_{nxn} f_x f_y \\ \sum_{nxn} f_x f_y & \sum_{nxn} f_y^2 \end{bmatrix} \quad (12)$$

Donde nxn es el tamaño de la ventana centrada alrededor del punto que se está tratando de identificar como punto de interés o no.

A partir de la matriz anterior aplicamos el conocido operador de *Harris y Stephens (1988)* que viene dado por la siguiente ecuación:

$$H = \det(A) - kTr(A) \geq U_m \quad (13)$$

Donde k es una constante que suele tomar valores pequeños ($0 \leq k \leq 1$) y un umbral U_m que suele tomar valores muy altos para obviar en la medida de lo posible falsas esquinas, es decir puntos de interés superfluos.

2.1.3 Enfriamiento simulado: Mejora del mapa de disparidad

Simulated annealing (SA) o enfriamiento simulado es un algoritmo de búsqueda meta-heurística para problemas de optimización global. El objetivo es encontrar una buena aproximación al óptimo global de una función en un espacio de búsqueda grande.

El nombre e inspiración viene del proceso de recocido del acero, una técnica que consiste en calentar y luego enfriar controladamente un material para aumentar el tamaño de sus cristales y reducir sus defectos. El calor causa que los átomos abandonen sus posiciones iniciales (un mínimo local de energía) y se muevan aleatoriamente. El enfriamiento lento les da mayores probabilidades de encontrar configuraciones con menor energía que la inicial.

El proceso, aplicado a nuestro problema se sintetiza en los pasos que se describen a continuación.

2.1.3.1Iteración

Se parte del mapa de disparidad obtenido mediante el proceso de correspondencia inicial, de forma que cada píxel tiene asignado un valor de disparidad s, este valor es lo que denominamos estado del píxel. El proceso de enfriamiento simulado trata de modificar estos estados con el objetivo de lograr valores mínimos de energía, es decir configuraciones estables.

Los vecinos de cada estado se eligen dependiendo del problema específico. Usualmente es una pequeña variación en la representación escogida. Nuestra implementación tiene en cuenta únicamente los vecinos más próximos, es decir, los que están a distancia 1 (contando la diagonal como un único paso).

Es necesario establecer un enlace r_{lk} entre el píxel l, que se está tratando de modificar su estado y un vecino k, este enlace viene dado por la ecuación siguiente.

$$r_{lk} = \begin{cases} 1 - |S_l - S_k| & si\ k \in N_l^8 \\ 0 & si\ k \notin N_l^8 \end{cases} \quad (14)$$

2.1.3.2 Probabilidad de transición

La probabilidad de hacer la transición al nuevo estado s' es una función $P(\delta, E, T)$ de la diferencia de energía $\delta E = E(s') - E(s)$ entre los dos estados, y de la variable T, llamada temperatura.

Una cualidad importante del método SA es que la probabilidad de transición P es siempre distinta de cero, aún cuando δE sea positivo, es decir, el sistema puede pasar a un estado de mayor energía (peor solución) que el estado actual. Esta cualidad impide que el sistema se quede atrapado en un óptimo local.

Cuando la temperatura tiende al mínimo, la probabilidad tiende a cero asintóticamente. Así, cada vez el algoritmo acepta menos movimientos que aumenten la energía.

Si δE es negativo, es decir, la transición disminuye la energía, el movimiento es aceptado con probabilidad $P = 1$.

No obstante en este trabajo se emplea la versión determinista en lugar de la originalmente establecida como probabilista.

2.1.3.3 Velocidad de enfriamiento

Otra cualidad del enfriamiento simulado es que la temperatura debe disminuir gradualmente conforme avanza la simulación. Hay muchas maneras de disminuir la temperatura, siendo la más usual la exponencial, dónde la temperatura T disminuye por un factor $\alpha < 1$ en cada paso.

En nuestra implementación probamos distintas velocidades de enfriamiento. Tras varias sesiones de prueba decidimos utilizar la velocidad de enfriamiento que mostramos a continuación:

$$T(t) = {}^{T_0}/_{\log(t+1)} \quad (15)$$

2.1.3.4 Función de actualización

Una vez definidos los conceptos anteriores nos queda el paso más importante: definir una función de actualización para la disparidad de cada píxel en función de su propio estado y el de sus vecinos más próximos.

Existen diferentes funciones, habiendo optado en este trabajo por la siguiente.

$$s_l(t+1) = \frac{1}{2}\left[f\left(l_l(t), T(t)\right) + s_l(t)\right] \quad (16)$$

Donde $l_l(t) = \sum_k r_{lk} s_k(t)$. Esta función premia aquellos píxeles con una relación de vecindad más fuerte y en menor medida los píxeles con una relación más débil, dada la relación de vecindad por r_{lk} como ya definimos anteriormente.

Por último tenemos que diseñar una función f que relacione el estado de las neuronas vecinas ($l_l(t)$) con la temperatura del sistema en el instante t. Los valores de disparidad previamente se proyectan al rango $[-1, 1]$ de forma lineal. Por este motivo, la función buscada debe estar saturada para que no tome valores inferiores a -1 ni superiores a 1.

En nuestra implementación hemos utilizado la tangente hiperbólica aprovechando que está saturada en los límites de interés. Su representación es la que se muestra en la figura 2.7.

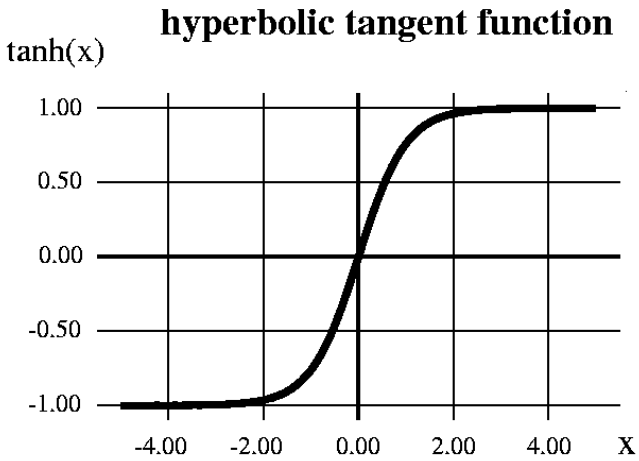

Figura 2.7. Función tangente hiperbólica.

2.1.3.5 Criterio de parada

Existen varias condiciones de parada, para determinar cuándo detener el proceso iterativo de enfriamiento.

- Cantidad máxima de iteraciones.
- No se obtiene una solución mejor después de varias iteraciones.

- Energía mínima del sistema. Donde la energía de nuestro sistema está definida por:

$$E = -\frac{1}{2}\sum_{l=1}^{N}\sum_{k=1}^{N} r_{lk}s_l s_k \quad (17)$$

2.1.4 Mapas cognitivos Fuzzy

Los Mapas Cognitivos Fuzzy (MCF) es otra de las técnicas que como en el caso de los SA trata de mejorar el mapa de disparidad. Son grafos difusos que representan relaciones causales. La información contenida, al ser difusa, permite un cierto grado de imprecisión en el conocimiento y los resultados obtenidos han de interpretarse de manera probabilista. La estructura de grafo permite de manera sistemática la propagación causal, en particular, el encadenamiento hacia delante y hacia atrás. Los MCF son especialmente aplicables en dominios de conocimiento borrosos.

Los MCF permiten modelar sistemas de retroalimentación con grados difusos de causalidad comprendidos en el intervalo $[0, 1]$. Para ello, primero hay que construir un diagrama del sistema mostrando las suposiciones iniciales del modelo. En el diagrama, cada nodo representa un evento difuso que ocurre en algún grado. Los nodos son conceptos causales y pueden modelar eventos, acciones, valores, metas o procesos.

Luego, se establece un vector de entrada con la condición inicial, se multiplica por la matriz derivada del diagrama y se obtienen resultados al iterar el vector resultante de la multiplicación con la misma matriz, obteniendo así retroalimentación para el sistema. Se debe declarar alguna condición de finalización de la iteración; generalmente se para de iterar cuando la diferencia entre una iteración y la iteración anterior no supera un cierto umbral. También se debe definir un número máximo de iteraciones para evitar ciclos infinitos o cómputos demasiados largos. En resumen, podemos decir entonces que los MCF son grafos difusos con retroalimentación o lo que es igual, redes neuronales backpropagation.

En nuestro proyecto vamos a combinar mediante un MCF la información obtenida de los distintos parámetros que caracterizan a cada píxel del mapa de disparidades obtenido con la de sus vecinos para suavizar dicho mapa y eliminar valores erróneos.

Comenzamos definiendo como estado inicial la imagen del mapa de disparidades que obtenemos con el método de correspondencia. Cada nuevo estado se obtiene después de calcular el nuevo valor de cada píxel en función de todos sus adyacentes. Iteramos hasta un máximo de iteraciones o hasta que la diferencia de un estado al anterior sea inferior a un épsilon dado, es decir, cuando la imagen se haya estabilizado.

Definimos entonces cada estado S_l de la siguiente forma:

$$s_l(t+1) = f\left(k\left(s_l(t) + \sum_k r_{lk} s_k\right)\right) - d_l s_l(t) \quad (18)$$

donde:

- f es una función de tipo sigmoide que trata de evitar valores dispersos en un conjunto cercano de píxeles.
- k es un factor de atenuación.
- el sumatorio en k es la aportación de los píxeles vecinos al píxel que se está tratando en cada momento.
- d_l es el peso del estado anterior en el cálculo del nuevo estado. Ajustando k y d podemos evitar cambios bruscos en las transiciones de un estado al siguiente.

3. Diseño

En este capítulo se presenta la arquitectura de diseño sobre la que se basa la aplicación y una detallada descripción del interfaz que ofrece la misma.

3.1 Arquitectura del sistema

Nuestro sistema al estar desarrollado en java es independiente del sistema operativo donde se ejecuta, únicamente necesitaremos disponer de la máquina virtual de Java en su versión 6.

Figura 3.1. Arquitectura del sistema.

La aplicación se diseña en base a los cuatro módulos que se especifican a continuación:

1. **GUI**: engloba todo lo relacionado con la interfaz gráfica. Tanto la interfaz principal de la aplicación como la visualización de los resultados obtenidos tras ejecutar los distintos métodos.
2. **Métodos**: son el núcleo de la aplicación. Contiene todos los métodos de visión estéreo implementados y descritos en el capítulo dos.

3. **Recursos**: es el conjunto de imágenes y archivos externos (ayuda…) usados por la aplicación.
4. **Útiles**: contiene herramientas útiles y comunes que pueden ser usados por varias clases a la vez.

La relación entre estos módulos puede verse en el siguiente diagrama de clases UML. Se ha obviado el módulo de recursos al ser éste un almacén de imágenes y utilidades.

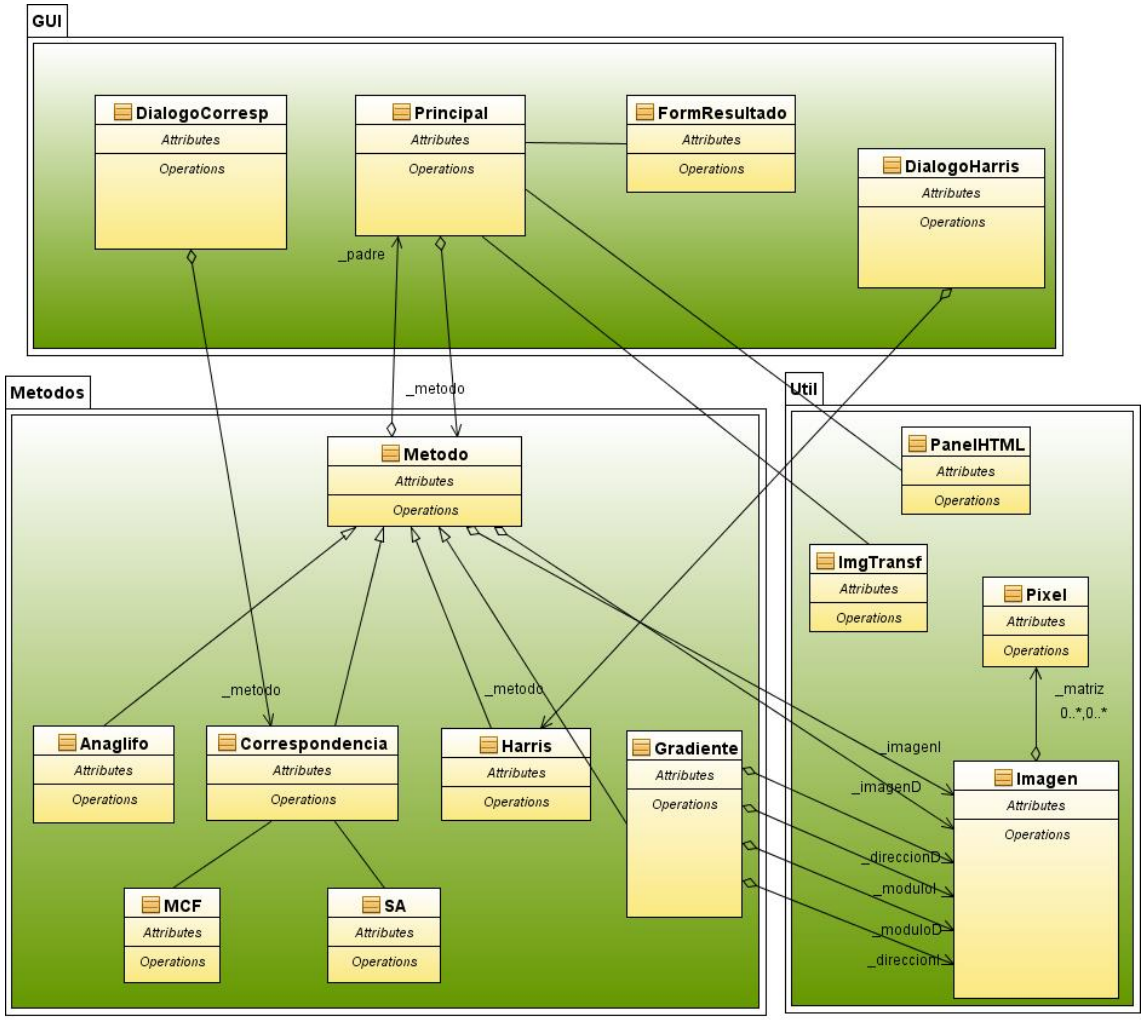

Figura 3.2. Diagrama de clases UML.

3.1.1 Descripción de los módulos

3.1.1.1 GUI

Este módulo contiene todas las interfaces de usuario disponibles en la aplicación. Consta de las siguientes clases:

- **Principal**: es el interfaz principal de la aplicación. En ella el usuario puede interactuar directamente con la aplicación seleccionando y ejecutando algoritmos de visión.
- **FormResultado**: es la interfaz que se le presenta al usuario tras la ejecución de un algoritmo. En ella podrá aplicar métodos de mejora de imagen sobre la obtenida mediante algún método basado en correspondencia.
- **DialogoCorresp**: es la ventana que se muestra al usuario para seleccionar el método a ejecutar basado en correspondencia, en él podrá fijar los parámetros (como por ejemplo: tamaño de ventana) que definen el método.
- **DialogoHarris**: análoga a la de correspondencia. Sin embargo, ofrece distintos parámetros configurables adaptados al algoritmo de Harris.

3.1.1.2 Métodos

Este es el módulo principal de la aplicación. En él encontramos una interfaz genérica para la implementación de los métodos. Está basado en el patrón Fachada.

Éste módulo posee las siguientes clases:

- **Método**: es una abstracción que contiene lo básico para la implementación de algoritmos de visión estereoscópica.
- **Anaglifo**: es la clase que se encarga de la obtención del anaglifo de dos imágenes estéreo dadas.
- **Correspondencia**: implementa los distintos algoritmos de correspondencia basada tanto el basado en el área como en las características. Es capaz de obtener el mapa de disparidad y el gradiente de imágenes estéreo.
- **Harris**: es la implementación del método de Harris de obtención de esquinas de imágenes.
- **MCF**: esta clase implementa la técnica de Mapas Cognitivos Fuzzy. Ésta técnica mejora los mapas de disparidad obtenidos mediante las correspondencias previas.

- **SA**: análogamente a MFC, ésta clase implementa la mejora a los mapas de disparidad conocida como enfriamiento simulado (*Simulated Annealing)*.

3.1.1.3 Útil

Este módulo es un 'cajón de sastre' que contiene funcionalidades muy útiles para otros módulos. Las clases importantes que ofrece son las siguientes:

- **PanelHTML**: esta clase es capaz de cargar código HTML en un JPanel para mostrarlo por pantalla. Es usado para mostrar la ayuda de la aplicación.
- **Píxel**: es una de las principales clases que posee éste módulo. Implementa las características básicas que posee un píxel.
- **Imagen**: es otra clase principal de la aplicación. Las imágenes cargadas pasan a ser una instancia de ésta clase haciendo que el uso y modificación de la imagen quede transparente en la implementación de los métodos.
- **ImgTransf**: es un conjunto de utilidades para las imágenes. Permite transformar valores de disparidades en forma de imagen (ya sea en escala de grises o en color), transformar la disparidad al rango [-1, 1] (usado en los algoritmos de mejora) y viceversa y por último también permite transformar una imagen obtenida mediante código java en una matriz de números flotantes.

3.1.2 Metodología de desarrollo

Para la implementación de la aplicación Visión 3D (nombre de la aplicación del proyecto) hemos usado el lenguaje Java en su versión JDK 1.6 y el entorno de desarrollo NetBeans 6.5.

Las etapas de desarrollo de nuestra aplicación son las siguientes:

- **Primera fase**: desarrollo de un interfaz sencillo, útil e intuitivo para poder probar rápidamente los algoritmos de visión. Además, nos proporciona una estimación de tiempo de ejecución de los mismos.
- **Segunda fase**: implementación de los algoritmos de visión estéreo. En esta fase es donde mayor número de problemas surgieron. No había ningún dato sobre las cámaras de las imágenes estéreo que teníamos y por tanto era muy difícil la depuración sobre los resultados obtenidos.

- **Tercera fase**: verificación y documentación de la aplicación.

En la implementación del los algoritmos de visión nos hemos apoyado en clases que posee Java para la obtención de las imágenes. Sin embargo, el procesado de las mismas se realiza a través de nuestra clase Imagen, dado que trabajar directamente sobre la imagen desde Java ralentiza la ejecución de los métodos y es conveniente trabajar sobre las matrices que la definen. Además, cada método es un hilo de ejecución. Por lo que no impide la interacción del usuario con la aplicación.

Por otro lado, lo más difícil ha sido la obtención de imágenes estéreo para las pruebas. Exceptuando las imágenes estéreo de artículos relacionados con Marte y alguna otra, la mayoría de imágenes estéreo vienen sin información acerca de las cámaras que las tomaron y por ello la correspondencia entre píxeles se dificultaba.

A pesar de los intentos por parte de la empresa TCP por conseguir imágenes reales del CNES francés o de la Agencia Espacial Europea, esto no ha sido posible por diversas razones técnicas y las conseguidas no han servido debido a las distorsiones que presentaban.

3.1.3 Funcionamiento

En cuanto al uso de la aplicación, el funcionamiento básico es el siguiente:

- Se cargan las dos imágenes estereoscópicas a tratar.
- Se elige un método a ejecutar sobre las mismas.
- Según el método se configuran sus parámetros.
- Se lanza el algoritmo a ejecutar.
- Se analizan los resultados obtenidos.
- Se ejecuta (opcionalmente y si la imagen lo permite) un algoritmo de mejora de la imagen resultado obtenida.

Para un mayor detalle en el funcionamiento de la aplicación ilustramos el funcionamiento de la aplicación con los siguientes diagramas de secuencia.

El diagrama de la figura 3.3 muestra el flujo que debe seguir un usuario de la aplicación para poder ejecutar un algoritmo.

Podemos observar que las operaciones de: configuración de método, obtener opciones… son propias de cada método concreto.

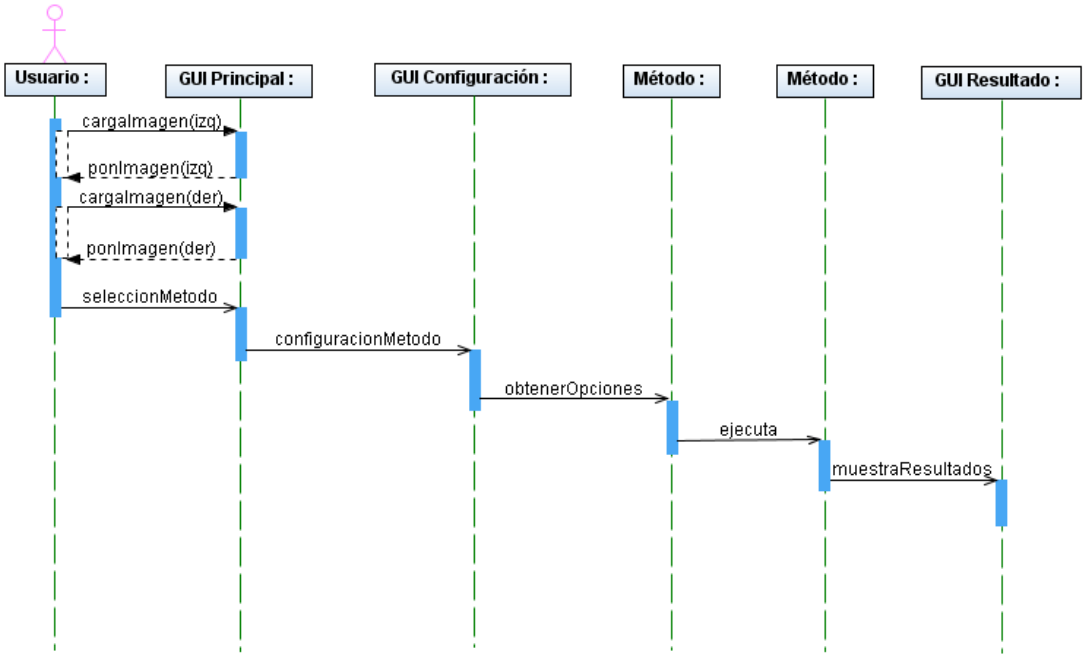

Figura 3.3. Diagrama de secuencia 1.

En caso de desear ejecutar un método y aplicar una mejora a su resultado el proceso a seguir por un usuario es análogo, tal y como se muestra en la figura 3.4.

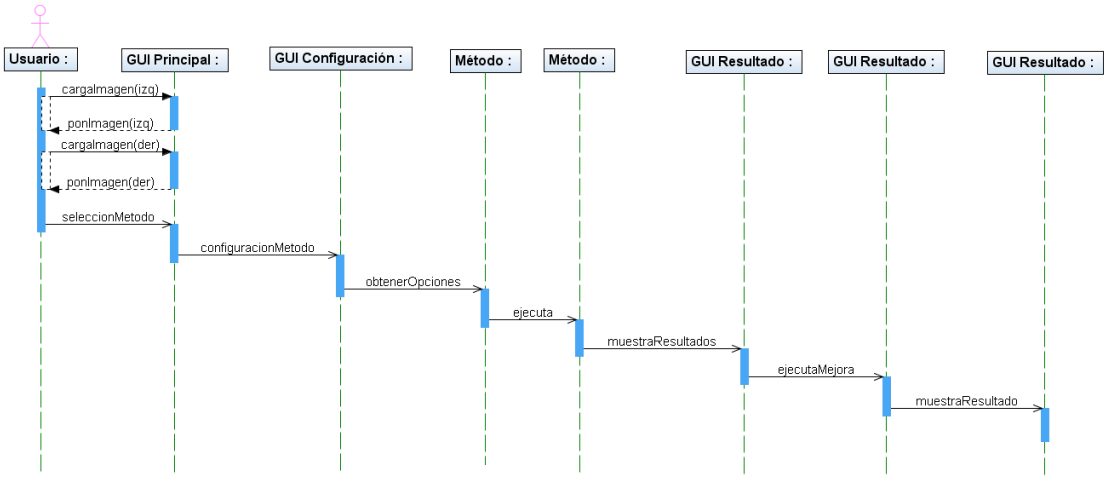

Figura 3.4. Diagrama de secuencia 2.

Como podemos ver, la ejecución de mejoras de la imagen obtenida se realiza desde la misma interfaz de resultado del método ejecutado. Incluso es posible guardar los resultados obtenidos donde el usuario desee, especificando convenientemente la ruta.

Aunque en los diagramas no aparece, el formulario principal presenta una actualización constante dado que muestra al usuario el tiempo estimado de ejecución del algoritmo.

La aplicación también dispone de distintas opciones como:

- Un menú de ayuda accesible desde el interfaz principal.
- Un menú de opción de hilos de ejecución a través del cual podemos configurar el número de unidades de proceso que tratarán las imágenes.
- Un menú con información relativa a los autores.
- Un menú en el que configurar el aspecto visual de la aplicación.

3.2 Interfaz de usuario

El interfaz de usuario consta fundamentalmente de dos ventanas, la principal y otra donde podremos ver el resultado de los distintos métodos implementados. También disponemos de otras ventanas auxiliares como la de ayuda y la de configuración de la propia aplicación y de los distintos métodos. De esta manera se proporciona al usuario un interfaz sencillo y fácil de usar.

3.2.1 Ventana principal

En la figura 3.5 podemos ver el logotipo de nuestra aplicación, este se muestra mientras que se carga la aplicación.

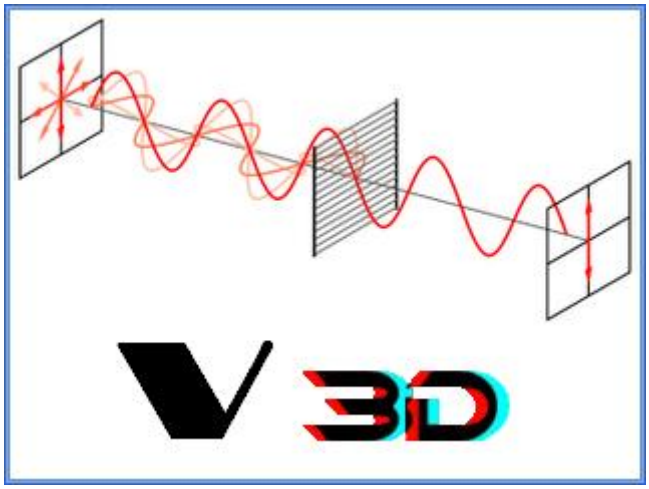

Figura 3.5. Logotipo.

La figura 3.6 muestra la ventana que se ofrece al usuario al iniciar la aplicación. En ella podemos observar un sencillo diseño en el que tenemos una barra de menú superior en la que podemos configurar la aplicación. En la parte central podemos realizar la carga de las dos imágenes a tratar. Finalmente en la parte inferior aparece un mensaje de estado del programa, el cual nos indicará el grado de progreso de la aplicación en cada momento.

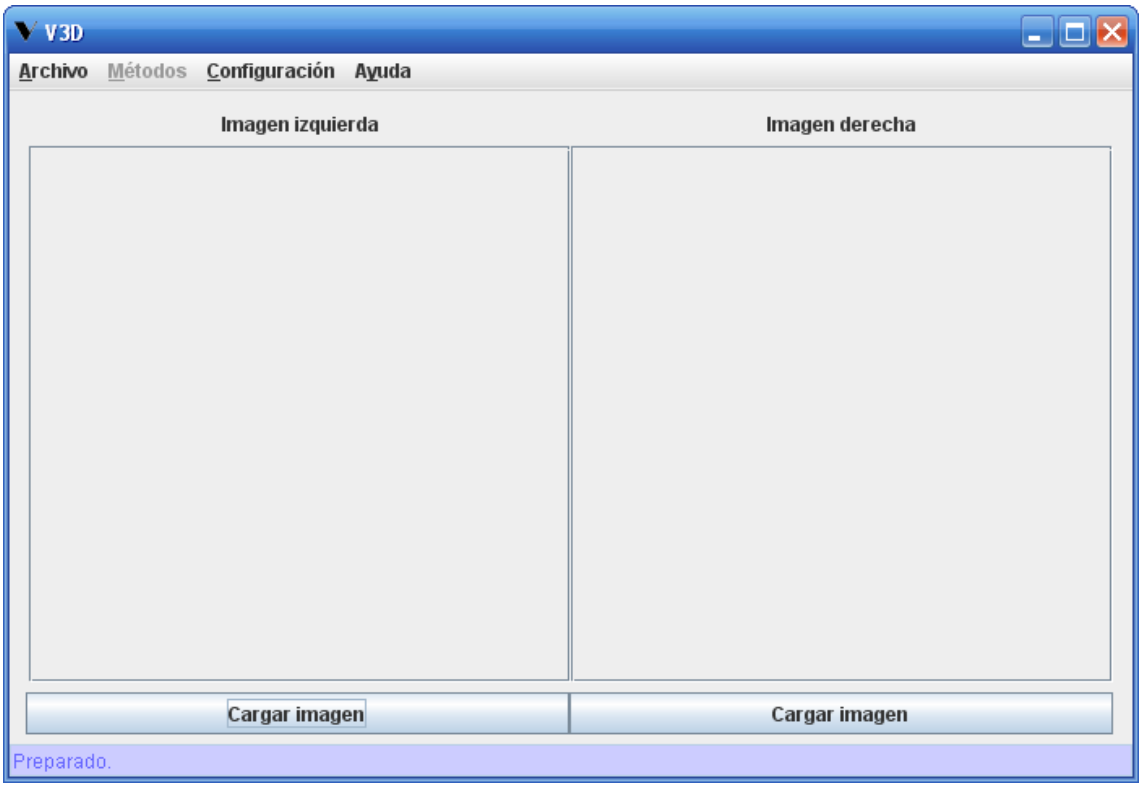

Figura 3.6. Ventana principal.

3.2.2 Ejecución y configuración de un método

Antes de ejecutar cualquier método se deberá cargar las dos imágenes del par estéreo que se quieren analizar. Estas dos imágenes deberán tener de la misma resolución para que la aplicación nos permita ejecutar un método. Para cargar las imágenes en la aplicación únicamente hay que presionar en los botones 'cargar imagen', que están situados en la parte inferior de la ventana principal y darán como resultado una ventana para elegir la ruta de la imagen a cargar.

Una vez cargadas ambas imágenes habrá que acceder al menú de métodos, como muestra la figura 3.7 y seleccionar uno de ellos.

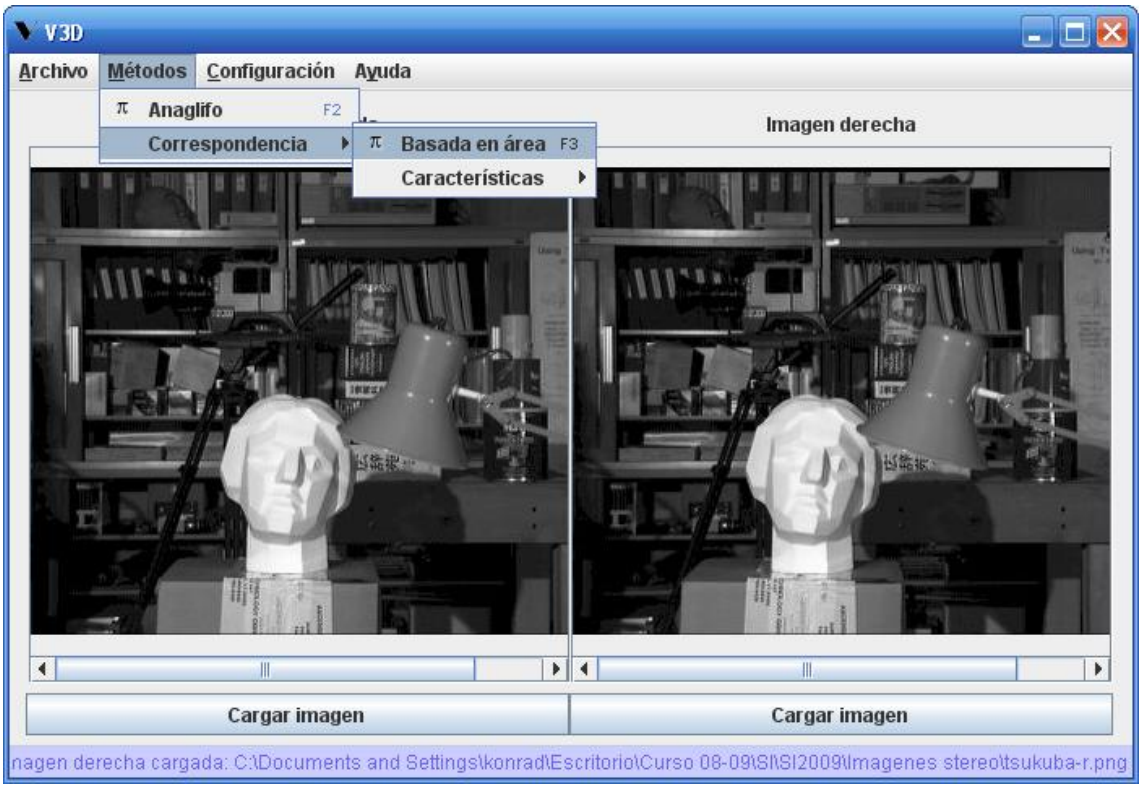

Figura 3.7. Ejecución de un método.

Una vez cargadas las imágenes. El interfaz principal permite la elección de ejecución de los métodos. Tras elegir uno de ellos, nos podemos encontrar con un interfaz del tipo mostrado en la figura 3.8.

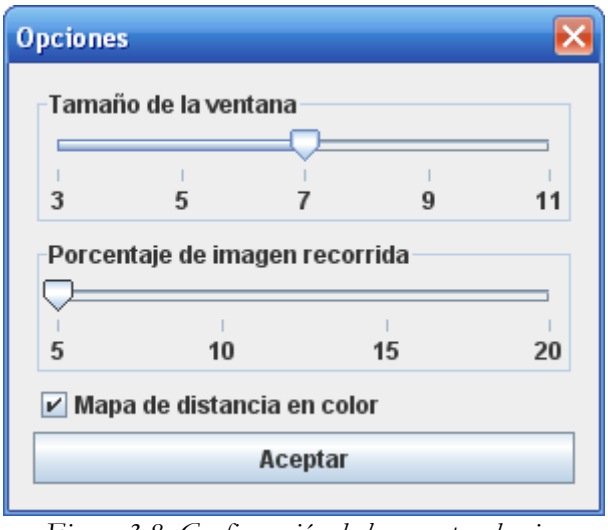

Figura 3.8. Configuración de la correspondencia.

Ésta ventana permite la configuración del método seleccionado por parte del usuario, pudiendo elegir manualmente el valor de los parámetros que desee.

3.2.3 Ventana de resultados

En la figura 3.9 la ventana muestra el resultado de aplicar un cierto método sobre las imágenes cargadas (en este caso correspondencia basada en el área). Como podemos observar, tiene un diseño simple e intuitivo ofreciendo el resultado directamente al usuario.

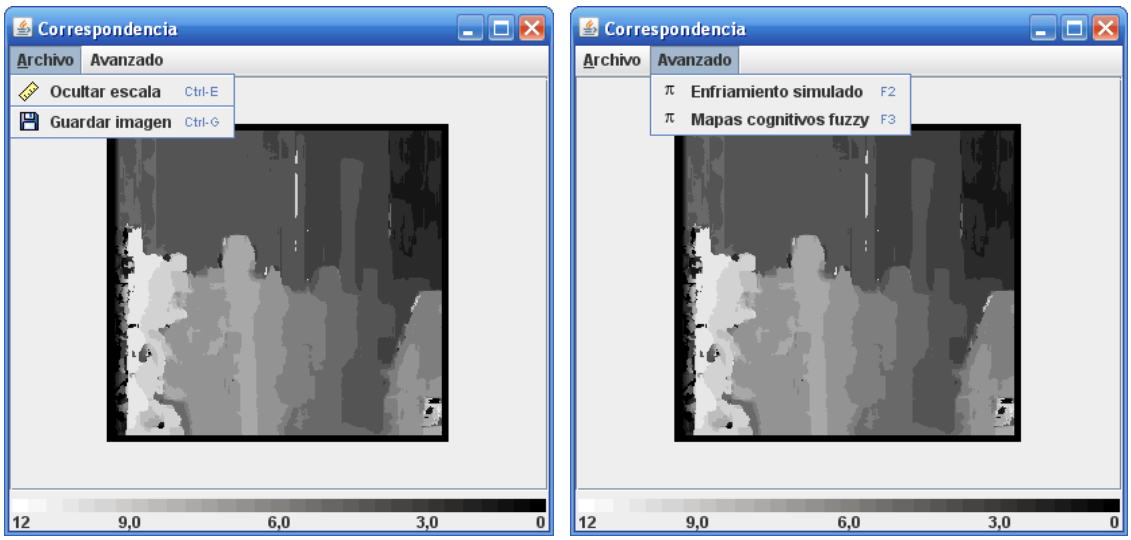

Figura 3.9. Ventana de resultados.

3.2.4 Configuración de unidades de proceso

La figura 3.10 muestra el sencillo interfaz en el que el usuario puede configurar cuántas unidades de proceso desea en la ejecución de los algoritmos de la aplicación. Sin embargo, esta funcionalidad no ha sido implementada por falta de tiempo.

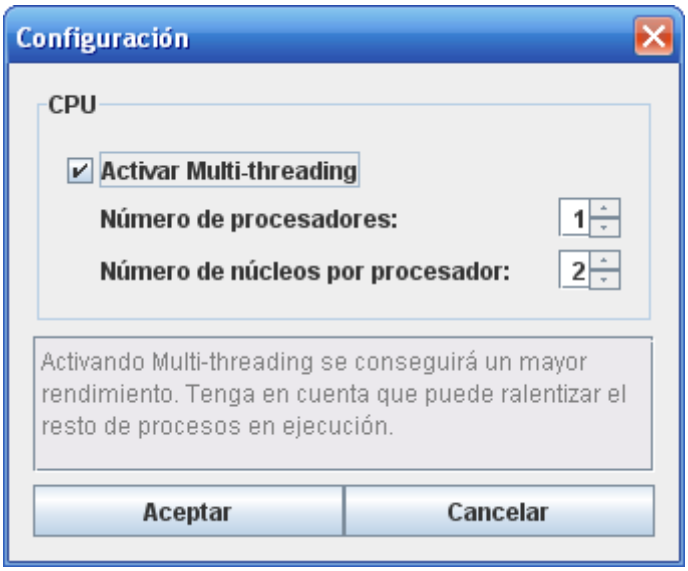

Figura 3.10. Configuración de la CPU.

3.2.5 Interfaz de ayuda

La ventana de la figura 3.11 muestra una sencilla ayuda en formato HTML donde se pueden consultar los detalles para utilizar la funcionalidad de la aplicación de una manera fácil e intuitiva.

Figura 3.11. Interfaz de ayuda.

4. Resultados

4.1 Algoritmos

El resultado de nuestros algoritmos puede analizarse desde dos puntos de vista:

- Rendimiento.
- Calidad del resultado.

El rendimiento es importante ya que este proyecto está pensado para una posible implementación en un vehículo autónomo. Por esta razón todos nuestros algoritmos implementados han sido pensados y diseñados para que sean lo más eficientes posible. Otro punto a destacar acerca del rendimiento es la posibilidad de paralelizar totalmente el proceso de obtener la correspondencia de las imágenes, ya que la implementación de los algoritmos, al no tener ninguna dependencia a la hora de procesar cada píxel de la imagen, lo permite.

En la tabla 4.1 podemos ver cómo ganamos velocidad de forma lineal por cada unidad de proceso que añadimos a nuestro sistema. Para ello analizamos un par estéreo de imágenes 384x288 con tamaño de ventana 11 y porcentaje de imagen recorrida 5.

Unidades de proceso	Tiempo (segundos)
1	13.24
2	6.93
4	3.60
8	1.87
16	0.99

Tabla 4.1. Relación entre unidades de proceso y tiempo de ejecución.

Nosotros nos vamos a centrar en la calidad del resultado obtenido, por lo que en los siguientes apartados analizaremos en profundidad desde este punto de vista los resultados de los algoritmos implementados.

4.1.1 Correspondencia

4.1.1.1 Correspondencia basada en el área

Antes de analizar cualquier resultado debemos tener en cuenta que el resultado obtenido por la correspondencia basada en el área en ningún caso puede ser perfecto. Estas imperfecciones pueden ser debidas a múltiples causas:

- **Mala calibración de las cámaras**: Las imágenes de la figura 4.1 y 4.2 fueron tomadas en la facultad de Ciencias Físicas de la UCM mediante un sistema estéreo fabricado por alumnos y profesores de la misma. En la figura 4.1 vemos cómo el tejado tiene distinta inclinación en ambas imágenes. Esto es un grave problema para la correspondencia basada en el área ya que esta sólo analiza la línea epipolar (horizontal).

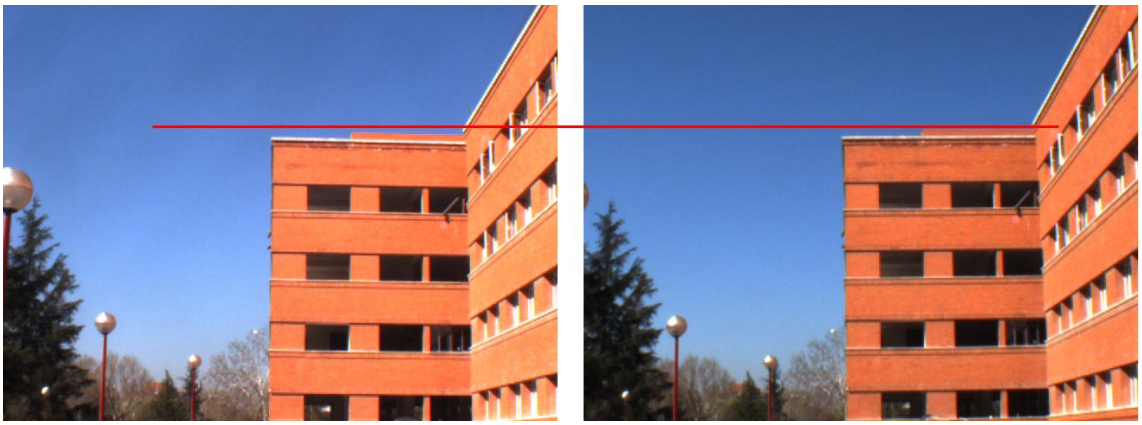

Figura 4.1. Mala calibración de las cámaras.

- **Distinta iluminación en cada imagen del par estéreo**: En la figura 4.2 podemos ver cómo influye la iluminación en las distintas imágenes debido a la separación de las cámaras (línea base).

Figura 4.2. Distinta iluminación en cada imagen del par estéreo.

- **Compresión en las imágenes**: Normalmente las imágenes son guardadas en archivos comprimidos mediante JPEG, el cual es un algoritmo de compresión con pérdida. Si la compresión es alta podremos perder información importante para los distintos métodos, lo que dará lugar a resultados menos precisos.
- **Oclusiones**: En una imagen puede verse algo que en la otra queda oculto por otro objeto debido a la perspectiva y el desplazamiento de una cámara con respecto a la otra. El error debido a esta causa es prácticamente imposible de corregir ya que nunca podremos encontrar el píxel que buscamos.

4.1.1.1.1 Ejemplo 1. Tsukuba.

Para un primer ejemplo vamos a utilizar las imágenes de la figura 4.3.

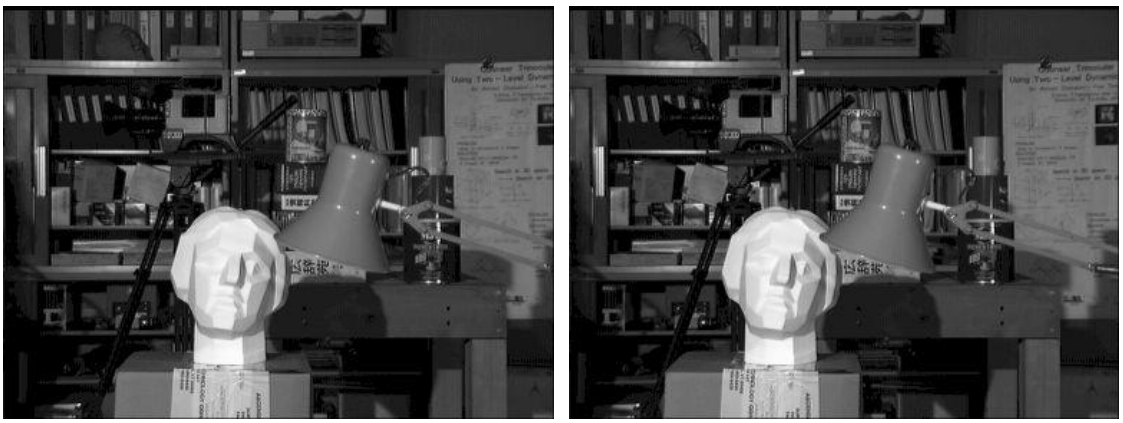

Figura 4.3. Par estéreo tsukuba.

El resultado de este primer ejemplo lo podemos ver en la figura 4.4. Aunque hay bastantes errores (píxeles con disparidad muy diferente a la de sus vecinos, prácticamente en la esquina superior derecha) podemos observar un mapa de disparidad donde se identifican los distintos objetos y su grado cualitativo de profundidad.

Figura 4.4. Correspondencia tsukuba (ventana 11, recorrido 5).

En la imagen resultante podemos ver que el objeto más cercano (según la escala de la figura 4.5, donde el blanco indica una disparidad muy alta y el negro disparidad 0) se corresponde a la lámpara, seguido de la cabeza, la mesa con las latas y la cámara.

Figura 4.5. Escala de grises.

4.1.1.1.2 Ejemplo 2. Corredor.

A continuación realizamos un análisis más detallado variando las propiedades de nuestro algoritmo de correspondencia para ver los distintos resultados obtenidos.

El par de imágenes estéreo que vemos en la figura 4.6 es sintético (generado por un computador). En esta imagen podemos apreciar una carencia del algoritmo de correspondencia basado únicamente en el área. El problema es el siguiente: al haber zonas (como los baldosines o la pared negra más cercana) con un área de píxeles con el mismo color y cercanos, el algoritmo de correspondencia trata de encontrar su homólogo en la línea epipolar (horizontal) de la otra imagen, de forma que al ser los píxeles y sus vecinos todos iguales la ventana se sitúa en el primer píxel que encuentra (considerando que se trata del mejor resultado).

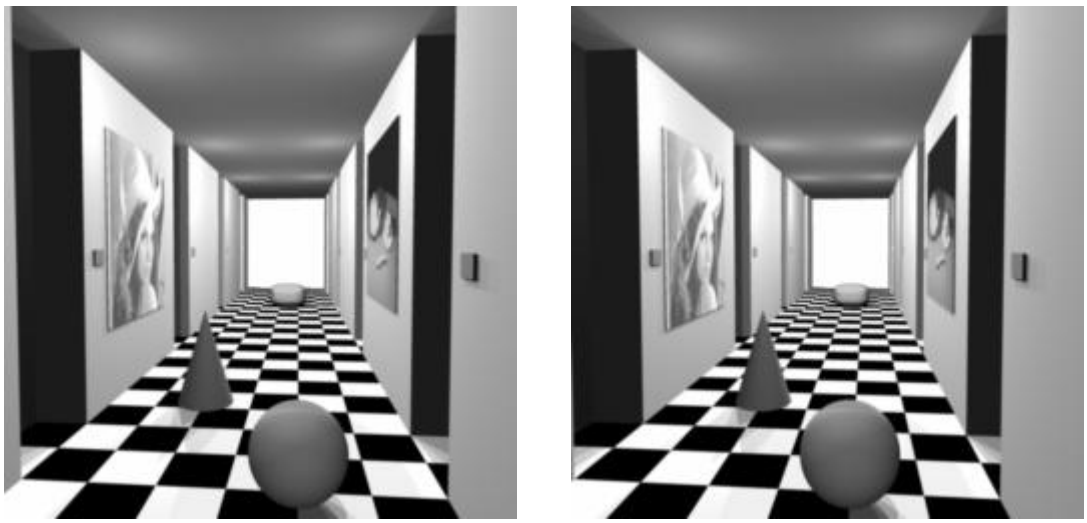

Figura 4.6. Par estéreo corredor.

De esta forma podemos ver cómo en el interior de los baldosines se obtiene una disparidad nula (como si estuviesen muy alejados), debiendo obtener una disparidad similar a la de los bordes de los mismos donde sí se obtiene un valor adecuado.

La figura 4.7 obtenida con un tamaño de ventana demasiado pequeño (3) muestra las imperfecciones comentadas en el párrafo anterior.

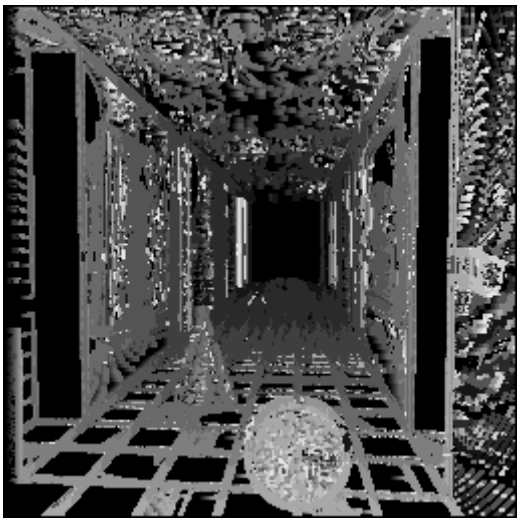

Figura 4.7. Correspondencia corredor (ventana 3, recorrido 5).

Estas imperfecciones pueden ser reducidas ampliando el tamaño de ventana para que el algoritmo tenga en cuenta mayor número de vecinos y tratar de que todos ellos no sean idénticos a la hora de buscar su homólogo en la otra imagen.

En la figura 4.8 podemos ver la mejora obtenida con respecto a la figura 4.7 ampliando únicamente el tamaño de ventana de 3 a 7. De esta forma pasaremos a tratar de 9 píxeles (tamaño de ventana 3) a 49 píxeles (tamaño de ventana 7).

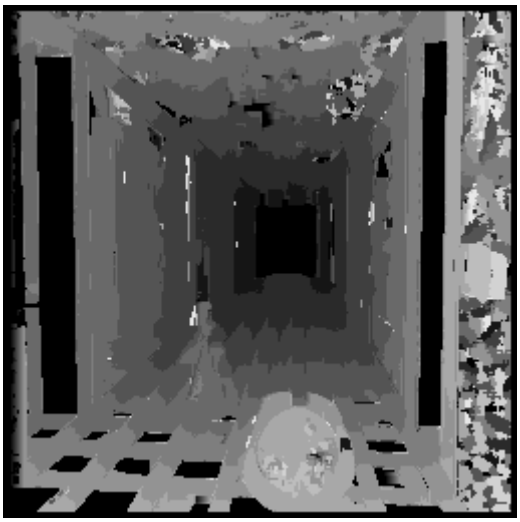

Figura 4.8. Correspondencia corredor (ventana 7, recorrido 5).

Por último volvemos a ampliar el tamaño de ventana a 11 (121 píxeles tratados) obteniendo mejores resultados, pero todavía con alguna deficiencia, tal y como se aprecia en la pared negra frontal de la figura 4.9.

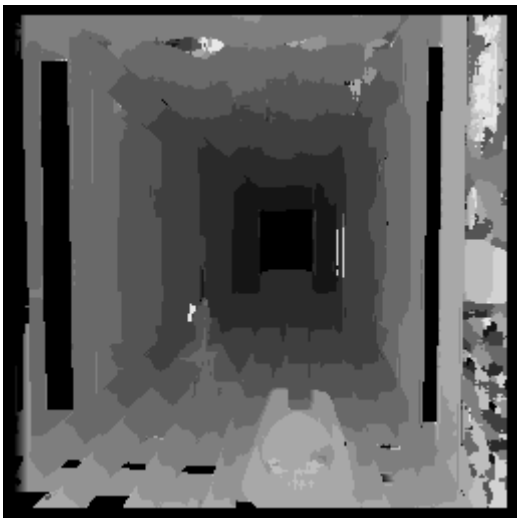

Figura 4.9. Correspondencia corredor (ventana 3, recorrido 5).

Es necesario considerar que al aumentar el tamaño de ventana mejora el resultado pero también consume mucho más tiempo de procesado de las imágenes como podemos ver en la tabla 4.2, la cual muestra el tiempo de procesado de un par estéreo de imágenes 256x256 con un tamaño de ventana variable y porcentaje de imagen recorrida 5.

Tamaño de ventana	Tiempo (segundos)
3	0.85
5	1.35
7	2.63
9	3.87
11	5.21

Tabla 4.2. Relación entre tamaño de ventana y tiempo de ejecución.

Muchas veces es recomendable obtener la correspondencia con un tamaño de ventana pequeño y aplicarle un algoritmo de mejora (véase apartados 4.1.2 enfriamiento simulado y 4.1.3 Mapas Cognitivos Fuzzy). De esta forma podemos obtener buenos resultados con un menor tiempo de procesamiento para una posible implementación que necesite obtener mapas de disparidad más o menos buenos en tiempo real.

4.1.1.1.3 Ejemplo 3. Parkmeter.

El último ejemplo que vamos a analizar corresponde a la imagen de un parquímetro (figura 4.10).

Figura 4.10. Par estéreo parkmeter.

Como en el ejemplo anterior del corredor, vamos a analizar el resultado de la correspondencia usando distintos tipos de ventana. Esta imagen ya no es sintética por lo que será más difícil de procesar, pero a favor tenemos que no presenta el problema del área de píxeles con un mismo color como pasaba en el ejemplo anterior.

En la figura 4.11, obtenida con un tamaño de ventana pequeño (3), podemos ver muchos píxeles con una disparidad incorrecta lo que genera una sensación de ruido en el mapa de disparidad.

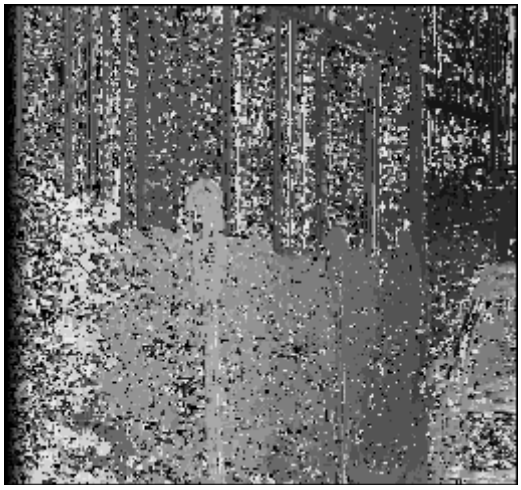

Figura 4.11. Correspondencia parkmeter (ventana 3, recorrido 5).

A continuación podemos ver, en la figura 4.12, cómo se reduce el ruido en el mapa de disparidad al aumentar el tamaño de ventana. Como comentábamos anteriormente también a costa de un mayor tiempo de procesamiento.

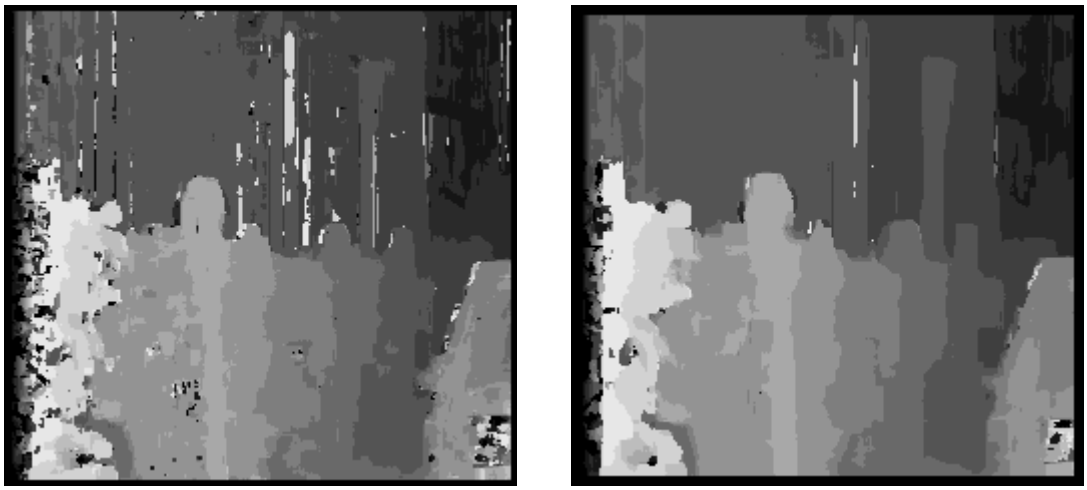

Figura 4.12. Correspondencia parkmeter (ventana 7 y 11, recorrido 5).

Antes comentábamos que podemos ganar velocidad obteniendo la disparidad con un tamaño de ventana pequeño y aplicarle un algoritmo de mejora. Otro método más simple, aunque no obtiene resultados tan buenos, es aplicar un filtro de suavizado al mapa de disparidad para tratar de eliminar el ruido. Podemos ver el resultado de esta operación en la figura 4.13.

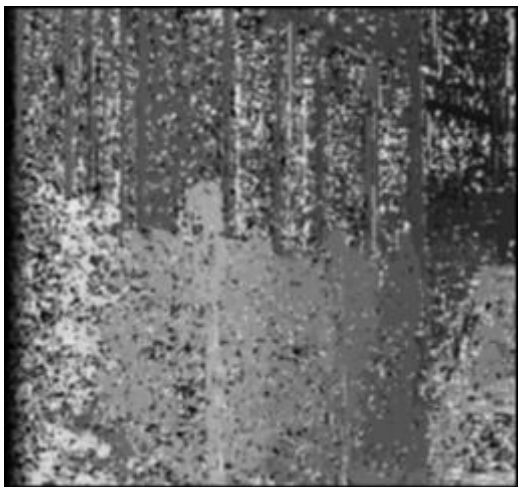

Figura 4.13. Correspondencia (ventana 3) + Filtro gaussiano.

4.1.1.2 Correspondencia basada en las características

La correspondencia basada en el área puede cometer errores, sobre todo si las imágenes contienen ruido o no han sido obtenidas correctamente (mal alineamiento de las cámaras, distinta iluminación en las imágenes, imágenes tomadas en distintos instantes de tiempo, etc.). Para intentar solucionar estos errores una técnica consiste en obtener características de las imágenes, con las que se intenta identificar la disparidad de las imágenes allí donde la correspondencia basada en el área no sea capaz de obtener un valor adecuado.

Para analizar los siguientes apartados vamos a utilizar el par estéreo de la figura 4.3.

4.1.1.2.1 Gradiente

Una de las características que podemos obtener de las imágenes es el gradiente en cada píxel de la imagen. En la figura 4.14 se representa en forma de imagen la dirección del gradiente de cada una de las dos imágenes del par estereoscópico.

Figura 4.14. Dirección de corredor.

También extraemos el módulo del gradiente, con lo que conseguimos obtener los bordes de los objetos presentes en las dos imágenes. El hecho de tener en cuenta estas dos características resulta ser muy útil a la hora de determinar la disparidad en puntos de la imagen conflictivos. El resultado en forma de imagen del módulo del gradiente lo podemos ver en la figura 4.15.

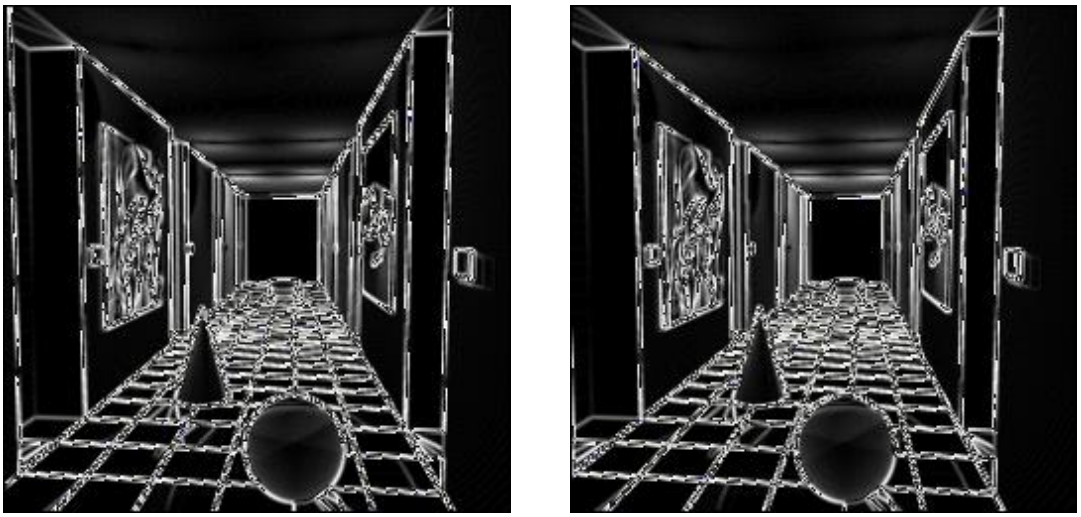

Figura 4.15. Módulo de corredor.

4.1.1.2.2 Harris

El método de Harris, comentado en la sección 2.1.2.3, resulta muy eficaz en la correspondencia basada en las características ya que obtiene los puntos de interés de las imágenes. Para mejorar la correspondencia basada en el área, el primer paso consiste en identificar los puntos de interés de la imagen izquierda con los correspondientes de la imagen derecha, con esto conseguimos obtener la disparidad de un punto de la imagen con exactitud. Una vez extraídos los puntos de interés identificados podemos ayudar al algoritmo de correspondencia basado en el área cuando éste no sea capaz de obtener la disparidad de un píxel y esté cercano a uno de los puntos identificados, de forma que se le asignará la disparidad obtenida por el punto de interés.

En la figura 4.16 podemos ver el resultado de aplicar Harris a las imágenes de ejemplo.

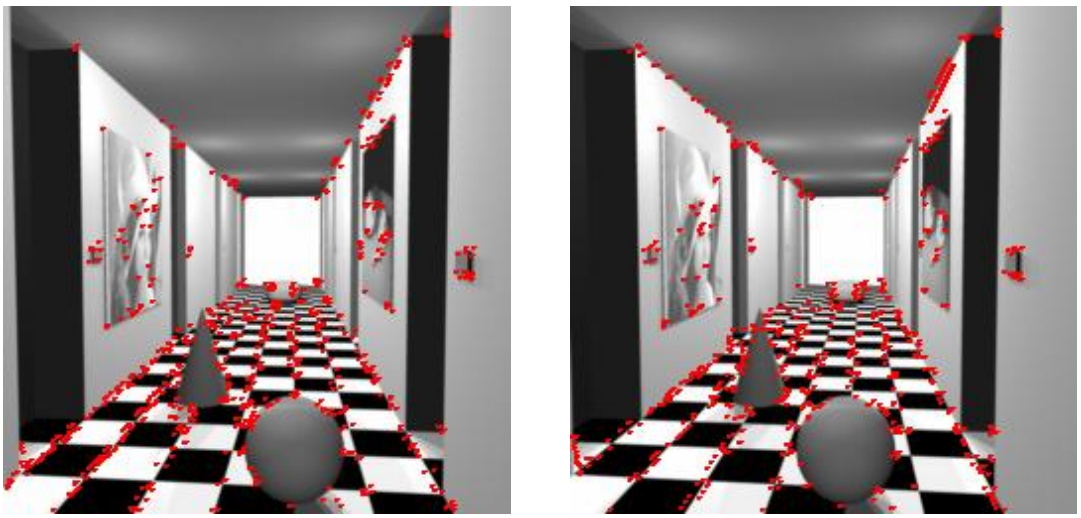

Figura 4.16. Harris (umbral = 200, k = 4).

Dependiendo del número de puntos de interés que necesitemos obtener podemos modificar el parámetro umbral. Con un umbral pequeño obtenemos un mayor número de puntos de interés (figura 4.16) y al contrario, con un valor de umbral mayor obtenemos mayor número de puntos de interés (figura 4.17).

Para imágenes complejas, como la de la figura 4.3, necesitaremos un mayor número de puntos de interés, de forma que nos será más útil el resultado obtenido en la figura 4.16 con un umbral más pequeño. Pero no siempre un mayor número de puntos de interese será mejor.

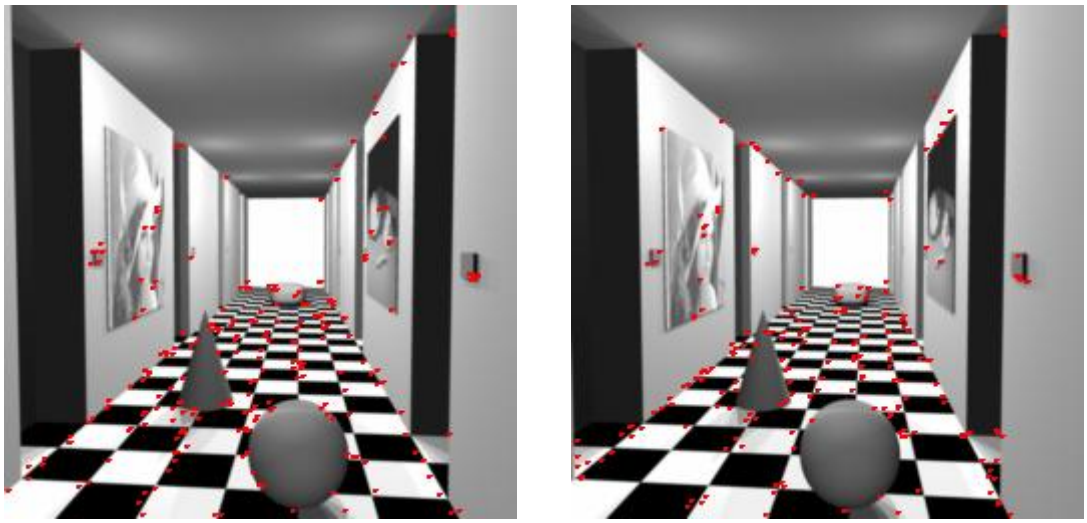

Figura 4.17. Harris (umbral = 300, k = 4).

4.1.2 Enfriamiento simulado

4.1.2.1 Ejemplo 1. Tsukuba.

En este apartado analizamos el resultado obtenido mediante el procedimiento de enfriamiento simulado aplicado a la mejora de los mapas de disparidad obtenidos mediante correspondencia basada en el área.

En la figura 4.18 podemos ver el resultado mejorado respecto del ya obtenido en la figura 4.13. Para mostrar mejor el comportamiento del algoritmo hemos mostrado el mapa de disparidad en color. Este mapa es el mismo que el representado en escala de grises en la figura 4.5 lo único que la escala es en color, como puede observarse en la figura 4.19.

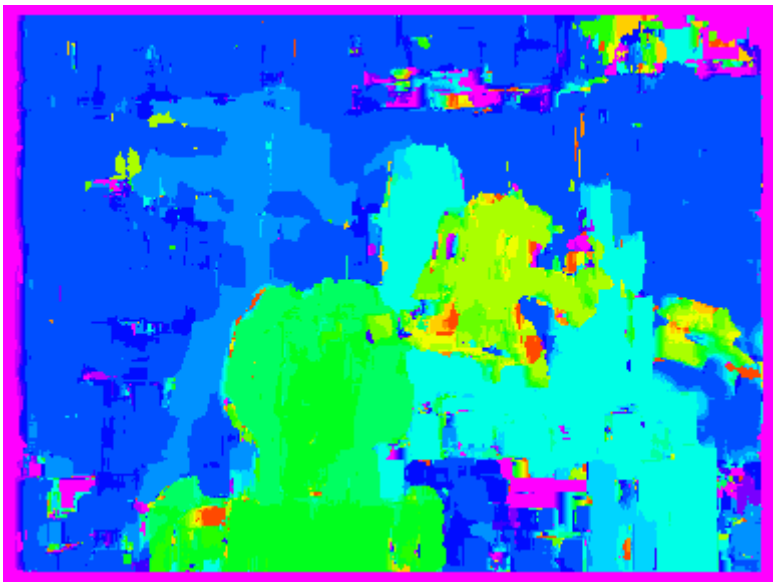

Figura 4.18. Correspondencia (ventana 11, recorrido 5).

Figura 4.19. Escala en color.

El resultado obtenido en la figura 4.18 puede considerarse aceptable, si bien podemos encontrar bastantes zonas de la imagen con valores de disparidad que apreciablemente son incorrectos.

Como explicamos en la parte de análisis se trata de hacer converger el mapa de disparidad original (figura 4.18) a otro con energía mínima de forma que valores de disparidad mal calculados sean 'arrastrados' por sus vecinos a los valores de éstos.

En la figura 4.20 podemos observar el resultado obtenido mediante enfriamiento simulado. Como se puede apreciar el 'ruido' se reduce de forma considerable, esto es debido a que los píxeles con disparidades erróneas rodeados de una disparidad correcta son modificados. Si tenemos un 1 rodeado de disparidades entorno a 9 nuestro algoritmo intentará rebajar la energía del sistema de forma que iteración a iteración si ese 1 no es realmente un 1 éste tenderá a valores cercanos a 9.

También podemos ver cómo define mejor las formas de los objetos de la imagen. Ahora tenemos claramente cinco planos en la imagen (de mayor a menor cercanía):

- Rojo – Naranja: Lámpara.
- Amarillo – Verde: Escultura.

- Azul claro: Mesa con sus libros y latas.
- Azul oscuro: Cámara.
- Violeta: Fondo.

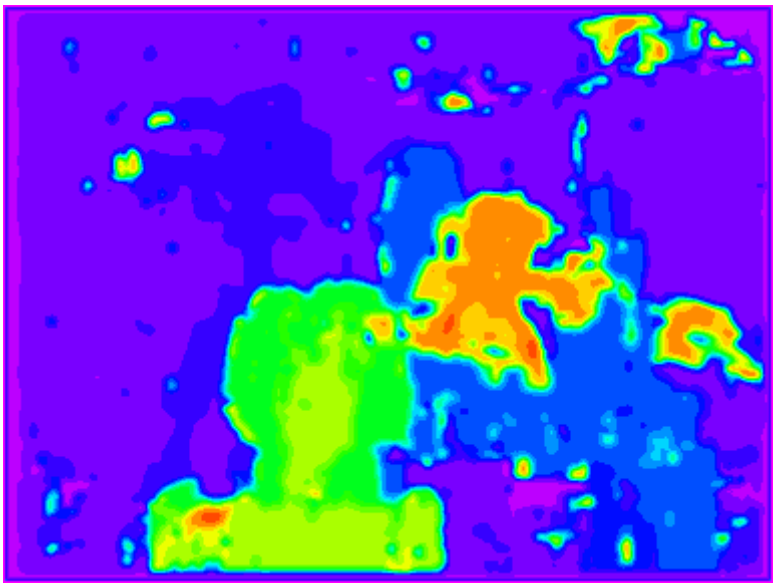

Figura 4.20. Enfriamiento simulado (ventana 11, recorrido 5).

4.1.2.2Ejemplo 2. Trees.

A continuación vamos a analizar las imágenes estéreo de la figura 4.21. Estas imágenes han sido tomadas con una mayor separación entre las cámaras por lo que tendremos que aumentar el parámetro de porcentaje de imagen recorrida. Este parámetro influye gravemente en el rendimiento del procesamiento cuando se aumenta ya que necesitará recorrer más imagen para encontrar el homólogo del píxel que buscamos. Sólo en este caso y cuando las imágenes sean tomadas a corta distancia se deberá aumentar este parámetro.

Figura 4.21. Par estéreo trees.

Si obtenemos el mapa de disparidad mediante correspondencia basada en el área obtenemos la imagen de la figura 4.22. En este mapa de disparidad podemos observar algunos errores debidos principalmente a la complejidad de la imagen.

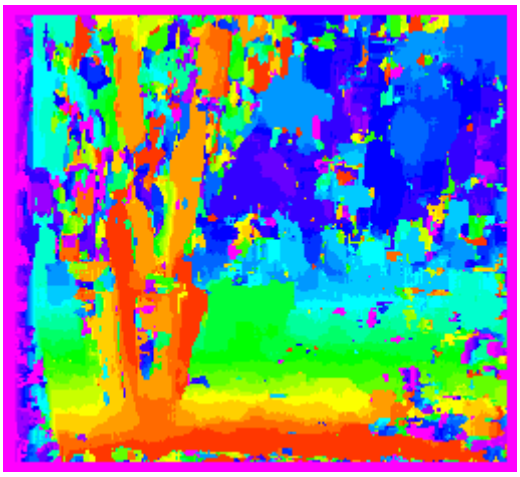

Figura 4.22. Correspondencia (ventana 11, recorrido 10).

Si al mapa de disparidad de la figura 4.22 le aplicamos enfriamiento simulado obtenemos el mapa de disparidad mejorado de la figura 4.23. En este mapa todavía persiste algún error, no obstante observamos cómo define mejor el contorno del árbol y del tronco que está en el centro de la imagen.

Si un vehículo autónomo analizara esta imagen empezaría a hacer un barrido desde abajo, entonces detectaría que a la izquierda la disparidad es muy alta (zona del árbol) lo que significa que existe un obstáculo ya que si no lo hubiera la distancia sería mayor por lo que la disparidad sería menor, como ocurre en el lado derecho de la imagen donde no encontraría ningún obstáculo.

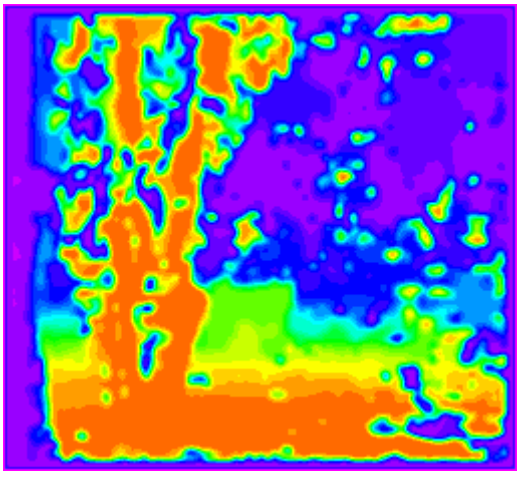

Figura 4.23. Enfriamiento simulado (ventana 11, recorrido 10).

4.1.3 Mapas cognitivos Fuzzy

4.1.3.1Ejemplo 1. Tsukuba.

En esta sección, al igual que en la anterior, el objetivo consiste en mejorar los mapas de disparidad mediante el segundo algoritmo de mejora implementado con esta finalidad. En este caso hemos implementado un algoritmo basado en Mapas Cognitivos Fuzzy.

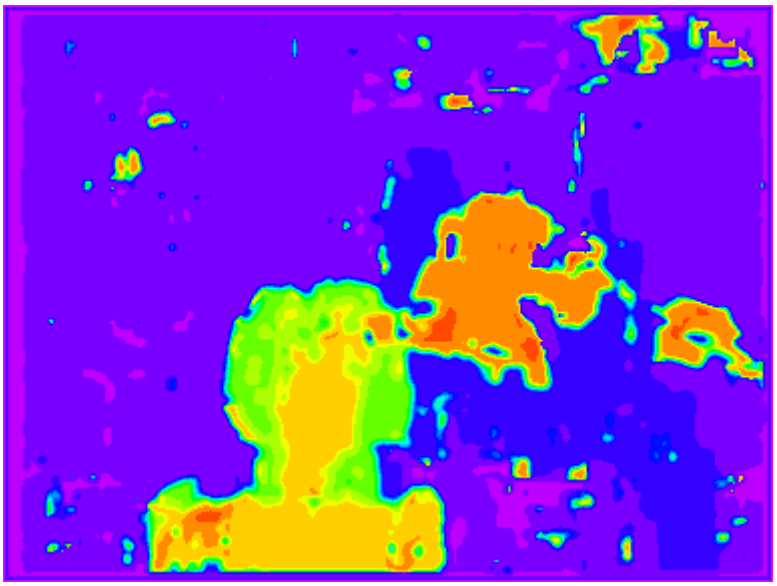

Figura 4.24. Correspondencia (ventana 11, recorrido 5).

De nuevo hemos vuelto a usar la imagen obtenida por la correspondencia basada en el área de la figura 4.18. Al igual que el enfriamiento simulado vemos cómo mejora el mapa de disparidad, si bien en este caso el algoritmo se muestra más agresivo. Esto se aprecia por el hecho de que la disparidad asociada a la cámara ha desaparecido.

4.1.3.2Ejemplo 1. Trees.

En último lugar analizamos el mapa de disparidad de la figura 4.22. El resultado obtenido, figura 4.25, es también similar al que se obtuvo mediante el procedimiento de enfriamiento simulado.

Si observamos el resultado, aunque se elimina una parte importante del 'ruido' del mapa de disparidad original, todavía aparecen zonas con valores altos de disparidad que el método no ha sido capaz de identificar (cuarto superior derecho de la figura 4.25).

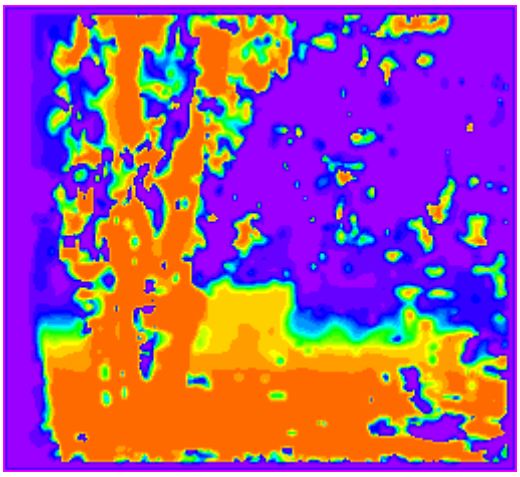

Figura 4.25. Correspondencia (ventana 11, recorrido 5).

Por último cabe destacar que tanto el algoritmo enfriamiento simulado como los Mapas Cognitivos Fuzzy consumen muy poco tiempo de procesamiento, decimas de segundo, para obtener un resultado aceptable.

5. Conclusiones

5.1 Descripción del desarrollo

En esta sección se describe más detalladamente los procesos y problemas por los que ha pasado el proyecto durante sus etapas de desarrollo.

5.1.1 Primera fase

Durante la primera fase del proyecto, la cual finalizó en Diciembre, nos centramos en la implementación del interfaz gráfico de la aplicación. Pensamos en una implementación sencilla y útil desde el primer momento para poder probar las imágenes lo antes posible.

Tras investigar sobre el lenguaje a utilizar, finalmente nos rendimos a la tecnología Java en su versión JDK 1.6 para el desarrollo de la aplicación, la razón principal es debida a que se trata de un lenguaje multiplataforma y con muchas clases implementadas para el tratamiento de imágenes.

Esta fase se centró principalmente en la investigación de lo que Java nos proporcionaba en cuanto a visión y comprensión del objetivo que nos habíamos planteado en este proyecto.

5.1.2 Segunda fase

Esta fase comprendida principalmente entre Enero y Abril supuso la etapa de desarrollo propiamente dicha de la aplicación.

Durante los inicios de ésta etapa comenzamos a implementar algunos de los métodos expuestos en esta memoria, por lo que simultáneamente empezamos a tener ciertos problemas. Estos problemas se sintetizan como sigue:

- Las imágenes estéreo que usamos no teníamos la certeza de que fuesen puras. No poseíamos ninguna información de cómo habían sido captadas. Por ello, algunas de ellas daban resultados aceptables mientras que en otras parecía que el algoritmo estaba fallando. Esto nos hizo depurar y cambiar varias veces los algoritmos,

finalmente conseguimos imágenes puras y tomadas sobre la epipolar las cuales funcionaban perfectamente con los algoritmos que habíamos implementado.

- Los algoritmos tardaban mucho en ejecutarse. Esto era debido a que las clases de Java aplicaban sus operaciones directamente sobre la imagen que estaba almacenada en disco. En consecuencia, tuvimos que crearnos unas clases para manejo de imágenes en las cuales volcábamos la información obtenida a través de las clases de Java. Con esto conseguíamos que se trabajara desde memoria secundaria. Gracias a ello, los tiempos de ejecución de los algoritmos sobre imágenes de tamaño considerable bajaron exponencialmente.
- En la robótica y más concretamente en los robots lanzados a Marte es esencial que el tiempo de ejecución de los algoritmos sea el mínimo posible. A causa de esto, intentamos investigar sobre cómo hacer que equipos multiprocesador ejecutaran el procesado de la imagen con los procesadores que el usuario indicase. Sin embargo, pospusimos esta opción para el final, ya que quedaba fuera de los objetivos planteados.

A finales del mes de Abril prácticamente terminamos toda la parte de diseño e implementación del proyecto. Durante este mes se probó exhaustivamente la funcionalidad de los algoritmos y se depuró hasta el más mínimo detalle. También se hizo que toda ejecución fuera en hebras para que el usuario no pensara que la aplicación había fallado, mostrándole por pantalla el estado de la aplicación en todo momento.

5.1.3 Tercena fase

Esta fase, abarcando el mes de Mayo ha consistido principalmente en la elaboración de la documentación sobre el proyecto.

La memoria del proyecto se ha tratado que tenga un enfoque algo diferente a las documentaciones típicas de Ingeniería del Software intentando evitar en todo momento un relleno excesivo con Casos de Uso, riesgos, etcétera. Nuestro objetivo ha sido explicar de manera detallada el contenido de la aplicación, cómo ha sido realizada sin entrar en detalles individuales y de iteraciones realizadas en el desarrollo centrándonos más sobre la implementación de los algoritmos.

Debido al interés de los algoritmos por parte de la empresa TCP, pensamos en un estudio a fondo de los algoritmos con el fin de conseguir un rendimiento aceptable de los mismos, de ahí el contenido teórico descrito en el capítulo dos.

5.2 Conclusiones del trabajo realizado

De los algoritmos y la aplicación realizada sacamos las siguientes conclusiones:

- El interfaz gráfico ha sido un éxito. Ha sido probado por técnicos de la empresa TCP con resultados satisfactorios por su uso fácil y directo a la hora de realizar las ejecuciones de los algoritmos.
- Los algoritmos implementados funcionan perfectamente sobre imágenes estéreo puras cuyas cámaras no hayan tomado las imágenes izquierda y derecha con distinta inclinación. Se dispone de un gran repertorio de imágenes sobre las que se pueden probar, observando que los resultados son los esperados sobre este tipo de imágenes.
- Los algoritmos que mejoran los resultados obtenidos de correspondencia basada en el área han surgido de las necesidades planteadas por TCP. Realizan una gran limpieza de ruido que el algoritmo básico no puede eliminar. Estos algoritmos dejan la imagen en perfectas condiciones para un posible reconocimiento de objetos o detección de obstáculos. Por ello podemos concluir que estos dos algoritmos constituyen el 'grueso' de nuestra aplicación.
- Como hemos comentado con anterioridad nos hubiese gustado que máquinas con varios procesadores pudiesen aprovecharse de ello para mejorar aún más los tiempos de ejecución, si bien no ha sido posible por falta de tiempo.
- Un aspecto relevante a destacar es el hecho de haber tomado contacto con el mundo empresarial a través de la empresa TCP.

5.3 Trabajos futuros

Visión 3D en un principio fue diseñado con el objetivo de ofrecer una planificación de caminos a robots espaciales. Sin embargo, ha quedado como una aplicación que ejecuta distintos algoritmos que obtienen características importantes de las imágenes estéreo por las dificultades surgidas y como consecuencia de la falta de tiempo.

Como mejoras y usos que podría darse al proyecto proponemos las siguientes:

- Aplicación didáctica sobre algoritmos de visión estereoscópica. Esta aplicación se puede ampliar con los numerosos algoritmos de visión estereoscópica que existen en la literatura y convertirse en una aplicación didáctica para comprensión de los mismos para estudiantes de este campo.
- Podemos ampliar el proyecto hasta que llegase al objetivo propuesto, es decir, que gracias a unas características dadas de las imágenes estereoscópicas y un reconocimiento de objetos u obstáculos de las mismas, un robot pudiese moverse sin problemas por terrenos rocosos o problemáticos. Para ello sería necesario poseer algún sistema que captara imágenes estereoscópicas cada cierto tiempo para que la aplicación trabajase y construyese rápidamente el nuevo mapa de navegación del robot.
- El proyecto podría enfocarse desde el punto de vista de la optimización de los algoritmos de visión estereoscópica y del procesamiento paralelo de imágenes para ganar el mayor rendimiento posible del procesado. Esto es un campo importantísimo en la robótica dado que los robots suelen tener una capacidad de proceso muy limitada.

Finalmente reseñar que el proyecto puede ampliarse con distintos enfoques y queda totalmente abierto a nuevas mejoras en el campo de la visión estereoscópica.

6. Bibliografía

[1]. Gonzalo Pajares, Jesús M. de la Cruz, José M. Molina, Juan Cuadrado y A. López; *Imágenes Digitales: Procesamiento práctico con JAVA;* RA-MA 2003;

[2]. Gonzalo Pajares y Jesús Manuel de la Cruz; *Ejercicios Resueltos de Visión por Computador;* RA-MA 2007;

[3]. Gonzalo Pajares y Jesús Manuel de la Cruz; *Visión por Computador: imágenes digitales y aplicaciones;* RA-MA, 2008 (2ª Edición);

[4]. R. C. González y R.E. Woods; *Digital Image Processing;* Prentice Hall, 2002;

[5]. M. Modesti, M.Pedrazzini, L.Canali, E.Destefanis; *Calibración de un sistema de visión estéreo para navegación de un AGV;*

[6]. J. N. Maki, J. F. Bell III, K. E. Herkenhoff, S. W. Squyres, A. Kiely, M. Klimesh, M. Schwochert, T. Litwin, R. Willson, A. Johnson, M. Maimone, E. Baumgartner, A. Collins, M. Wadsworth, S. T. Elliot, A. Dingizian, D. Brown, E. C. Hagerott, L. Scherr, R. Deen, D. Alexander and J. Lorre; *Mars Exploration Rover Engineering Cameras*; Journal of geophysical research, vol. 108, no. E12, 8071, doi:10.1029/2003JE002077, 2003.

7. Anexos

7.1 Manual de usuario

Este anexo contiene el manual de usuario de la aplicación V3D, el cual permitirá a cualquier usuario ejecutar la aplicación y todos los métodos que ofrece.

7.1.1 Inicio de la aplicación

Para ejecutar la aplicación del proyecto es imprescindible tener instalado la máquina virtual de Java en su versión JRE 1.6.

Para iniciar la aplicación se puede optar por una de las siguientes alternativas:

- Ejecutar el siguiente comando:
  <Ruta de java>/bin/java.exe -jar -Xmx512m <Ruta de V3D>/Proyecto.jar
- Abrir el proyecto Java mediante el entorno de desarrollo NetBeans y ejecutarlo.
- Ejecutar el archivo por lotes Play.bat, el cual es un script (MS-DOS) que ejecuta la aplicación del proyecto (Sólo para Windows).

7.1.2 Interfaz principal

Tras iniciar la aplicación, se muestra al usuario la ventana principal, la cual posee las siguientes funcionalidades:

- Un menú Archivo mediante el cual el usuario puede únicamente cerrar la aplicación.
- Un menú Métodos el cual permanecerá inactivo hasta que el usuario cargue las imágenes estereoscópicas con las que desee trabajar. Este menú posee la ejecución de los métodos de visión.
- Un menú Configuración que permite fijar ciertos parámetros de la aplicación.
- Un menú de Ayuda con información sobre el uso de la aplicación así como información acerca de sus autores.
- En la parte central hay dos zonas en las que podemos cargar imágenes.

- En la parte inferior de la ventana encontramos una barra de estado de la aplicación. Esta barra muestra, en cada momento, lo que está haciendo la aplicación. Por ejemplo si está ejecutando un algoritmo, muestra al usuario el tiempo estimado en el que este finalizará su ejecución.

En la figura 7.1 podemos ver la ventana principal de la aplicación nada más ser ejecutada.

Figura 7.1. Ventana principal.

7.1.3 Ejecutar un método

Para poder ejecutar un método es imprescindible primero haber cargado las dos imágenes estéreo a tratar. Para ello lo haremos mediante la parte central de la ventana principal. Pulsando el botón cargar imagen (por ejemplo: de la izquierda) podremos cargar una imagen.

Una vez pulsado el botón cargar imagen se mostrará una ventana como la de la figura 7.2 donde podremos elegir la ruta de la imagen.

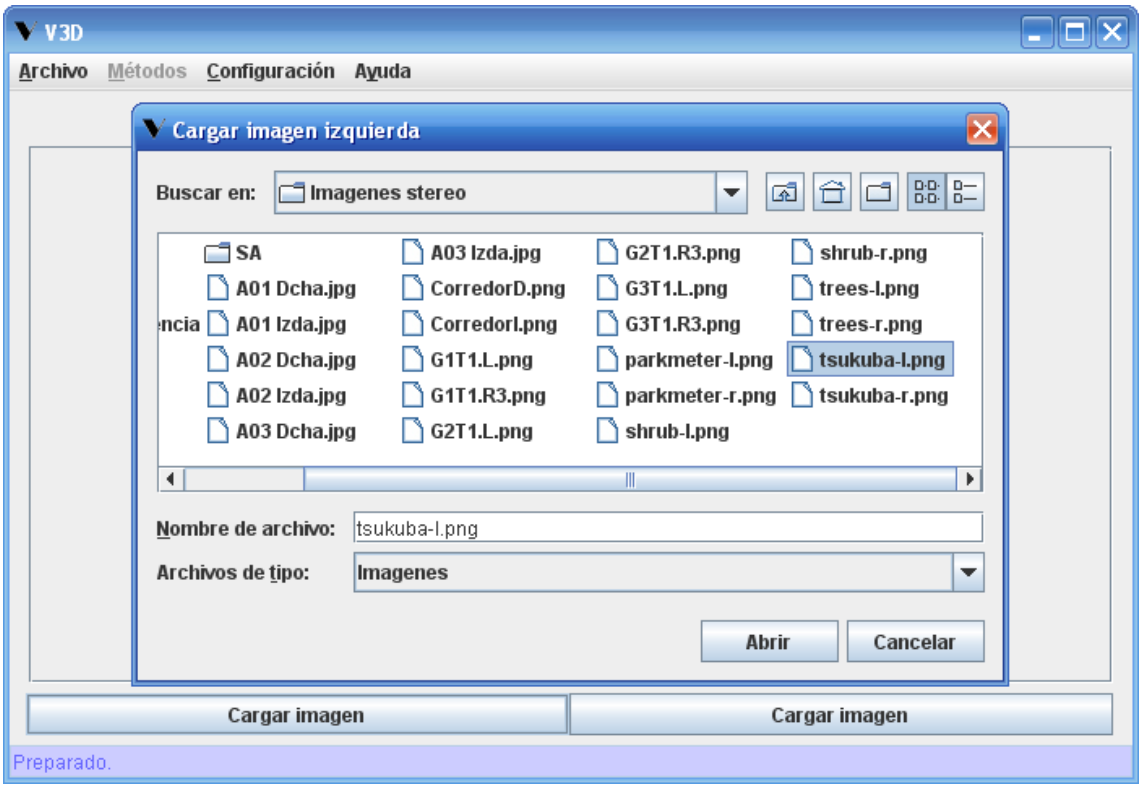

Figura 7.2. Carga de una imagen.

En la figura 7.3 podemos ver el resultado una vez cargada la imagen izquierda.

Figura 7.3. Imagen cargada.

Análogamente cargamos la imagen derecha, tras lo cual podemos observar que el menú de ejecución de métodos está activo, dejándonos seleccionar los algoritmos para ejecutar.

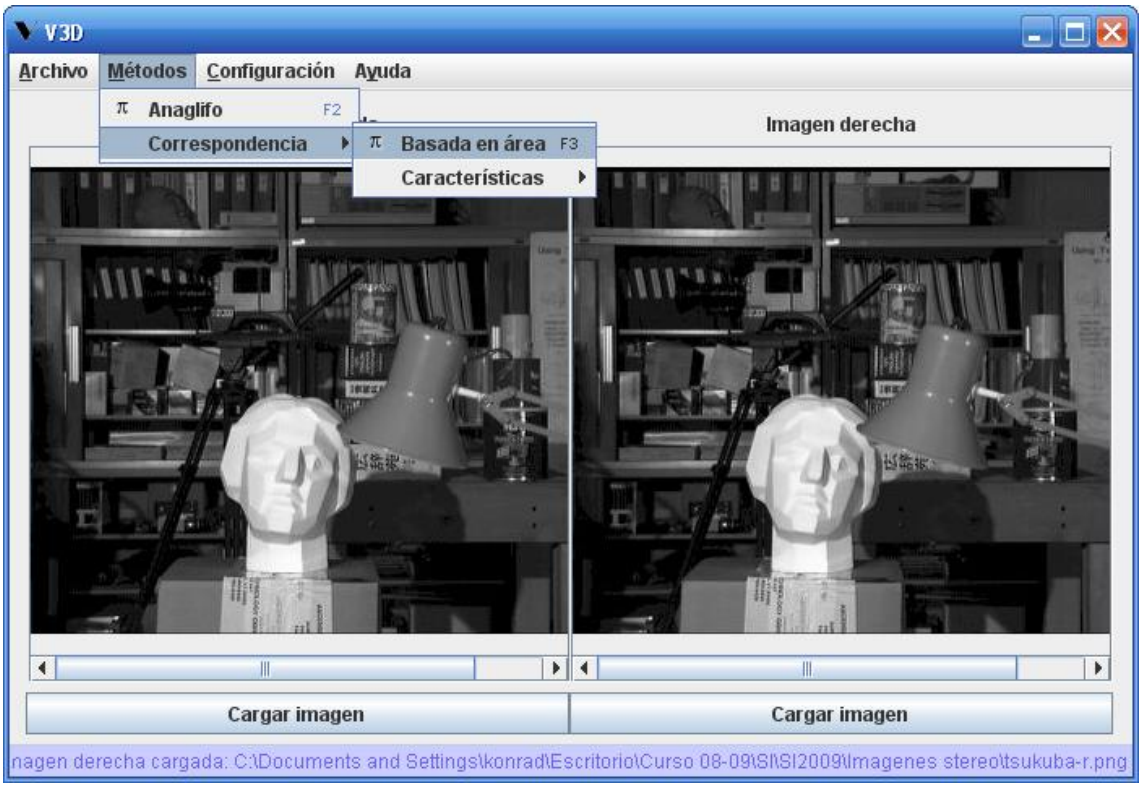

Figura 7.4. Ejecución de un método.

Al elegir uno de los algoritmos a ejecutar (por ejemplo Correspondencia basada en el área) nos saldrá una interfaz de configuración de parámetros del algoritmo (figura 7.5).

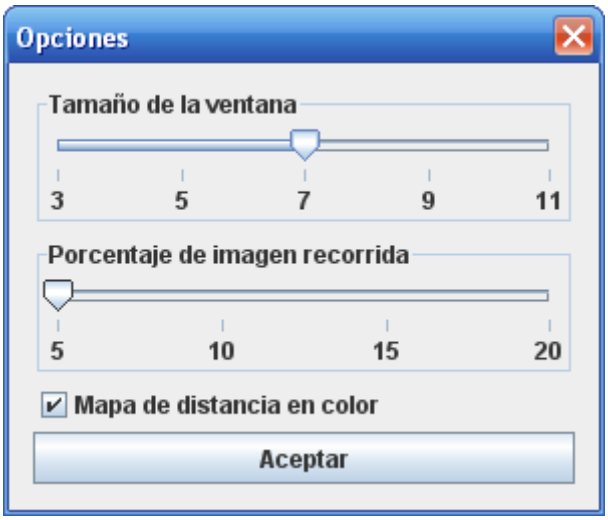

Figura 7.5. Configuración de un método.

Hay que tener en cuenta que se debe tener conocimientos básicos de dichos algoritmos para poder elegir de manera correcta los parámetros deseados. Sin embargo, por defecto vienen fijados unos valores con los que se conseguirá un resultado aceptable para la mayoría de los casos. Elegidos los parámetros adecuados se selecciona Aceptar, pasando seguidamente a las ventanas de resultado.

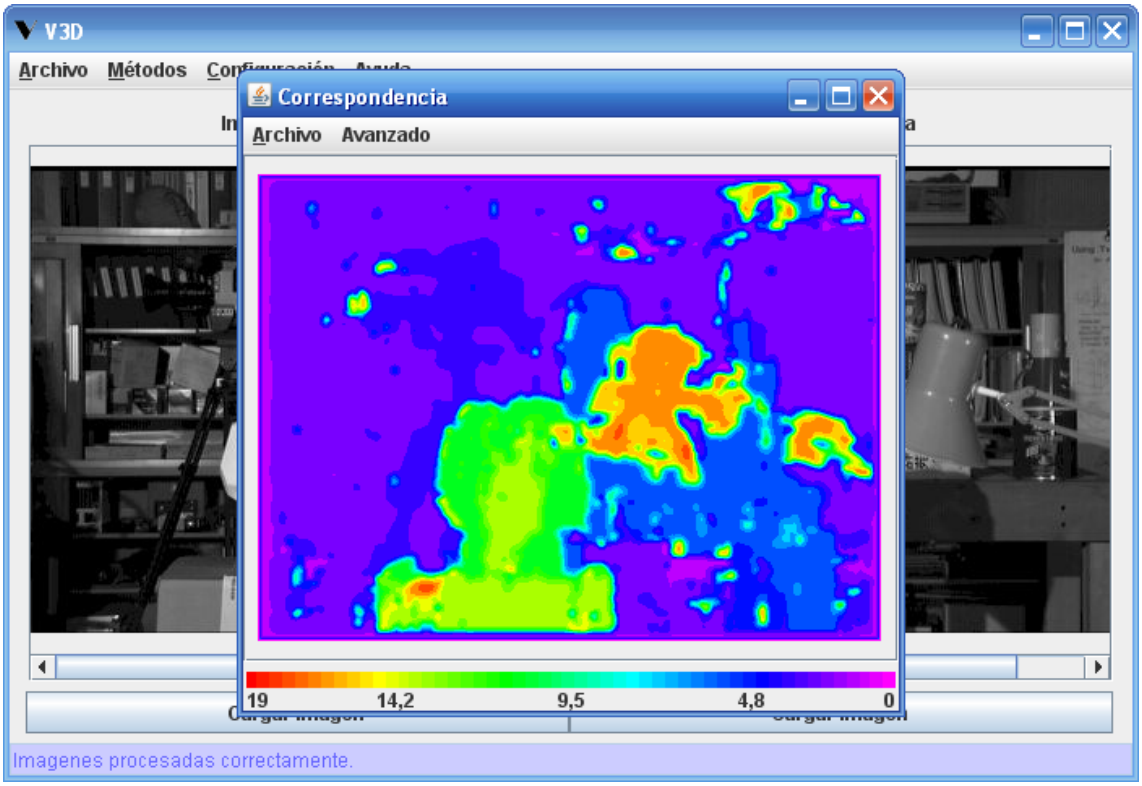

Figura 7.6. Resultado de un método.

Como podemos observar en la figura 7.7 la ventana de resultado nos proporciona un menú de opciones a través del cual podemos realizar lo siguiente:

- El menú Archivo nos permite mostrar la escala de colores o grises de la imagen, además de poder guardarla en disco para futuras comparaciones con otros resultados.
- El menú Avanzado nos ofrece la posibilidad de ejecutar algoritmos de mejora sobre el resultado obtenido. En este caso, al tratarse de un algoritmo de correspondencia basada en el área, nos permite elegir uno de los dos métodos disponibles.

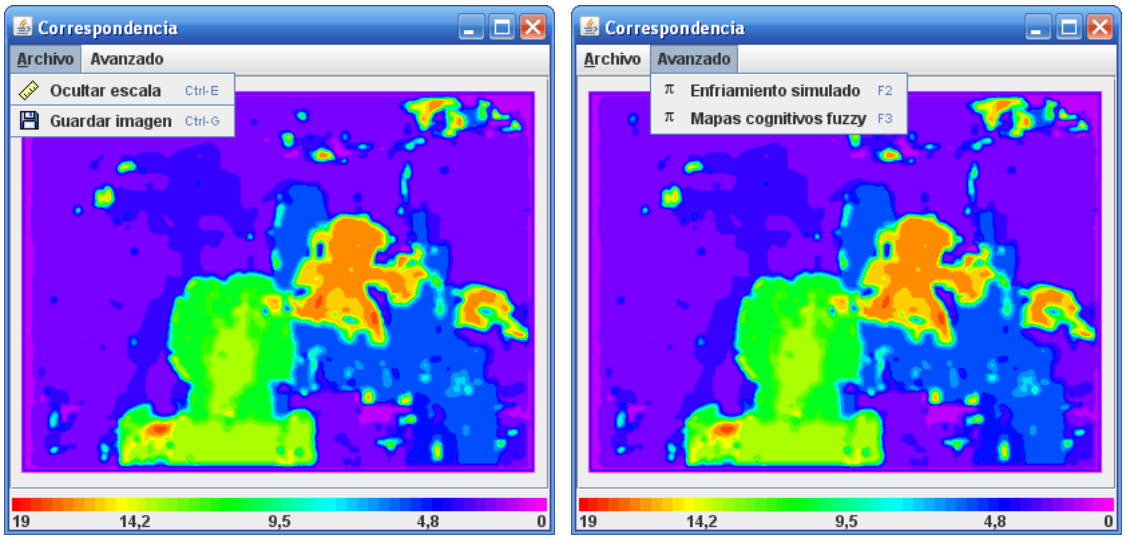

Figura 7.7. Opciones del resultado.

7.1.4 Cambiar configuración de la aplicación

Como hemos comentado anteriormente, en el menú principal de la aplicación existe un menú de configuración.

Figura 7.8. Configuración de la aplicación.

Este menú de configuración nos permite:

- Cambiar el número de hilos que harán uso los algoritmos durante su ejecución.

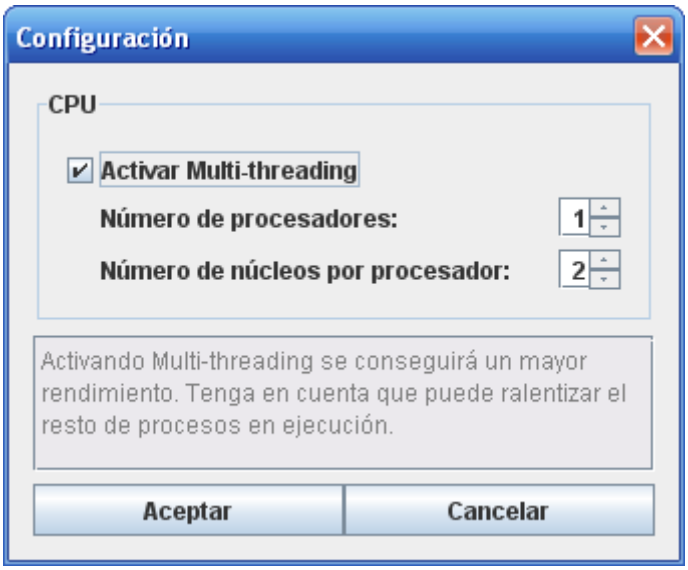

Figura 7.9. Configuración de las unidades de proceso.

- Cambiar el tema de la aplicación a la estándar de Java o a la que haya en el sistema operativo donde se ejecuta la aplicación para no perder la estética.

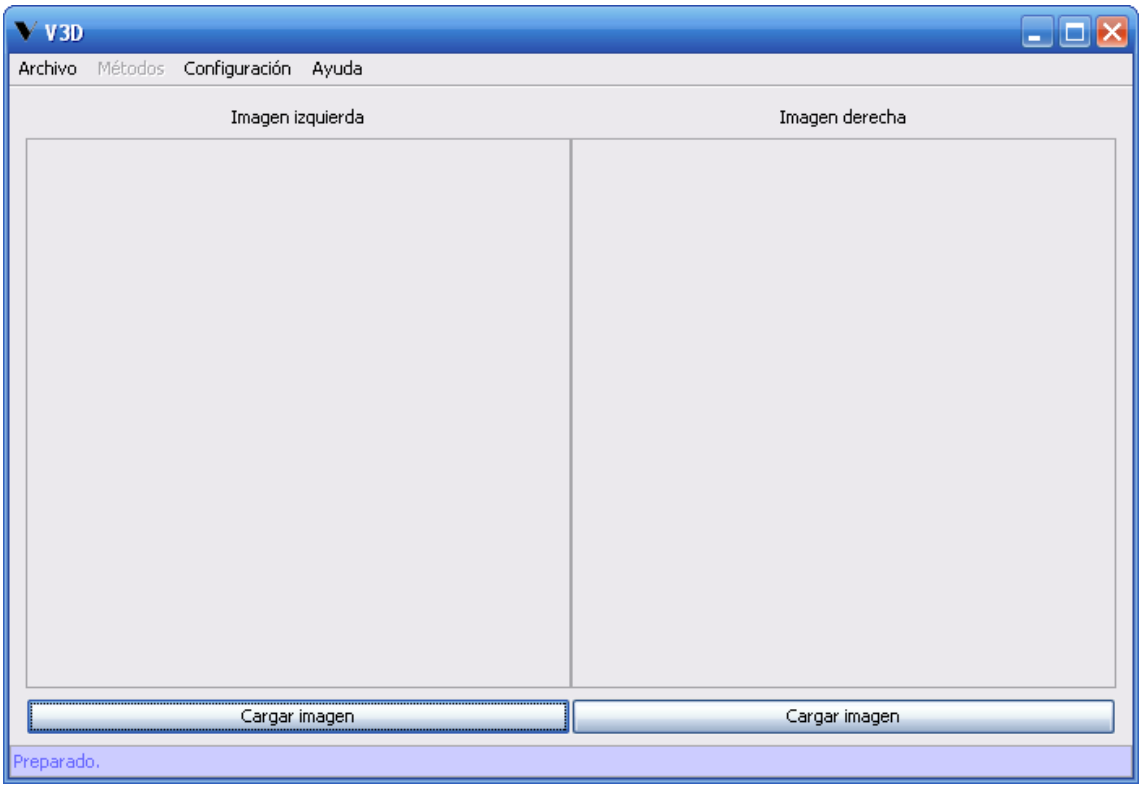

Figura 7.10. V3D en Windows.

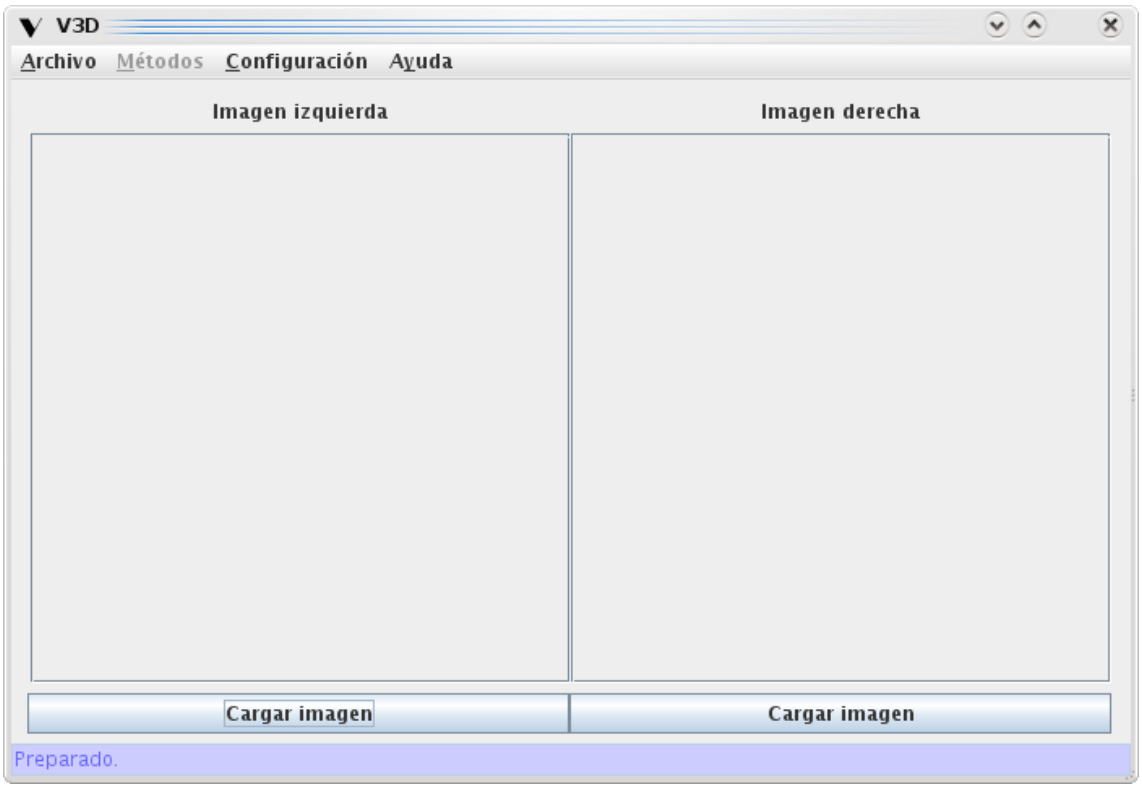

Figura 7.11. V3D en Linux.

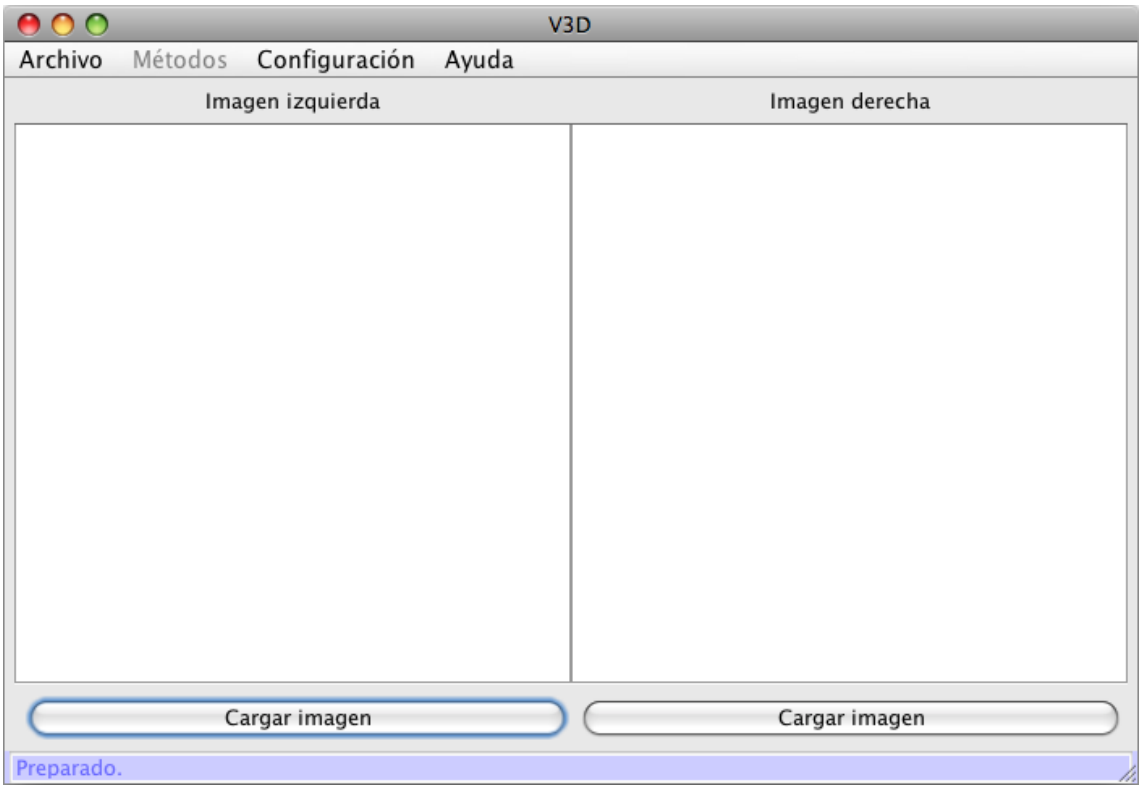

Figura 7.12. V3D en Mac.