APLICACIONES DEL OPERADOR SIFT AL RECONOCIMIENTO DE OBJETOS

Universidad Carlos III de Madrid

Departamento de Automática

Escrito por Abel Alguacil Gómez

Tutor: *Arturo de la Escalera Hueso*

9 de diciembre de 2009

Dedicado a todas las personas que me han dado fuerza todos estos años y especialmente a mis padres y mis hermanos

Índice general

Índice de figuras

Capítulo 1

Introducción

En el mundo industrial, las herramientas de visión cada vez cobran mayor protagonismo. Cada vez se diseñan más sistemas que sean capaces de reconocer objetos y se está investigando en sistemas de visión estéreo para poder realizar planos 3D de habitaciones y para cálculos de distancias.

En todos estos casos es necesario realizar una correspondencia de objetos entre las distintas imágenes. Este tipo de correspondencia se puede hacer de múltiples formas, pero en la mayoría de los casos hay que tener excesivamente controlada la iluminación, la orientación de las cámaras y su respectiva calibración para que el algoritmo de *matching* sea lo más sencillo posible, y poderlo implementar de un modo fácil y económico.

En numerosas aplicaciones estos problemas no son excesivamente impor-

tantes, ya que el sistema de visión se encuentra en ambientes interiores y con cámaras fijas, en los cuales se pueden tener controladas todas estas variables. Pero en otras muchas aplicaciones, la iluminación no está controlada por tratarse de un ambiente exterior, o la calibración de las cámaras no se puede asegurar ya que las cámaras se encuentran en movimiento. En estos casos la correspondencia de imágenes se convierte en una ardua tarea, la cual en numerosas ocasiones debe realizarse en tiempo real y obteniendo resultados fiables.

Por lo tanto, el objetivo de este proyecto es la implementación de un algoritmo de correspondencia de imágenes, el cual sea inmune, en la medida que se pueda, a la rotación, a la escala, a la iluminación y a ser posible al punto de vista de la cámara. Para ello, basándose en los estudios de Krystian Mikolajczyk y Cordelia Schmid [5] en los cuales se compara el alcance de distintos descriptores que hallan regiones de interés de imágenes usando propiedades como intensidad de píxel, color textura, bordes, etc. Listan una serie de descriptores cuya comparativa se muestra en la *Figura 1.1.*

En los resultados experimentales que se muestran en la *Figura 1.1*, realizados por Mikolajczyk y Schmid, se consideran las transformaciones geométricas (escala, rotación y traslación), manchas, compresión de imágenes Jpeg y cambios de iluminación.

Descriptor	recall	1-precision	#nearest neighbor correct matches
GLOH	0.25	0.52	192
SIFT	0.24	0.56	177
Shape context	0.22	0.59	166
PCA-SIFT	0.19	0.65	139
Moments	0.18	0.67	133
Cross correlation	0.15	0.72	113
Steerable filters	0.12	0.78	90
Spin images	0.09	0.84	64
Differential invariants	0.07	0.87	54
Complex filters	0.06	0.89	44

Recall, 1-precision, and number of correct matches obtained with different descriptors for a fixed number of 400 nearest neighbor matches on the image pair displayed in Fig. 13. The regions are detected with Hessian-Affine.

Figura 1.1: *Resultados obtenidos por el estudio de Mikolajczyk y Schmid [5].*

Se puede observar que el descriptor SIFT (Scale Invariant Feature Transform) obtiene una gran cantidad de correspondencias correctas, sólo superado por el método GLOH (Gradient Location and Orientation Histogram). Lo único que el método GLOH es mucho más costoso, produciendo múltiples resultados.

Debido a esta razón, y a que el descriptor SIFT es más eficiente que el método GLOH, se ha pensado en la implementación de un descriptor SIFT, junto con un algoritmo de *matching*, el cual sea totalmente invariante a la escala y rotación, además de ser parcialmente inmune a la iluminación como al punto de vista de la cámara.

Capítulo 2

Descriptor SIFT

En este proyecto, tal y como se ha indicado anteriormente, se pretende implementar un descriptor SIFT para la correspondencia de imágenes. Este descriptor fue descrito por *David G. Lowe* de la universidad de la Columbia Británica en Vancouver, en el artículo[3]. En ese artículo se describe un método para obtener características de imágenes que sean invariantes tanto a escala como rotación y parcialmente inmune a la iluminación como al punto de vista de la cámara. Además dichas características se obtienen tanto en el dominio espacial como en el frecuencial, reduciendo así la probabilidad de disrupción por oclusión o ruido.

Este método consta de cuatro pasos, los cuales son:

1. Detección de máximos y mínimos espacio-escala: El primer paso es la búsqueda de puntos en la imagen que puedan ser *keypoints*. Se reali-

za usando diferencias de funciones gausianas para identificar puntos interesantes que sean invariantes a la escala y a la orientación.

2. Localización de los *keypoints*: De los puntos obtenidos en el apartado anterior se determinan la localización y la escala de los mismos, de los cuales se seleccionan los *keypoints* basándose en la medida de la estabilidad de los mismos.

3. Asignación de la orientación: A cada localización del *keypoint* se le asigna una o más orientaciones, basado en las orientaciones de los gradientes locales de la imagen.

4. Descriptores de los *keypoints*: Los gradientes locales se miden y se transforman en una representación que permite importantes niveles de la distorsión de la forma local y el cambio en la iluminación.

El aspecto esencial de este método se corresponde al número de *keypoints* que se puede generar, ya que si de cada imagen obtenemos muy pocos *keypoints*, a la hora de realizar el *matching* se dispondrán de muy pocos puntos candidatos que puedan coincidir en ambas imágenes. Pero éste no es el caso de este método, ya que según el artículo [3], se indica que de una imagen normal de 500×500 píxeles se pueden llegar a obtener alrededor de 2000 puntos característicos estables. Como este dato es una suposición teórica, para comprobar el alcance de dicho método, se ha realizado una prueba con

un ejecutable implementado por el propio profesor *Lowe*, el cual obtiene los *keypoints* de una imagen. Se hizo una prueba con la figura mostrada a continuación, cuya resolución es de 640×480 píxeles. Con esta prueba se pudo comprobar que dicha suposición es bastante certera, ya que se obtuvieron 1877 *keypoints*. De todos modos, tal y como se indica en el artículo [3], el número de *keypoints* obtenidos varía dependiendo del número de objetos que haya en la imagen y el número de aristas de estos objetos.

Figura 2.1: *Imagen obtenida de la biblioteca de la UC3M del campus de Leganés a una resolución de* 640×480 *píxeles.*

2.1. Detección de máximos y mínimos espacio-escala

Tal y como se ha indicado anteriormente, la primera etapa del descriptor SIFT es la detección de puntos que puedan ser *keypoints*. Para ello, hay que identificar las localizaciones y las escalas en los objetos que se puedan repetir desde diferentes puntos de vista del mismo objeto. Para ello hay que detectar localizaciones en la imagen, buscando características que sean estables en todas las escalas posibles. Una vez se han dado estas razones, el único núcleo de espacio-escala posible es la función gausiana.

Por lo tanto, se define una función espacio-escala $L(x, y, \sigma)$ de una imagen como la convolución de una gausiana de una variable escalar, $G(x, y, \sigma)$, con la imagen, $I(x, y)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.1}$$

donde la función gausiana queda definida como:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2.2}$$

Para el cálculo eficiente de la localización de los *keypoints* estables, tanto en el espacio como en escala, se propone usar los máximos y los mínimos de la función $D(x, y, \sigma)$ que es la diferencia de funciones gausianas convolucionadas con la imagen. La diferencia de gausianas se puede obtener de dos escalas próximas separadas por una constante multiplicativa k:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

$$= L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.3}$$

El autor de dicho método indica que existen varios motivos para escoger esta función para obtener los posibles *keypoints*. El primero de ello se debe a la eficiencia de dicha función para ser computada.

La segunda razón se debe a que la diferencia de gausianas es una buena aproximación de la laplaciana normalizada, $\sigma^2\nabla^2 G$. Esto es muy importante, ya que la función laplaciana normalizada por el factor σ^2 es el requerimiento esencial para obtener una invarianza en escala. Además, también se puede demostrar, que los máximos y los mínimos de la función laplaciana normalizada producen las características más estables de una imagen, en comparación con otras funciones como pueden ser el gradiente, la función hessiana o la función detectora de esquinas de Harris.

La relación entre la función D y $\sigma^2\nabla^2 G$ puede entenderse como la ecuación de la difusión de calor, donde el parámetro temporal es σ^2 ($t = \sigma^2$):

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G \tag{2.4}$$

Se puede observar que $\nabla^2 G$ puede ser implementado como una diferencia finita, usando como escalas cercanas $k\sigma$ y σ:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{2.5}$$

Entonces,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G \tag{2.6}$$

Esto muestra que la función diferencia de gausianas difiere de la laplaciana normalizada sólo por el factor $(k-1)$. Este factor en la ecuación solamente es una constante sobre todas las escalas, por lo tanto no influencia en el cálculo de los máximos y los mínimos.

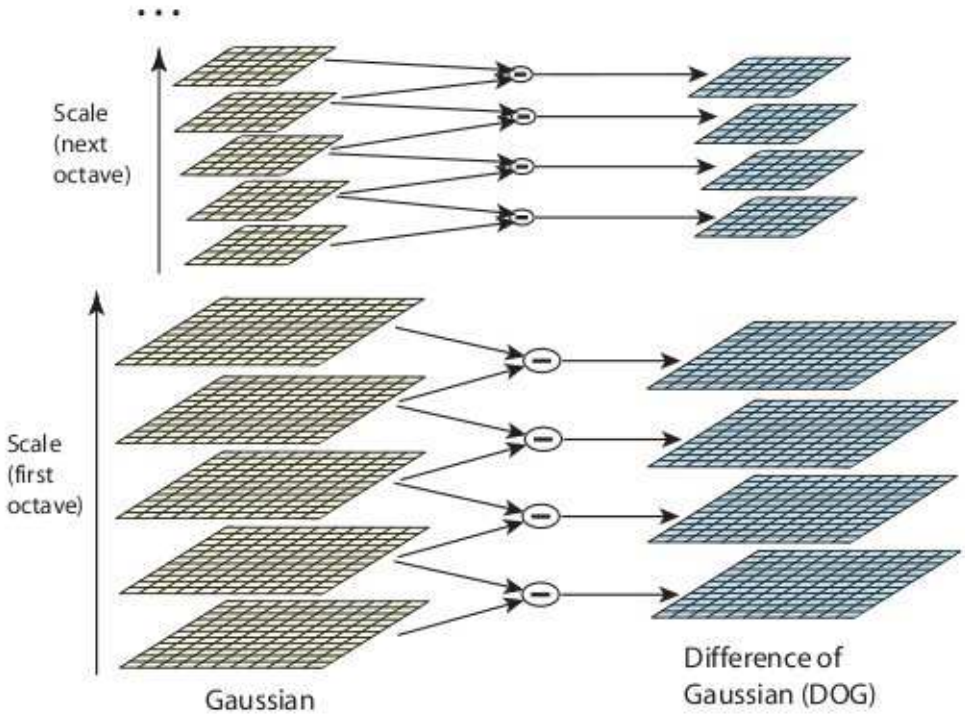

Figura 2.2: *Método de obtención de la imagen diferencia de gausianas.*

Una forma eficiente para la construcción de la función $D(x, y, \sigma)$ es la que se describe en la *Figura 2.2*. La imagen inicial es de forma incremental con-

volucionada con gausianas para producir imágenes separadas por un factor k constante en la escala, que se muestran apiladas en la columna de la izquierda. Cada imagen que forma la imagen inicial se denomina octava. Se divide cada octava de la escala un número entero s de intervalos, ya que $k = 2^{1/s}$. Después se tiene que producir $s + 3$ imágenes por octava en el grupo de imágenes borrosas por efecto de la función gausiana, de modo que la detección de máximos y mínimos final cubre una octava completa. A continuación se restan las imágenes adyacentes para obtener la imagen diferencia de gausianas. Una vez que se ha terminado de procesar una octava completa, hay que volver a muestrear las imágenes, pero esta vez con un valor de σ el doble del inicial, tomando ahora el segundo píxel en cada fila y en cada columna. La precisión de muestreo en relación con σ no es diferente que para el inicio de la anterior octava, pero el cálculo se reduce considerablemente.

Figura 2.3: *Niveles de imágenes borrosas en una misma octava.*

Se puede observar en la *Figura 2.3* cómo son las distintas imágenes en una misma octava. En este caso se muestran todos los niveles de una octava com-

pleta. Se ve, que al aumentar el nivel en una misma octava, las imágenes son cada vez más borrosas. Precisamente con estas imágenes son con las que se obtienen las imágenes diferencias de gausianas.

Para detectar los máximos y los mínimos locales, no sólo se escoge el píxel que es mayor o menor que sus ocho vecinos en la imagen. También hay que seleccionar el máximo y el mínimo entre las imágenes superior e inferior de la columna, tal y como se muestra en la *Figura 2.4*. El coste computacional para obtener estos máximos y mínimos es relativamente bajo, debido a que la mayoría de los puntos muestreado son eliminados en las primeras comprobaciones.

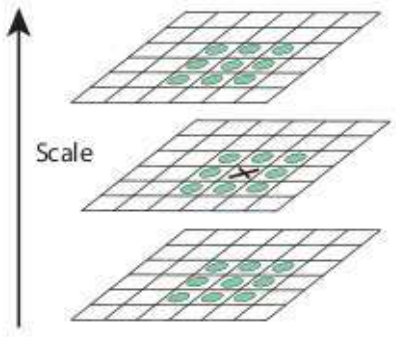

Figura 2.4: *Visualización de los puntos vecinos que se tienen en cuenta para el cálculo de los máximos y mínimos.*

Un aspecto importante es la selección de la frecuencia de muestreo tanto en

escala como en espacio. Desafortunadamente, la selección de una frecuencia de muestreo espacial muy pequeña no asegura la detección de todos los máximos y mínimos existentes, ya que dichos puntos críticos pueden estar extremadamente juntos de una forma arbitraria. Pero se puede comprobar experimentalmente que los puntos críticos que se encuentran muy cercanos son puntos inestables, por lo tanto no deberían ser seleccionados ya que darían *keypoints* erróneos. Por lo tanto, si se selecciona un número de escalas por octava muy elevado, se encontrarían un número muy elevado de *keypoints*, pero al ser estos más inestables, la repetitividad de los mismos será menor, tal y como se muestra en las siguientes gráficas.

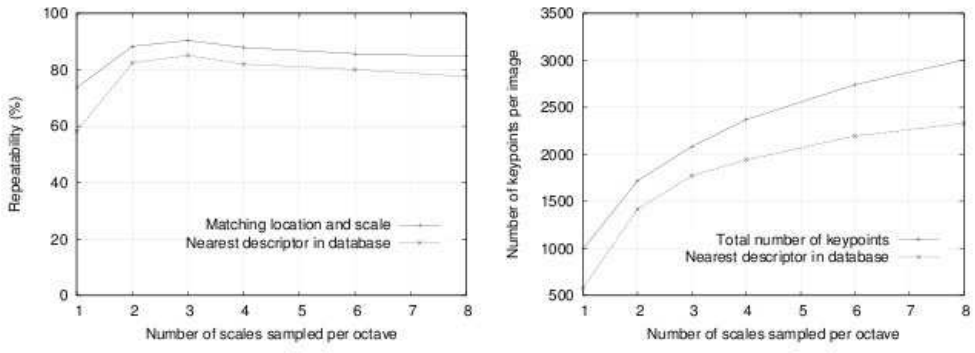

Figura 2.5: *Porcentaje de repetitividad de los keypoints encontrados (izquierda) y el número total de keypoints (derecha) respecto al número de escalas por octava.*

Esto no contradice a lo dicho anteriormente, que para realizar una buena

identificación de objetos se requiere una gran cantidad de puntos característicos. El problema es que si se tienen más de la cuenta, estos puntos puede que no sean significativos del objeto, y por lo tanto, se obtendrán puntos erróneos, pero también más puntos correctos. Por lo tanto, dependiendo de la aplicación para la cual se quiera usar, habrá que parametrizar el descriptor a un valor de escalas por octava o a otro, y también dependerá si después del descriptor se puede implementar un algoritmo que elimine los puntos erróneos, ya sea mediante puntos paralelos o por agrupación de puntos (por ejemplo el algoritmo max-min).

2.2. Localización de los keypoints

Una vez se han encontrado los posibles *keypoints*, el siguiente paso es llevar a cabo un ajuste detallado de la ubicación, la escala y la proporción de las principales curvaturas cercanas a los datos. Esta información permite rechazar puntos que no son estables debido a su bajo contraste o a una pobre localización a través de un borde.

La primera implementación que se puede realizar para este enfoque es simplemente localizar los *keypoints* a la ubicación y escala del punto central del punto de muestreo. En cambio, existe otro método mediante un ajuste a una función cuadrática, la cual mejora notablemente los resultados en cuanto a

coincidencia y estabilidad en los puntos. Dicha función es el desarrollo de Taylor de la función diferencia de gausianas:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.7}$$

donde D y sus derivadas son evaluadas en el punto de muestreo y $x = (x, y, \sigma)^T$ es el offset de dicho punto. La ubicación del punto crítico, $\widehat{x}$, es determinado mediante la derivada de la función anterior respecto a x e igualando dicha ecuación a cero, se obtiene entonces:

$$\widehat{x} = -\frac{\partial^2 D}{\partial x^2}^{-1}\frac{\partial D}{\partial x} \tag{2.8}$$

Se puede demostrar que la hessiana y la derivada de D se aproximan por el uso de diferencias de puntos de muestreo más cercanos. De este modo, el resultado del sistema lineal 3×3 puede ser resuelto con un mínimo coste. Si el offset de $\widehat{x}$ es mayor que 0.5 en cualquier dimensión, entonces significa que dicho punto crítico está muy cerca de otro punto de muestreo. Entonces, en ese caso, el punto de muestreo se cambia y se realiza la interpolación alrededor de dicho punto.

El valor de la función en el punto crítico, $D(\widehat{x})$, es útil para rechazar los posibles candidatos que sean inestables con bajo contraste. Dicho valor se obtiene utilizando las dos ecuaciones anteriores, por lo tanto:

$$D(\widehat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\widehat{x} \tag{2.9}$$

Según se indica en el artículo [3], también se deben descartar aquellos puntos cuyo valor de $|D(\widehat{x})|$ sea menor de 0.03, siempre que se asuma valores de los píxeles en el rango $[0, 1]$.

Con todo lo descrito hasta ahora se han descartado todos aquellos puntos con bajo contraste. Pero también se deben descartar aquellos puntos que posean una pobre localización a lo largo de un borde, ya que la función diferencia de gausianas posee una alta respuesta a lo largo de los bordes.

Un pico pobremente definido en la función diferencia de gausianas indica que habrá una larga curvatura principal en la dirección del borde, pero pequeña en la dirección perpendicular del mismo. La curvatura principal se puede procesar a partir de una matriz hessiana de 2×2, evaluada en el *keypoint*.

$$H = \begin{pmatrix} D_{x,x} & D_{x,y} \\ D_{x,y} & D_{y,y} \end{pmatrix} \tag{2.10}$$

Las derivadas necesarias para obtener la matriz H se han calculado tomando diferencias con los vecinos del punto de muestreo.

Los autovalores de la matriz H son proporcionales a las curvaturas principales de D. Se toma α como el mayor de los autovalores y β como el menor. Entonces, se puede obtener la suma de los autovalores a partir de la traza

de H y su producto del determinante:

$$Tr(H) = D_{x,x} + D_{y,y} = \alpha + \beta \tag{2.11}$$

$$Det(H) = D_{x,x}D_{y,y} - (D_{x,y})^2 = \alpha\beta \tag{2.12}$$

Si las curvaturas tienen signos diferentes, entonces dicho punto se debe descartar, ya que no está representando a un máximo o a un mínimo. Entonces, si se toma r como la relación que existe entre los dos autovalores ($\alpha = r\beta$), se obtiene:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{2.13}$$

que sólo depende de la relación que existe entre los autovalores. El valor $(r+1)^2/r$ es mínimo cuando los dos autovalores son idénticos y aumenta conforme va creciendo r. Entonces para verificar la relación entre las curvaturas sólo es necesario calcular la relación existente entre el cuadrado de la traza de H y su determinante.

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{2.14}$$

Según se indica en el artículo [3], un valor umbral bastante razonable, sería tomar $r = 10$. Así, de este modo, descartamos los puntos inestables por una pobre localización en un borde.

2.3. Asignación de orientaciones

La asignación de una orientación a los *keypoints* es muy importante, ya que si se consigue una orientación coherente basada en las propiedades locales de la imagen, el descriptor puede ser representado en relación de dicha orientación y por lo tanto ser invariante a la rotación. Este enfoque contrasta con la de otros descriptores invariantes a la orientación, que buscan propiedades de las imágenes basadas en medidas invariantes a la rotación. La desventaja de este enfoque es que limita el número de descriptores que se pueden usar y rechaza mucha información de la imagen.

Para establecer una orientación adecuada, este método se basa en el gradiente local de la imagen alrededor de los *keypoints*. Para ello se utilizará la imagen suavizada por la gausiana a la mayor escala determinada por el *keypoint*, de modo que todos los cálculos se realizan de un modo invariable a la escala. Por cada imagen de muestreo, $L(x, y)$, la magnitud del gradiente, $m(x, y)$, y la orientación, $\theta(x, y)$, se precalculan usando diferencias de píxeles:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.15}$$

$$\theta(x,y) = tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \tag{2.16}$$

Un histograma de orientación se forma a partir de las orientaciones de los

gradientes de los puntos de muestreo que se encuentran dentro de la región que rodea al *keypoint*. El histograma de orientación posee 36 divisiones que abarcan los 360º del rango de orientaciones. Cada muestra añadida al histograma es pesada por su magnitud del gradiente y por una máscara gausiana circular con un valor de σ 1.5 veces el valor que posea el *keypoint*.

Los picos en el histograma de orientaciones corresponden a las direcciones dominantes de los gradientes locales. Se detecta el mayor pico del histograma, y entonces, si existen otros máximos superiores al 80 % del pico principal, éstos se utilizarán para crear otros *keypoints* con nuevas orientaciones. Por lo tanto, existirán varios *keypoints* con la misma localización, pero con diferentes orientaciones. Solamente el 15 % de los *keypoints* disponen de orientaciones múltiples, pero estos puntos contribuyen significativamente en la estabilidad del método. Por último, para determinar con mayor exactitud la localización del pico en el histograma, éste se interpola con una parábola que es fijada por los tres puntos más cercanos a cada pico.

2.4. Descriptores de los keypoints

Una vez que ya se tienen todos los *keypoints* de la imagen, se tiene que segmentar la vecindad del *keypoint* en 4×4 regiones de 4×4 píxeles. Una vez que ya se ha dividido la vecindad del *keypoint*, se genera un histograma

de orientación de gradiente para cada región. Para ello, se utiliza una ponderación gausiana con un ancho $\sigma = 4$ píxeles.

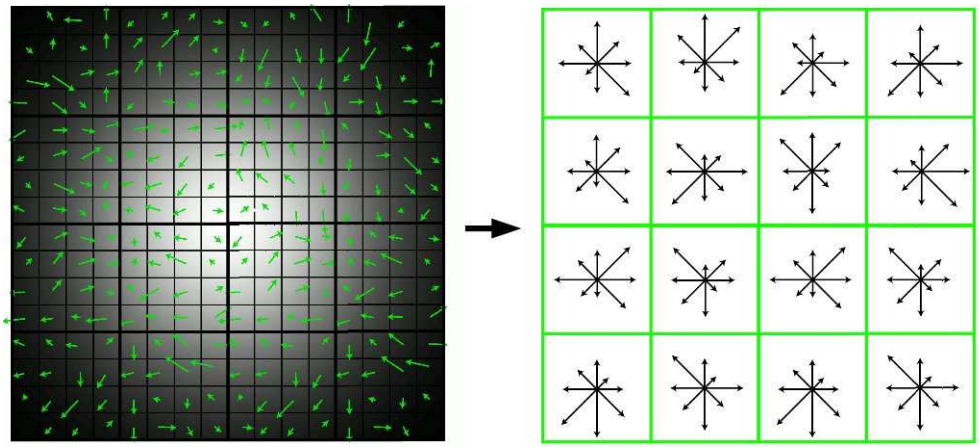

Figura 2.6: *Segmentación de la vecindad de los keypoints y los histogramas de orientación de cada región.*

En la *Figura 2.6* se representa cómo se haría la construcción de los diferentes histogramas de orientación. Pero dicha construcción presenta un gran problema, ya que en el caso de un pequeño desplazamiento espacial, la contribución de un píxel puede pasar de una casilla a otra, lo que provoca cambios repentinos del descriptor. Este desplazamiento también puede deberse al hecho de una pequeña rotación.

Para evitar estos problemas, todos los píxeles contribuyen a todos los vecinos, tanto en magnitud como en orientación. Esta contribución se multiplica

por un peso $1 - d$, donde d es la distancia al centro de la casilla. Esta distancia está normalizada al tamaño de una región, es decir, la longitud de una región de 4 píxeles tiene una distancia $d = 1$. Además, el valor mínimo del peso $1 - d$ es de 0, ya que los pesos no pueden ser negativos.

Como los histogramas de orientación de cada región están divididos en 8 barras, por cada vecindad del *keypoint* se puede construir un histograma tridimensional de $4 \times 4 \times 8$ valores, formando así un vector con todos estos valores.

Dicho vector es normalizado en su magnitud, para así poder eliminar los cambios de contraste. Los cambios de brillo ya son eliminados por el uso del gradiente. Pero el efecto de éste puede ser muy fuerte. Para evitar este problema, el valor de cada componente se limita a un máximo de 0,2. Una vez limitados los valores de los componentes del vector, se vuelve a normalizarlo. Con esto se tiene en cuenta más a la orientación del gradiente que a su magnitud.

Capítulo 3

Algoritmo de Matching

Una vez que ya se tienen calculados los descriptores, sólo queda realizar el *matching* entre los puntos de las distintas imágenes. Para ello, el profesor *Lowe* propone el método de *el vecino más próximo*. Este método está basado en la distancia euclídea entre vectores.

El algoritmo consiste en calcular los descriptores de una imagen A, en la cual se quiere encontrar un objeto que está definido en una imagen B. Cuando ya se tienen los vectores de los descriptores de todos los *keypoints*, por cada *keypoint* de la imagen A se calcula la distancia que existe entre su vector descriptor y todos los demás vectores de los puntos característicos de la imagen B.

Una vez que se tienen calculadas todas las distancias por cada *keypoint* de

la imagen A, se calcula la relación entre las dos distancias menores. Si esta relación es menor que un valor umbral, existe un correspondencia entre el *keypoint* de la imagen A y el *keypoint* más cercano de la imagen B.

Para escoger el valor umbral, se debe tener en cuenta que es más probable obtener un falso negativo que un falso positivo, por lo tanto, se puede seleccionar un valor umbral alto con la seguridad de obtener un número de falsos positivos bastante bajo. Para concretar el número de falsos positivo y falsos negativos que se pueden obtener, se han realizado diversos cálculos, siendo resumidos en la *Figura 3.1.*

En dicho gráfico se puede observar que para un valor de relación de 0.8, se han aceptado como puntos positivos más del 95 % de los valores correctos. Pero para dicho valor también se han aceptado el 10 % de los valores erróneos.

De todos modos, para mejorar la eficiencia de la búsqueda de objetos, se recomienda que se use la transformada de Hough generalizada. Este algoritmo se aplica a los *keypoints* del objeto que se quiere encontrar, obteniendo así la forma que generan dichos puntos en la imagen. Cuando se pretende buscar dicho objeto en otra imagen, a los puntos característicos que tengan una respuesta positiva del algoritmo de correspondencia del vecino más próximo,

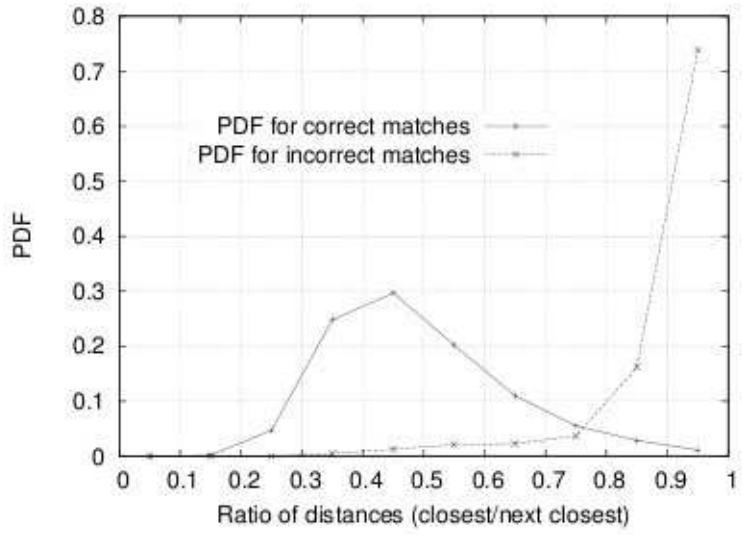

Figura 3.1: *Función de distribución de probabilidad respecto al valor umbral de rechazo del matching, tanto para valores correctos como para valores incorrectos.*

se les somete a un test. Si dichos puntos describen la misma forma que la descrita por la transformada de Hough del objeto a buscar, se interrumpe el proceso de *matching*, ya que el objeto ha sido encontrado satisfactoriamente. Por consiguiente, si no describen la misma forma, se puede dar por finalizada la búsqueda siendo negativo su resultado.

Este método es muy útil para objetos que tengan muchos *keypoints*, ya que con pocos puntos positivos se puede determinar la posición del objeto que se desea encontrar. También es muy útil para encontrar objetos que se encuen-

tran parcialmente ocultos. Además, si en lugar de buscar siempre un objeto de una imagen en otra, se dispone de una base de datos con los descriptores de numerosos objetos, dicho algoritmo reduce notablemente el tiempo de cálculo, ya que no es necesario procesar todos los *keypoints* de los objetos para establecer la búsqueda como positiva o negativa.

Capítulo 4

Comparativa de resultados

A pesar de que, como se ha comprobado hasta el momento, el descriptor SIFT está completamente definido y de que existe un ejecutable programado por el profesor *Lowe* a disposición de quien quiera usarlo, el código de implementación no se encuentra a disposición pública. Además, en dicho ejecutable sólo se encuentra el descriptor, pero no contiene el algoritmo necesario para realizar el *matching* de los puntos de las imágenes. Por lo tanto, para realizar el descriptor, se han utilizado unas bibliotecas programadas por *Andrea Vedaldi* para su tesis en la Universidad de California [6], en las cuales se encuentran las funciones necesarias para poder obtener los *keypoints* de esa imagen con todas las características obtenidas de cada punto.

Dichas funciones se pueden usar en este proyecto, ya que no hay ninguna

restricción por parte del programador para su uso no comercial. Por lo tanto se han usado las funciones necesarias para implementar los algoritmos de cálculo de *keypoints* y sus descriptores.

A parte de implementar un código que calcule los descriptores de los distintos *keypoints* de las imágenes que se quieren tratar, también se han desarrollado unas funciones que realizan el *matching* de los puntos de las distintas imágenes. El método escogido para realizar dicho *matching* es identificando *el vecino más próximo* de los *keypoints*, pero suprimiendo el bloque en que se calcula la transformada de Hough generalizada, ya que incrementa notablemente el tiempo de operación para imágenes pequeñas.

Debido a que no se sabe el valor de los parámetros del ejecutable programado por *Lowe*, se han realizado diversos tests con distintas imágenes, las cuales suponen un uso distinto del descriptor. Dichas pruebas se han realizado usando el mismo algoritmo de *matching*, para poder así comparar efectivamente el resultado del propio descriptor. Además, también se han ido variando los parámetros del descriptor, según los valores recomendados por el artículo [3], para así ver cuáles son los valores más eficientes para los distintos usos que pueda tener dicho algoritmo. También se ha variado el valor umbral para la correspondencia, según las mismas recomendaciones que se han aplicado para los parámetros del descriptor. Este parámetro del

algoritmo de *matching* toma valores entre 0 y 1. Si dicho umbral es igual a 0, se rechaza la coincidencia de todos los puntos característicos. Mientras que si se iguala ese parámetro a 1, todos los *keypoints* que se encuentren en la imagen a buscar tendrán una correspondencia.

4.1. Búsqueda de patrones

El primer uso que se ha comprobado es la búsqueda de un patrón dentro de una imagen. Esta prueba consiste en buscar un recorte de una imagen en ella misma. Las razones de realizar estas pruebas se deben, a que en un principio no se disponen de valores razonables de los diferentes parámetros que se pueden usar en el descriptor. Es por eso que se han realizado diversas pruebas modificando cada uno de los parámetros posibles, para así poder tener una orientación de cuáles de éstos son significativos y cuáles son los valores razonables de uso.

Figura 4.1: *Patrón usado para realizar las pruebas de búsqueda de patrones.*

El partrón a buscar es el representado por la *Figura 4.1*, que es una imagen de 180×200 píxeles de resolución.

La imagen en la cual se pretende encontrar el patrón anterior es la mostrada por la *Figura 4.2* que es una imagen de 800×640 píxeles de resolución.

Figura 4.2: *Imagen usada para realizar la búsqueda de patrones.*

Se puede observar que la imagen usada tiene muchos bordes y picos. Esto debería ayudar a priori a encontrar muchos puntos característicos y así conseguir que sea más fácil la búsqueda del patrón. De ser así, al disponer de una

gran cantidad de *keypoints*, se debería fijar un valor bajo en el parámetro de rechazo en el proceso de coincidencia de puntos, ya que de este modo no obtendremos una gran cantidad de falsos positivos. Para poder tener una referencia real de los valores umbrales del *matching*, se han realizado pruebas fijando dichos valores a 0.5 (*Figura 4.3*), 06 (*Figura 4.4*), 07 (*Figura 4.5*) y 08 (*Figura 4.6*).

Figura 4.3: *Búsqueda de patrones por el ejecutable de Lowe con un valor umbral de 0.5.*

Se puede observar que el descriptor encuentra una gran cantidad de *keypoints* que tienen correspondencia entre la imagen y el patrón. Lo único, que debido a la gran cantidad de puntos característicos generados por el patrón, se observa que para el caso de que el valor umbral sea 0.8 existe una gran

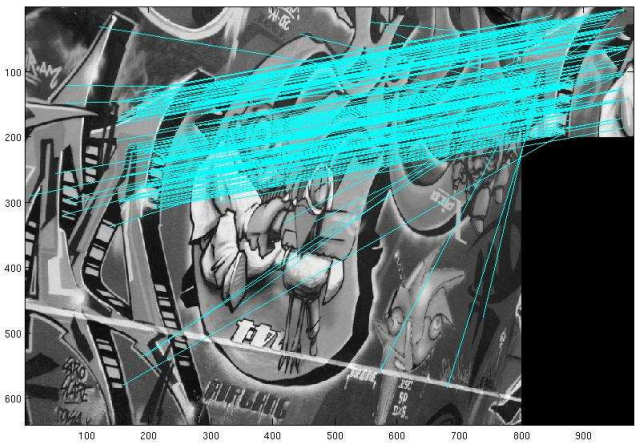

Figura 4.4: *Búsqueda de patrones por el ejecutable de Lowe con un valor umbral de 0.6.*

Figura 4.5: *Búsqueda de patrones por el ejecutable de Lowe con un valor umbral de 0.7.*

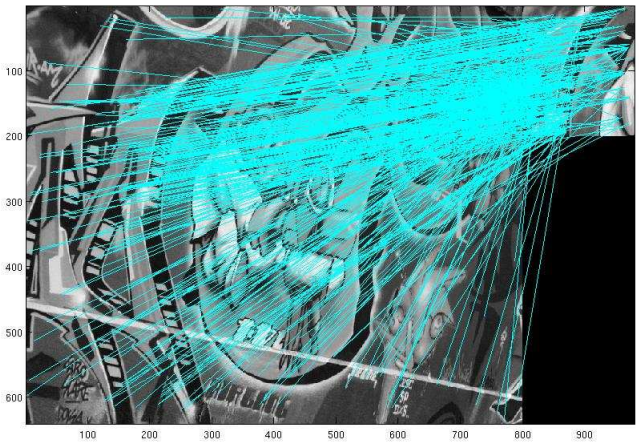

Figura 4.6: *Búsqueda de patrones por el ejecutable de Lowe con un valor umbral de 0.8.*

cantidad de falsos positivos, tal y como se dijo antes. Conforme se va disminuyendo el valor del parámetro de rechazo del algoritmo de coincidencia, se observa que el número de falsos positivos va disminuyendo, llegando incluso a estar por debajo de la decena, en el caso de fijar dicho parámetro a 0.5.

Una vez que se ha comprobado el funcionamiento del ejecutable de *Lowe*, se procederá a mostrar los resultados del software implementado para este proyecto. Para ello se tiene que probar su funcionamiento variando diversos parámetros, como son:

- Número de octavas: Es el número de bloques de imágenes borrosas que genera el descriptor para detectar los candidatos a puntos característi-

cos. Para las pruebas tomará valores de -1, 1, 3 y 5. Que el número de octavas tome un valor negativo no es una incongruencia, ya que, según Andrea Vedaldi [6], este valor sólo es un valor de referencia para saber cual sería el primer valor que deben tomar los diversos parámetros del descriptor.

- Número de niveles por octavas: Es el número de imágenes que contiene cada octava para generar las diferencias de gausianas. Para las pruebas tomará valores de 3, 6, 8 y 10.

- Valor umbral del algoritmo de *matching*: Al igual que se hizo en el caso del ejecutable de *Lowe*, se probará con los valores de 0.5, 0.6, 0.7 y 0.8

El modo en que se deben ajustar estos parámetros se explicará con más detalle en el *Apéndice A*.

A continuación se mostrarán las imágenes resultados de la búsqueda del patrón con un número de octavas igual a 1 y niveles por octava igual a 3. Para ello se muestran los resultados para los valores del parámetro de rechazo de 0.5 (*Figura 4.7*), 0.6 (*Figura 4.8*), 0.7 (*Figura 4.9*) y 0.8 (*Figura 4.10*).

Se puede observar, que en este caso, el número de puntos característicos coincidentes entre la imagen y el patrón es mucho menor que en el caso de usar el ejecutable de *Lowe*. También cabe destacar, que el número de falsos

Figura 4.7: *Búsqueda de patrones por el software implementado en este proyecto con 1 octava, 3 niveles por octava y un valor umbral de 0.5.*

positivos que se obtienen es nulo para los casos de usar un valor de rechazo de 0.5, 0.6 y 0.7, y para el caso de 0.8 sólo existe un falso positivo. Con estos resultados se puede deducir que este algoritmo, con los parámetros de número de octavas y niveles por octavas descrito anteriormente, es bastante preciso en sus cálculos, pero no genera suficientes puntos característicos como para asegurar que se ha realizado satisfactoriamente una búsqueda.

Una vez realizadas las primeras pruebas con el software implementado para este proyecto, para disponer de un primer punto de referencia de su comportamiento, se procederá a comprobar como afecta al resultado final las variaciones del número de octavas. Para ello se fijarán el número de niveles

Figura 4.8: *Búsqueda de patrones por el software implementado en este proyecto con 1 octava, 3 niveles por octava y un valor umbral de 0.6.*

Figura 4.9: *Búsqueda de patrones por el software implementado en este proyecto con 1 octava, 3 niveles por octava y un valor umbral de 0.7.*

Figura 4.10: *Búsqueda de patrones por el software implementado en este proyecto con 1 octava, 3 niveles por octava y un valor umbral de 0.8.*

por octava a 3, y el parámetro de rechazo para la coincidencia de puntos a 0.7. Por lo tanto, el número de octavas serán -1 (*Figura 4.11*), 3 (*Figura 4.12*) y 5 (*Figura 4.13*).

Con estos resultados se puede comprobar que si fijamos el número de octavas a 1, 3 ó 5 los resultados apenas varían. En cambio, si fijamos dicho valor a -1, los resultados obtenidos son muy parecidos a los observados con el ejecutable de *Lowe*, aunque para el mismo valor de rechazo de la coincidencia de puntos, el software implementado para este proyecto obtiene menos puntos. Esto se debe, a que el ejecutable de *Lowe* no tiene implementado un filtro, el cual da al algoritmo cierta inmunidad a los cambios de brillo. Por lo tanto, lo único

Figura 4.11: *Búsqueda de patrones por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.7.*

Figura 4.12: *Búsqueda de patrones por el software implementado en este proyecto con 3 octavas, 3 niveles por octava y un valor umbral de 0.7.*

Figura 4.13: *Búsqueda de patrones por el software implementado en este proyecto con 5 octavas, 3 niveles por octava y un valor umbral de 0.7.*

importante para el número de octavas es que su valor sea negativo o positivo.

Una vez comprobado cómo afecta a los resultados los cambios en el número de octavas, sólo queda observar la importancia de variar los niveles por octava. Para ver esta importancia, se han realizado diversas pruebas fijando el valor de rechazo a 0.7. En primer lugar, el número de octavas se ha fijado a -1, y el número de niveles por octavas ha tomado los valores de 6 (*Figura 4.14*), 8 (*Figura 4.15*) y 10 (*Figura 4.16*).

Con estas pruebas se puede observar que el número de puntos característicos coincidentes entre la imagen y el patrón apenas varía al modificar el

Figura 4.14: *Búsqueda de patrones por el software implementado en este proyecto con -1 octava, 6 niveles por octava y un valor umbral de 0.7.*

Figura 4.15: *Búsqueda de patrones por el software implementado en este proyecto con -1 octava, 8 niveles por octava y un valor umbral de 0.7.*

Figura 4.16: *Búsqueda de patrones por el software implementado en este proyecto con -1 octava, 10 niveles por octava y un valor umbral de 0.7.*

número de niveles por octava, fijando el número por octava a -1. Lo que sí es característico, es que los puntos coincidentes son distintos al cambiar el número de niveles por octava, llegando incluso a no existir ningún falso positivo si el número de niveles por octava es igual a 10. De todos modos, como el algoritmo encuentra una gran cantidad de puntos coincidentes, se puede considerar que la búsqueda del patrón ha sido satisfactoria.

A continuación se mostrarán los resultados al variar el número de niveles por octava fijando el número de octavas a 3. Para ello, este parámetro tomará los valores de 6 (*Figura 4.17*), 8 (*Figura 4.18*) y 10 (*Figura 4.19*).

Figura 4.17: *Búsqueda de patrones por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.7.*

Figura 4.18: *Búsqueda de patrones por el software implementado en este proyecto con 3 octavas, 8 niveles por octava y un valor umbral de 0.7.*

Figura 4.19: *Búsqueda de patrones por el software implementado en este proyecto con 3 octavas, 10 niveles por octava y un valor umbral de 0.7.*

El comportamiento de la variación del número de niveles por octava, para un número de octavas igual a 3, es igual que para el caso de que este último parámetro sea -1. El número total de *keypoints* coincidentes entre la imagen y el patrón es aproximadamente igual, pero los puntos coincidentes son distintos para las diferentes pruebas. Lo único, es que el número de puntos coincidentes es menor que en el caso de que el número de octavas sea -1. Pero aún así, existe un número significante de puntos como para poder determinar la búsqueda del patrón como positiva, ya que el número de falsos positivos es nulo.

Además de comparar los *keypoints* encontrados por cada prueba, también

hay que tener en cuenta los tiempos de ejecución. Se puede comprobar que el ejecutable de *Lowe*, a pesar de ser la aplicación que más *keypoints* encuentra, es también la que más tiempo tarda en procesar, estando en el alrededor de los 25 segundos en el caso más favorable. En cambio, en las pruebas realizadas con el software implementado para este proyecto con un valor de número de octavas igual a -1 (que es la configuración más lenta), apenas llega a los 10 segundos en tiempo de ejecución, llegando incluso llegar al orden del segundo si el número de octavas es 3. Estos tiempos de ejecución se pueden reducir, si en lugar de presentar los resultados en imágenes se almacenan en variables para su posterior utilización.

Además se puede ver que debido a la gran cantidad de puntos coincidentes que encuentra el ejecutable de *Lowe*, si se utiliza un valor umbral muy alto para realizar el *matching*, no se puede apreciar si se ha obtenido un resultado satisfactorio, ya que existen una gran cantidad de falsos positivos. En cambio, para el software propio, la cantidad de *keypoints* encontrados es menor, pudiéndose utilizar así un valor umbral alto, debido a que el número de falsos positivos es despreciable. Es por esto, que en las pruebas sucesivas se utilizará para el ejecutable de *Lowe* un valor umbral para el *matching* de 0.5 y 0.6, mientras que para el software propio, se fijará el parámetro de rechazo a 0.7 y 0.8.

4.2. Reconocimiento de objetos

Una vez realizadas las pruebas de búsqueda de patrones en imágenes, se procedió a realizar las pruebas de reconocimiento de objetos en imágenes. Como se pudo comprobar en la búsqueda de patrones, el ejecutable de *Lowe* funciona mucho mejor con un valor umbral de *matching* bajo y es por eso, que en este caso, todas las pruebas que se hagan con dicho ejecutable se realizarán con unos valores umbrales de 0.5 y 0.6. Mientras que el software propio funciona mucho mejor con valores umbrales altos, por eso se utilizará para dichos casos los valores de 0.7 y 0.8.

Figura 4.20: *Escena en la que se buscarán los diferentes objetos.*

El reconocimiento de objetos es parecido a la búsqueda de patrones, pero la diferencia consiste en que la imagen a buscar no se encuentra exactamente

igual en la imagen en la que se realiza la búsqueda. En estas pruebas lo que tenemos es una imagen que muestra una escena, como se representa en la *Figura 4.20*.

Los objetos que se deben encontrar en la escena mostrada en la *Figura 4.20* son los que se muestran en la *Figura 4.21*.

Figura 4.21: *Objetos que se deben encontrar en la imagen escena. De izquierda a derecha se encuentran un libro, una caja de pan, y una caja de arroz basmati.*

Como cada uno de los objetos presenta una localización diferente que el resto, se analizará, por lo tanto, la búsqueda de cada objeto de un modo distinto que al del resto. En un primer lugar se analizará la búsqueda del libro. Se puede observar que el libro se encuentra rotado en el plano de la

imagen y que además se encuentra parcialmente oculto. Esto debería reducir notablemente el número de puntos coincidentes entre la imagen del libro y la imagen de la escena.

Del mismo modo en que se realizó las pruebas de la búsqueda de patrones, primero se analizará la búsqueda que realiza el ejecutable de *Lowe*, siendo los resultados los que se muestran en la *Figura 4.22* (parámetro de rechazo igual a 0.5) y *Figura 4.23* (parámetro de rechazo igual a 0.6).

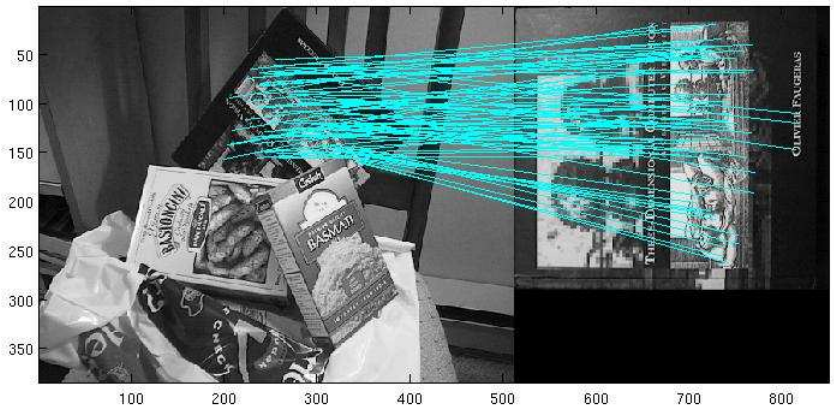

Figura 4.22: *Búsqueda de un libro en una escena con varios objetos por el ejecutable de Lowe con un valor umbral de 0.5.*

Se puede observar que existen una gran cantidad de puntos coincidentes en ambas pruebas, produciéndose un número muy bajo de falsos positivos (siendo incluso nulo para un valor de rechazo de 0.5). Por eso se puede con-

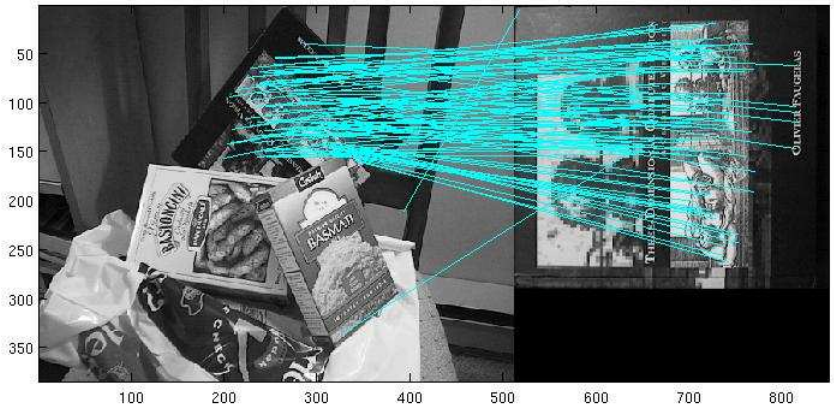

Figura 4.23: *Búsqueda de un libro en una escena con varios objetos por el ejecutable de Lowe con un valor umbral de 0.6.*

siderar que la búsqueda del libro ha resultado positiva. Es notable destacar que en el caso de búsqueda de patrones existe una mayor cantidad de falsos positivos, tanto en número como en porcentaje respecto al total de puntos coincidentes, ya que los parámetros se han mantenido constantes para ambos casos.

Una vez visto los resultados que ofrece el ejecutable de *Lowe*, se procede a mostrar el funcionamiento del software implementado para este proyecto. Para ello se realizaron diversas pruebas modificando los diferentes parámetros según los siguientes valores:

- Número de octavas: Los valores que tomará este parámetro será de -1 y 3.

- Número de niveles por octavas: Para las pruebas tomará valores de 3, 6, 8 y 10.
- Valor umbral del algoritmo de *matching*: Tal y como se indicó con anterioridad se fijará a 0.7 y 0.8

En primer lugar se mostrarán los resultados para un valor del número de octavas igual a -1 y el parámetro de rechazo será igual a 0.7. Los valores del número de niveles por octava, tal y como se ha mencionado anteriormente, serán de 3 (*Figura 4.24*), 6 (*Figura 4.25*), 8 (*Figura 4.26*) y 10 (*Figura 4.27*).

Figura 4.24: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.7.*

Se puede ver que existe una gran cantidad de *keypoints* coincidentes entre la imagen escena y la imagen del objeto a buscar, y por ello se puede

Figura 4.25: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 6 niveles por octava y un valor umbral de 0.7.*

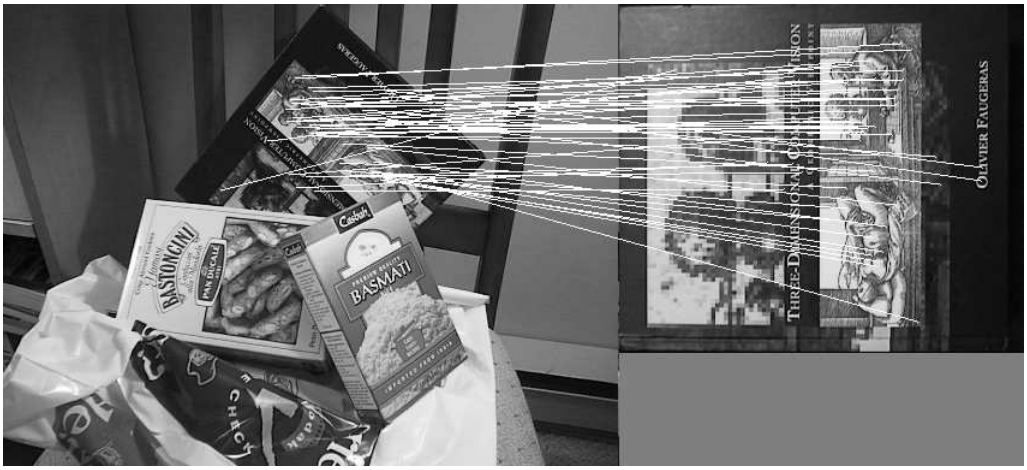

Figura 4.26: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 8 niveles por octava y un valor umbral de 0.7.*

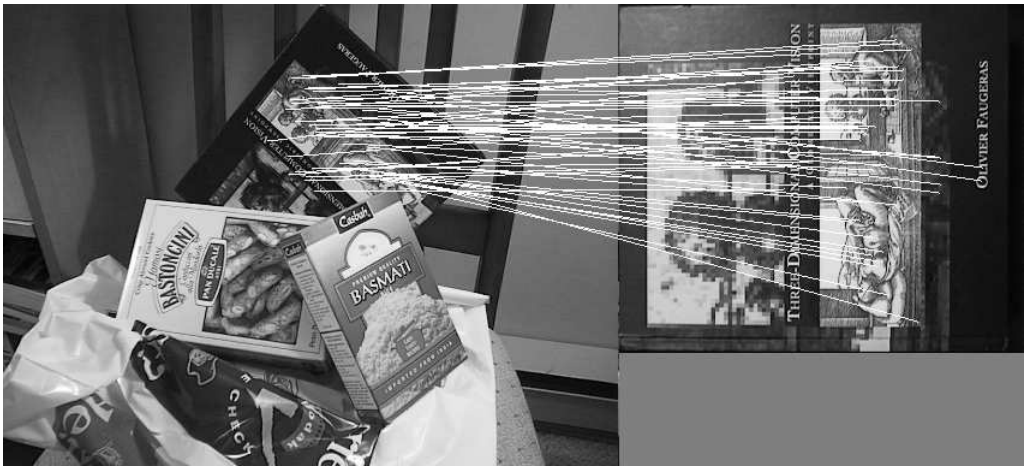

Figura 4.27: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 10 niveles por octava y un valor umbral de 0.7.*

considerar que el objeto ha sido reconocido satisfactoriamente. Además se puede observar que en ningún caso se observa ningún falso positivo, lo que indica la gran efectividad del algoritmo con estos parámetros. Otro punto que cabe destacar, es que los resultados no son muy diferentes al variar el número de niveles por octava, ya que no modifica notablemente el número de puntos coincidentes, porque simplemente varían la situación de los puntos característicos. En cambio si se nota una gran diferencia en el tiempo de ejecución, ya que el programa es mucho más rápido cuando el número de niveles por octava es menor.

A continuación se mostrarán los resultados obtenidos con un valor del parámetro

de rechazo del algoritmo de *matching* igual a 0.8. Al variar dicho parámetro cabe esperar que el número de puntos coincidentes entre ambas imágenes sea mayor, llegando incluso a la aparición de algún falso positivo. Al igual que en el caso anterior, los valores del número de niveles por octava será igual a 3 (*Figura 4.28*), 6 (*Figura 4.29*), 8 (*Figura 4.30*) y 10 (*Figura 4.31*).

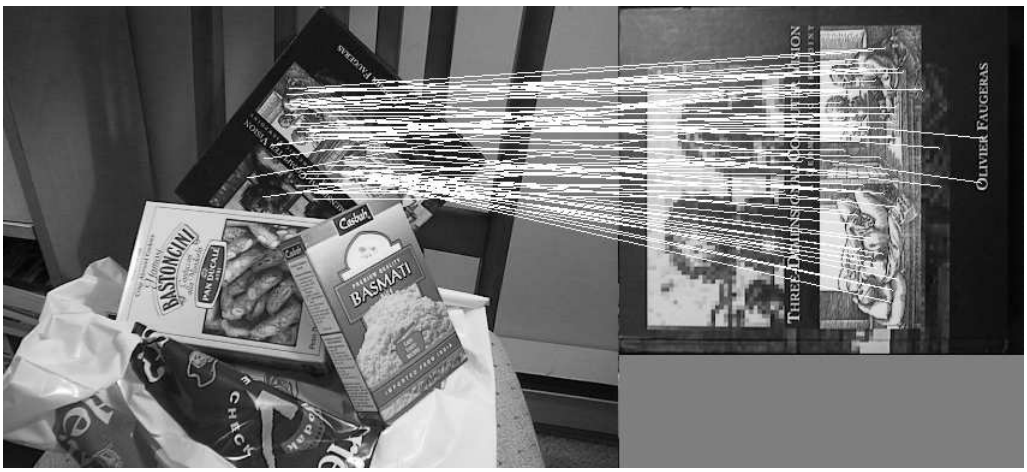

Figura 4.28: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.8.*

Se puede observar, tal y como se había indicado con anterioridad, que el número de puntos coincidentes entre ambas imágenes ha aumentado respecto al uso de un valor umbral de 0.7. También se aprecia, que a pesar de no variar visiblemente el número de *keypoints* encontrados al aumentar el número de niveles por octava, si que aparecen ciertos falsos positivos. Esto

Figura 4.29: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 6 niveles por octava y un valor umbral de 0.8.*

Figura 4.30: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 8 niveles por octava y un valor umbral de 0.8.*

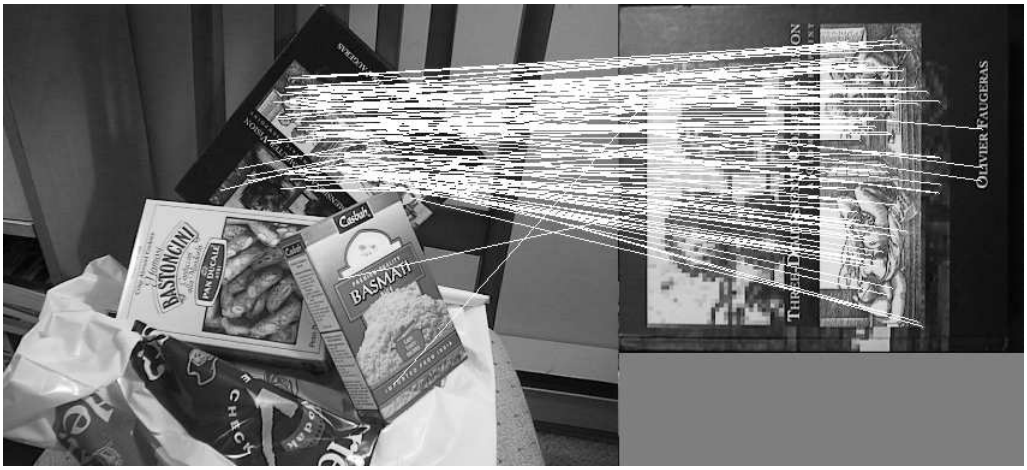

Figura 4.31: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 10 niveles por octava y un valor umbral de 0.8.*

indica que dicho parámetro no tiene excesiva relevancia cuando fijamos el número de octavas a -1. Por lo tanto, para sucesivas pruebas, el número de niveles por octavas no se tendrá en cuenta cuando el valor del número de octavas sea igual a -1.

Una vez vislumbrados los resultados fijando el número de octavas a -1, se procederá a mostrar los resultados fijando dicho parámetro a 3. Para ello, en un primer lugar, se igualará el parámetro de rechazo de la coincidencia de puntos a 0.7, variando por tanto el valor del número de niveles por octava a 3 (*Figura 4.32*), 6 (*Figura 4.33*), 8 (*Figura 4.34*), y 10 (*Figura 4.35*).

Figura 4.32: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 3 niveles por octava y un valor umbral de 0.7.*

Figura 4.33: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.7.*

Figura 4.34: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 8 niveles por octava y un valor umbral de 0.7.*

Figura 4.35: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 10 niveles por octava y un valor umbral de 0.7.*

En estos casos se puede observar que el número de puntos encontrados es muy bajo, por lo tanto no se puede considerar que se ha realizado un reconocimiento del objeto a buscar, ya que no se puede asegurar que los puntos que se encuentren siempre sean puntos correctos y no falsos positivos. Si ahora realizamos la mismas pruebas, pero fijando el valor de rechazo de la coincidencia de puntos a 0.8, se observará que el número de *keypoints* encontrado es mayor, pero quizás sea insuficiente para determinar que la búsqueda se ha hecho correctamente.

Figura 4.36: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 3 niveles por octava y un valor umbral de 0.8.*

En estos casos, el número de puntos coincidentes entre la imagen escena y la del objeto es algo mayor que para el caso de fijar el parámetro de rechazo

Figura 4.37: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.8.*

Figura 4.38: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 8 niveles por octava y un valor umbral de 0.8.*

Figura 4.39: *Búsqueda de un libro en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 10 niveles por octava y un valor umbral de 0.8.*

del algoritmo de *matching* a 0.7. Pero cabe destacar que estos resultados no siguen un orden lógico, ya que en caso de que el número de niveles por octava sea igual a 6 (*Figura 4.37*) es cuando obtenemos un menor número de puntos coincidentes, seguido de fijar dicho parámetro a 8 (*Figura 4.38*). En todos estos casos, habría cierta duda de indicar como positiva la búsqueda del libro en la escena, ya que el número de puntos coincidentes no es muy alto, pero no se encuentra entre ellos ningún falso positivo. La decisión de poder aceptar como estos resultados como positivos dependerá del uso que se quiera dar a la aplicación y de las restricciones de dicho caso.

Como resumen, se puede indicar que al fijar el número de octavas a -1, la

búsqueda del libro se puede indicar como positiva, ya que los resultados obtenidos son muy parecidos a los que se obtienen al usar el ejecutable de *Lowe*. En cambio si dicho parámetro lo fijamos a 3, si utilizamos un valor umbral de 0.7, no se puede indicar que dicha búsqueda se ha realizado satisfactoriamente. Mientras que si dicho valor es de 0.8, el resultado de la búsqueda quedará en función del uso que se le dé a dicho software.

A continuación se muestran los resultados obtenidos al realizar la búsqueda de la caja de pan. Dicho objeto, al igual que el libro, se encuentra tanto parcialmente oculto, como girado en el plano de la imagen. Pero además, dicho objeto también se encuentra girado en otro plano distinto al de la imagen, por lo tanto, la imagen queda parcialmente deformada, haciendo más complicada su identificación. En un primer lugar se mostrarán en la *Figura 4.40* (Parámetro de rechazo igual a 0.5) y en la *Figura 4.41* (Parámetro de rechazo igual a 0.6) los resultados obtenidos con el ejecutable de *Lowe*.

Como se puede observar en la *Figura 4.40* y la *Figura 4.41*, el ejecutable de *Lowe* encuentra perfectamente la caja de pan, ya que el número de *keypoints* encontrado es muy elevado, existiendo únicamente un falso positivo. Es destacable que los resultados obtenidos para la búsqueda de la caja de pan son muy parecidos a la búsqueda del libro, ya que la caja de pan se encuentra un poco deformada a causa del cambio del punto de vista de la

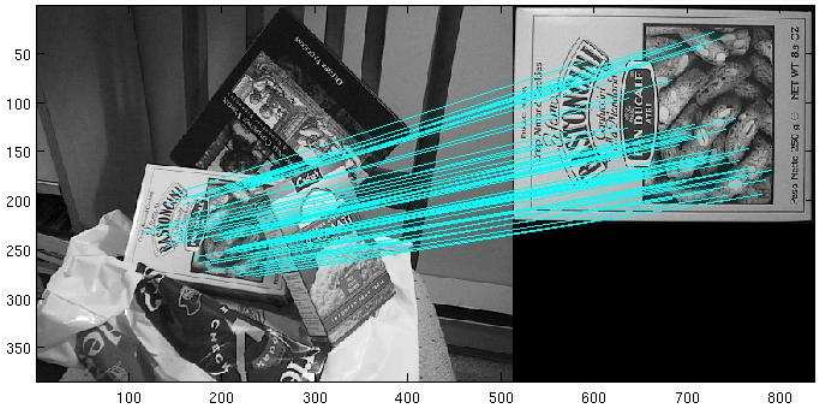

Figura 4.40: *Búsqueda de una caja de pan en una escena con varios objetos por el ejecutable de Lowe y un valor umbral de 0.5.*

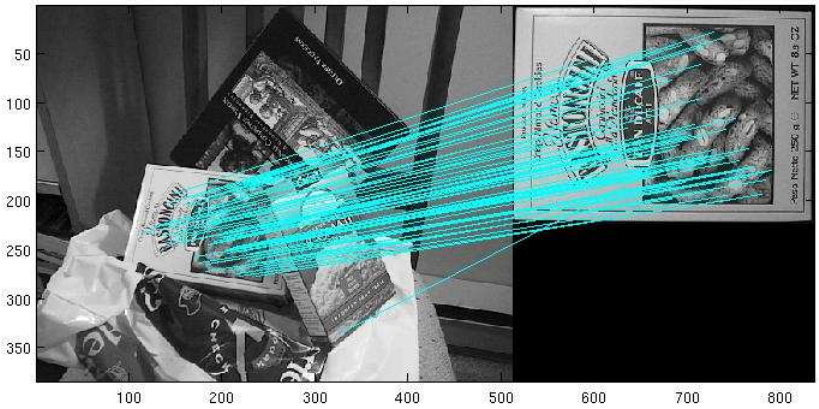

Figura 4.41: *Búsqueda de una caja de pan en una escena con varios objetos por el ejecutable de Lowe y un valor umbral de 0.6.*

cámara, y por lo tanto el número de *keypoints* encontrados debería ser mucho menor y el número de falsos positivos mayor. Aunque esto puede ser así debido a que la caja de pan presenta una oclusción mucho menor que en el caso del libro.

A continuación se muestran los resultados obtenidos con el software implementado para este proyecto. En un primer lugar, se realizarán los cálculos fijando el número de octavas a -1, el número de niveles por octava será igual a 3 y el valor umbral de rechazo de la coincidencia de puntos se fijarán tanto a 0.7 (*Figura 4.42*) y a 0.8 (*Figura 4.43*).

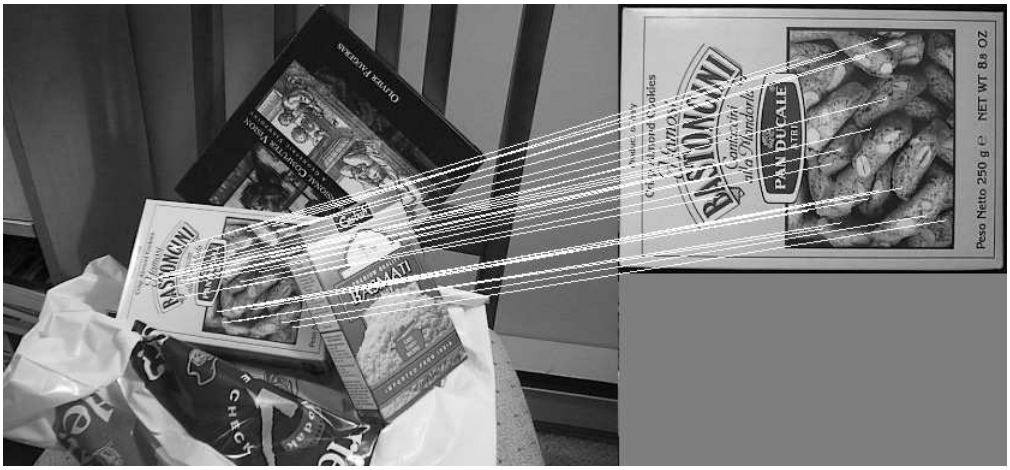

Figura 4.42: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.7.*

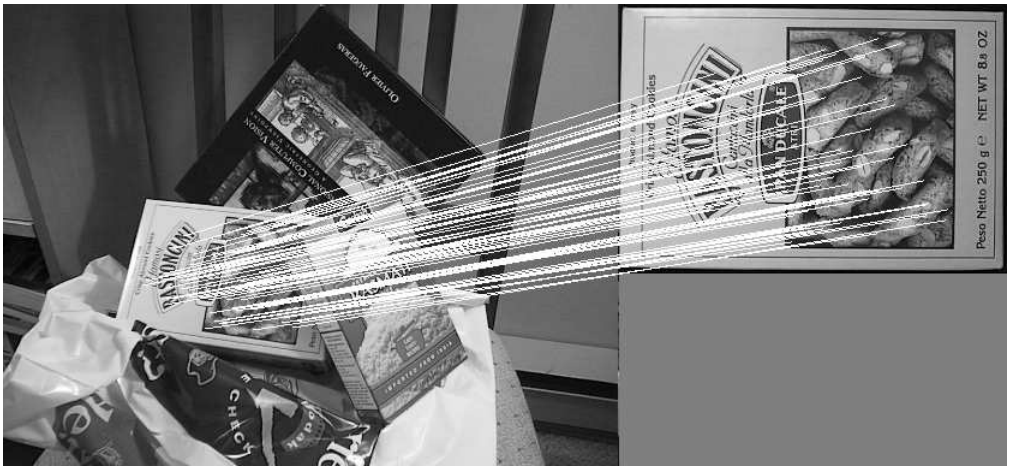

Figura 4.43: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.8.*

Se puede observar que en ambas pruebas la caja de pan ha sido encontrada satisfactoriamente, ya que existe una gran cantidad de puntos coincidentes entre la imagen escena y la de la caja de pan. Además de de que el número de *keypoints* encontrados es alto, entre éstos no existe ningún falso positivo, haciendo ver que el algoritmo es bastante eficiente.

Una vez comprobado que si se fija el número de octavas a -1, el algoritmo encuentra perfectamente la caja de pan, se procederá a mostrar los resultados igualando dicho parámetro a 3. Para ello se mantendrá en primer lugar el valor umbral del algoritmo de *matching* a 0.7, mientras que el número de niveles por octava se irá modificando a 3 (*Figura 4.44*), 6 (*Figura 4.45*), 8

(*Figura 4.46*) y 10 (*Figura 4.47*).

Figura 4.44: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 3 niveles por octava y un valor umbral de 0.7.*

Se puede ver con claridad que en ninguno de los cuatro casos se ha realizado un reconocimiento satisfactorio, al igual que pasaba con la búsqueda del libro. Además, cabe destacar que en los casos de que el número de niveles por octava es igual a 3, 6 y 8, los resultados son exactamente idénticos, variándose nada más en la existencia de un punto más en el caso de que dicho parámetro sea igual a 10.

Visto que si fijamos el parámetro de rechazo a 0.7 no se realiza una búsqueda satisfactoria, tal y como pasaba en la búsqueda del libro, se procedió a

Figura 4.45: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.7.*

Figura 4.46: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 8 niveles por octava y un valor umbral de 0.7.*

Figura 4.47: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 10 niveles por octava y un valor umbral de 0.7.*

averiguar si para un valor de 0.8 obteníamos los mismos resultados que se obtienen cuando se busca el libro en la imagen escena, siendo estos resultados los que se muestran en las *Figuras 4.48, 4.49, 4.50 y 4.51.*

En este caso se puede ver que los resultados mostrados tienen mayor coherencia que en el caso de la búsqueda del libro, ya que al aumentar el número de niveles por octava, el número de puntos coincidentes entre ambas imágenes también aumenta. Pero tal y como pasaba en el caso de la búsqueda del libro, el número de *keypoints* encontrado no es suficientemente elevado como para determinar, sin riesgo alguno, que la búsqueda se ha realizado de un modo satisfactorio. Pero debido a que el número de falsos positivos es

Figura 4.48: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 3 niveles por octava y un valor umbral de 0.8.*

Figura 4.49: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.8.*

Figura 4.50: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 8 niveles por octava y un valor umbral de 0.8.*

Figura 4.51: *Búsqueda de una caja de pan en una escena con varios objetos por el software implementado en este proyecto con 3 octavas, 10 niveles por octava y un valor umbral de 0.8.*

nulo y el tiempo de ejecución es muy bajo, no se puede descartar el uso de este software con estos parámetros. Es por ello que se debe comprobar que, para el uso que se quiera dar a dicho software con estos parámetros, los resultados obtenidos son suficientemente buenos como para determinar la búsqueda como positiva.

Por todo lo visto en los resultados que se han conseguido al buscar la caja de pan, se puede indicar que al fijar el número de octavas a -1, la búsqueda se puede indicar como positiva, ya que los resultados obtenidos son muy parecidos a los que se obtienen al usar el ejecutable de *Lowe*, pero utilizando para ello un gasto de tiempo mucho menor. En cambio si dicho parámetro lo fijamos a 3, si utilizamos un valor umbral de 0.7, no se puede indicar que dicha búsqueda se ha realizado satisfactoriamente. Mientras que si dicho valor es de 0.8, el resultado de la búsqueda quedará en función del uso que se le dé a dicho software.

Ya por último, se mostrarán los resultados obtenidos con la búsqueda de la caja de arroz basmati. Este caso es bastante interesante de estudiar, ya que el objeto a buscar está deformado notablemente, debido a que el punto de vista de la cámara desde el cual se ha obtenido la imagen del objeto ha sido modificado. Según *Lowe* [3] [4] y Humpire Mamani [2], el método es apto siempre que el objeto esté girado hasta 15º. En este caso se puede observar

que el objeto está girado bastante más de 15º, por lo tanto no se debería obtener un positivo en la búsqueda del objeto. Pero si observamos las *Figuras 4.52 y 4.53*, se puede observar que el ejecutable de *Lowe* sí encuentra una correspondencia entre ambas imágenes.

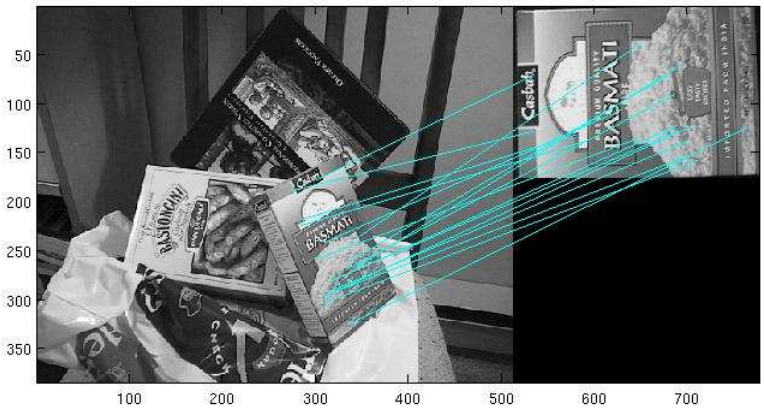

Figura 4.52: *Búsqueda de una caja de arroz basmati en una escena con varios objetos por el ejecutable de Lowe y un valor umbral de 0.5.*

Es interesante ver que el número de puntos característicos que se encuentran es bastante elevado, y por lo tanto se puede considerar que la búsqueda de la caja de arroz basmati se ha realizado con éxito, a pesar de la existencia de varios falsos positivos.

Si ahora se realiza la misma prueba con el software implementado para este proyecto, se podrá ver resultados de lo más variopinto. En primer lugar, si

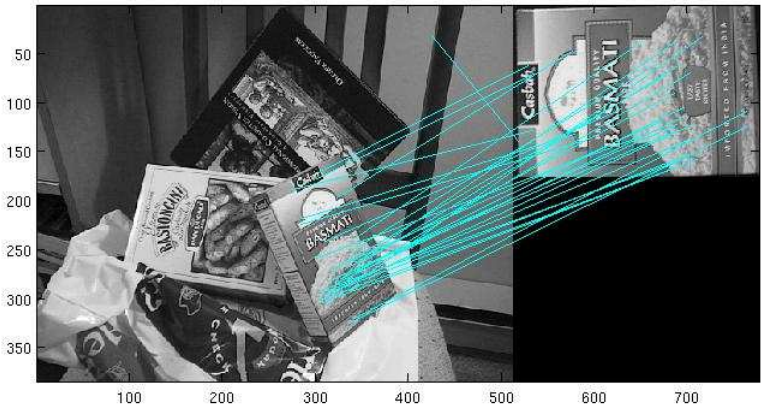

Figura 4.53: *Búsqueda de una caja de arroz basmati en una escena con varios objetos por el ejecutable de Lowe y un valor umbral de 0.6.*

se fija el número de octavas a -1, tanto para un valor umbral de 0.7 (*Figura 4.54*) como de 0.8 (*Figura 4.55*), al igual de lo que sucedía con el ejecutable de *Lowe*, el algoritmo encuentra una gran cantidad de puntos coincidentes entre ambas imágenes, estableciendo así la búsqueda de la caja de arroz basmati como positiva. Pero en el caso del uso del software propio de este proyecto, el número de falsos positivos es bastante menos importante.

En cambio, si realizamos la misma búsqueda igualando el número de octavas a 3, para cualquier valor del número de niveles por octava como del parámetro de rechazo del algoritmo de *matching*, se observa que la búsqueda no se realiza satisfactoriamente, ya que no encuentra ningún punto coincidente entre la imagen de la escena con la de la caja de arroz basmati,

Figura 4.54: *Búsqueda de una caja de arroz basmati en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.7.*

Figura 4.55: *Búsqueda de una caja de arroz basmati en una escena con varios objetos por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.8.*

mostrándose estos resultados en la *Figura 4.56*.

Figura 4.56: *Búsqueda de una caja de arroz basmati en una escena con varios objetos por el software implementado en este proyecto con 3 octavas.*

En este caso se puede observar, que debido a que el objeto está girado más de 15º, si configuramos el software implementado para este proyecto con un número de octavas igual a 3, no funciona, ya que no encuentra ninguna coincidencia entre las imágenes. En cambio, tanto el ejecutable de *Lowe* como si configuramos el software propio con un número de octavas igual a -1, si realizan una búsqueda correcta.

Como resumen del reconocimiento de objetos, se puede indicar que el software propio tiene dos usos:

1. Si el número de octavas es igual a -1. En este caso el software funciona

perfectamente, reconociendo los objetos en casos de oclusión, rotación e incluso cambio de punto de vista de la cámara. Pero el tiempo de ejecución es bastante alto como para implementarse en un sistema de tiempo real (a pesar de que el tiempo de ejecución sea menor que en el caso del ejecutable de *Lowe*). Es por ello que se debería usar en sistemas de calidad o de comprobación.

2. Si el número de octavas es igual a 3. En este caso el software no funciona bien en todos los casos, ya que si el cambio del punto de vista de la cámara es excesivamente alto (giro superior a los 15º), el algoritmo no encuentra correspondencia alguna. Pero el tiempo de ejecución con estos parámetros es bastante bajo, llegándose a poder usar en un sistema de tiempo real si se rebaja un poco más su tiempo de ejecución, ya sea mediante una mejora del equipo en el cual se ejecuta dicho software o mediante una paralelización con la tarjeta gráfica.

4.3. Producción imágenes panorámicas

Tras las pruebas de búsqueda de patrones y de reconocimiento de objetos, se procedió a realizar unas pruebas para comprobar si este algoritmo podría usarse para producir imágenes panorámicas. Para dichas pruebas se realizaron dos fotos en la biblioteca del campus de Leganés de la Univer-

sidad Carlos III de Madrid, las cuales muestran dos partes distintas de la sala, pero mostrando zonas comunes.

Figura 4.57: *Imágenes tomadas en la biblioteca del campus de Leganés de la Universidad Carlos III de Madrid.*

En las imágenes mostradas en la *Figura 4.57* se puede observar que están tomadas desde ángulos distintos. Además, dichas imágenes tienen una luminosidad distinta, lo cual hace que su búsqueda sea más complicada. Las pruebas que se han realizado pretenden buscar las zonas comunes a ambas imágenes, para poder así averiguar en qué zonas de las imágenes se debe hacer la fusión para formar una imagen panorámica. Además, en una de las fotos se puede observar que aparece la pantalla de un portátil en primer plano. La existencia de este objeto es una prueba extra para comprobar la inmunidad de dicho algoritmo a la presencia de objetos distintos en las imágenes.

En este caso se observa que las zonas comunes son unas ventanas, un cuadro y una mesa con un estudiante sentado. Para dar por satisfactoria la prueba, el algoritmo deberá indicar *keypoints* coincidentes en las zonas comunes de ambas imágenes, sin obtener puntos en las zonas que no se tienen en común. Por lo tanto, el algoritmo deberá reconocer tanto las ventanas en común, como el cuadro, la mesa y el estudiante que aparecen en la zona común a ambas imágenes.

En primer lugar, al igual que en todas las pruebas anteriores, se realizaron los primeros cálculos con el ejecutable de *Lowe*, con unos valores umbrales de 0.5 y 0.6, siendo los resultados obtenidos los mostrados en las *Figuras 4.58* y 4.59.

Se observa que el descriptor ha reconocido los objetos en común de las dos imágenes, mostrando la coincidencia de bastantes *keypoints*, llegando incluso a reconocer los fluorescentes del techo. El problema que tiene es que ha realizado el *matching* en ciertas ventanas de manera errónea, para el caso de usar un valor umbral de 0.6. Este error es bastante importante, ya que se consigue que exista una incertidumbre en la selección de la zona común para poder realizar la fusión de ambas imágenes. También se observa que el número de *keypoints* encontrado no es excesivamente elevado, teniendo en

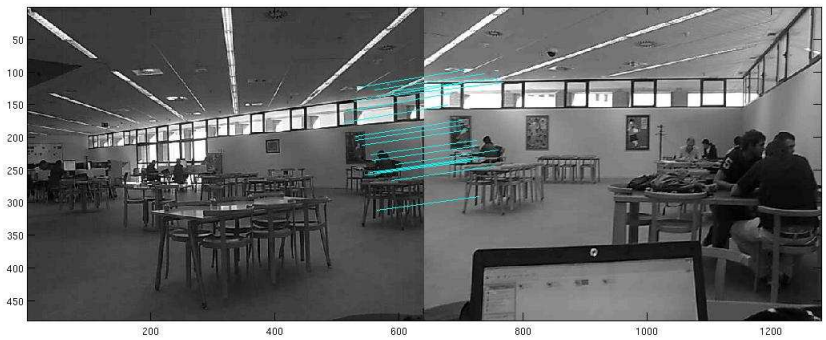

Figura 4.58: *Producción de imágenes panorámicas por el ejecutable de Lowe con un valor umbral de 0.5.*

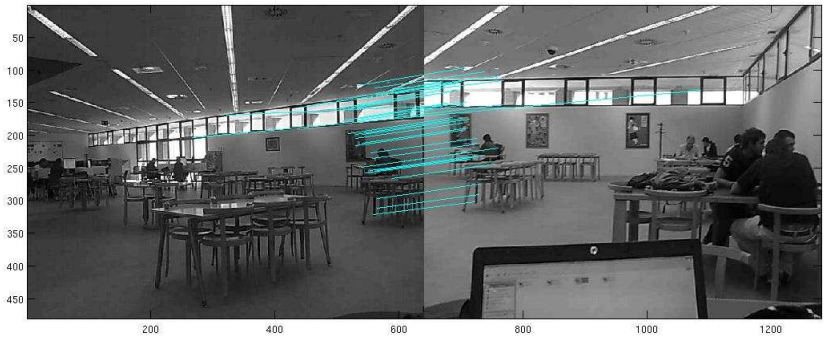

Figura 4.59: *Producción de imágenes panorámicas por el ejecutable de Lowe con un valor umbral de 0.6.*

Figura 4.60: *Producción de imágenes panorámicas por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.7.*

cuenta los resultados de las pruebas anteriores. Este resultado es un dato interesante, porque las imágenes usadas para estas pruebas poseen una resolución mayor que las usadas en las pruebas anteriores, pudiéndose calcular así un mayor número de *keypoints*. Además, las imágenes tienen numerosos bordes en los cuales se pueden obtener muchos *keypoints*.

Para poder comparar estos resultados con el software implementado para este proyecto, en las *Figuras 4.60 y 4.61* se muestran los resultados obtenidos con un número de octavas igual a -1.

Se puede observar que la zona común ha sido determinada de una forma inequívoca, ya que al igual de lo sucedido con el ejecutable de *Lowe*, el software

Figura 4.61: *Producción de imágenes panorámicas por el software implementado en este proyecto con -1 octava, 3 niveles por octava y un valor umbral de 0.8.*

implementado para este proyecto ha detectado todos los objetos presentes en la zona común a ambas imágenes. Pero en este caso, para cualquier valor umbral probado, no se ha producido ningún falso positivo, por lo que la zona queda perfectamente determinada.

Una vez visto el funcionamiento de dicho software con el parámetro de número de octavas igual a -1, se procedió a cambiarlo a 3. Como se ha podido ver con anterioridad que el número de niveles por escalas no afecta de una forma notable al número de *keypoints* coincidentes, incluso para un número de octavas igual a 3, pues se ha procedido a hacer estas pruebas fijando dicho parámetro a 6. Por lo tanto se han realizado las pruebas modificando únicamente el parámetro de rechazo del algoritmo de *matching* entre

los valores de 0.7 (*Figura 4.62*) y 0.8 (*Figura 4.63*).

Figura 4.62: *Producción de imágenes panorámicas por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.7.*

Para el caso de usar un valor umbral igual a 0.7 (*Figura 4.62*) se puede comprobar que no se encuentra la zona común, ya que sólo se ha realizado con coincidencia de un sólo punto. En cambio para un valor umbral de 0.8 (*Figura 4.63*), se observa que el número de *keypoints* coincidentes entre ambas imágenes es bastante reducido, pero describen perfectamente la zona común, ya que reconoce como común tanto al estudiante como a la mesa en la cual se encuentra. Además no existe ninguna falsa correspondencia entre las zonas de ambas imágenes, por lo tanto queda perfectamente detallada la zona donde se puede realizar la fusión entre las imágenes.

Figura 4.63: *Producción de imágenes panorámicas por el software implementado en este proyecto con 3 octavas, 6 niveles por octava y un valor umbral de 0.8.*

Al contrario de lo que pasaba en las pruebas anteriores, para la formación de imágenes panorámicas, el tiempo de ejecución no es un factor determinante, ya que no se implementará en ningún sistema en tiempo real. Por lo tanto lo más importante es que la zona común quede determinada de forma inequívoca, que es lo que ocurre cuando se trabaja con el software implementado para este proyecto con un número de octavas igual a -1, porque, tanto si dicho parámetro se fija a 3 o si se utiliza el ejecutable de *Lowe*, existen casos en los que la zona común no queda determinada, ya sea por falta de puntos coincidentes o por la existencia de falsos positivos que hacen que exista una incertidumbre en la determinación de la zona.

Capítulo 5

Conclusiones y posibles mejoras

Una vez mostrados todos los resultados se pueden sacar varias conclusiones:

- En primer lugar, se ha podido descubrir que este algoritmo es inmune a la iluminación ya que en las pruebas de la formación de imágenes panorámicas, las fotografías tenían distinta iluminación y aún así el software ha conseguido determinar la zona común a ambas.

- Se ha comprobado también que el algoritmo es muy eficiente, ya que el número de falsos positivos es muy bajo, produciendo que la tasa de error sea muy baja.

- También hay que indicar que el software responde perfectamente a los cambios de rotación y de escala. Además funciona bien en casos de oclusión de objetos.

- Dependiendo de los ajustes de los parámetros, el software responde

muy bien ante los cambios de vista de la cámara. El coste para conseguir esto es un aumento notable del tiempo de ejecución.

- Además, el tiempo de ejecución, a pesar de ser bastante menor que el necesitado usando el ejecutable de *Lowe*, sigue siendo elevado para ser usado en aplicaciones de tiempo real, a no ser que se mejore la máquina en la cual se ejecute.

Por todo esto, se pueden modificar ciertos aspectos para conseguir mejores resultados, como son los que se enumeran a continuación:

- Se puede modificar el método de *matching* usado, ya que dicho método utiliza funciones trigonométricas inversas, las cuales hacen aumentar el tiempo de ejecución, pero de manera poco significativa, ya que la mayor parte del cálculo se implica en el módulo del descriptor.

- También se puede implementar un descriptor basado en el descriptor SIFT denominado SURF (Speeded Up Robust Features), el cual es mucho más rápido que el método SIFT, ya que los *keypoints* contienen muchos menos descriptores debido a que la mayor cantidad de los descriptores son 0. Este descriptor se puede considerar una mejora debido a que la modificaciones que supondría en el código no serían excesivas, ya que el descriptor SURF utiliza la gran mayoría de las funciones que utiliza el descriptor SIFT.

Apéndice A

Manual de uso

En este apéndice se pretende explicar el uso de este software. Este manual de uso es válido para máquinas Linux, ya que este proyecto ha sido desarrollado en un equipo en el cual estaba instalado Ubuntu 8.10.

Los pasos a seguir son los que a continuación se enumeran:

1. Establecer el valor de los parámetros correspondientes del descriptor. Para ello hay que editar el archivo 'descript.cpp' para ajustar los valores de octaves (número de octavas) y levels (número de niveles por octava) y editar el archivo 'match.cpp' para ajustar el valor de distaratio (valor umbral del *matching*). Para establecer el número de octavas a -1, hay que igualar el parámetro octaves y first a -1. Mientras que si se quiere establecer el número de octavas a 3, al parámetro first hay que adjudicarle el valor de 1 y al parámetro octaves el valor de 3.

2. Compilar el código mediante el *Makefile* adjunto.

3. Para ejecutar el programa escribir el siguiente comando:

```
$>./match 'Imagen1.PGM' 'Imagen2.PGM'
```

   Siendo Imagen1.PGM la imagen en la cual se desea buscar el objeto o el patrón y la Imagen2.PGM sería la imagen del objeto o del patrón.

4. Por último, para ver los resultados obtenidos, sólo es necesario abrir la imagen generada con el nombre 'resultado.pgm'

Apéndice B

Compilación del código

En este apéndice se pretende explicar todas las cuestiones necesarias para compilar este software en un ordenador el cual tiene instalado un sistema operativo Linux.

En un primer momento hay que tener en cuenta las siguientes indicaciones:

- Este código tiene módulos programados en C y C++, por lo tanto en la máquina que se desee compilar debe tener instaladas las bibliotecas propias de estos lenguajes de programación, como son 'math.h', 'string', 'algorithm', 'limits', 'iostream', 'fstream' y 'sstream'.
- Además se deben de disponer de los diez archivos que componen el software completo cuyos nombres son los que a continuación se indican:
  - 'descript.cpp'
  - 'descript.h'

- 'dibujar.cpp'
- 'gest_struct.h'
- 'match.cpp'
- 'sift.cpp'
- 'sift.hpp'
- 'sift.ipp'
- 'sift-conv.tpp'

- También se debe disponer de un archivo *Makefile* el cual se ejecutará con la siguiente instrucción:

```
$>make match
```

Con todo esto el código se compilará sin ningún error.

Bibliografía

[1] De la Escalera, Arturo "Visión por computador". Prentice Hall, 2006.

[2] Humpire Mamani, Gabriel E. "Extracción de características en imágenes digitales". 1 de junio de 2009.

[3] Lowe, David G. "Distinctive Image Features from Scale-Invariant Keypoints". 5 de enero de 2004.

[4] Lowe, David G. "Object Recognition from Local Scale-Invariant Features".

[5] Mikolajczyk, Krystian y Schmid, Cordelia "A performance evaluation of local descriptors". IEEE Transactions on Pattern Analysis & Machine Intelligence, 2005.

[6] Vedaldi, Andrea. "An open implementation of the SIFT detector and descriptor".